Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

Гавриляк М.С.

ПРОГРАМУВАННЯ МЕРЕЖНИХ ПОСЛУГ

Навчальний посібник

Чернівці
Чернівецький національний університет
2022

УДК 655.3

Гавриляк М.С.

Програмування мережних послуг / автор.:, М.С. Гавриляк– Чернівці: Чернівец. нац. ун-тет, 2022, с. 179

Навчальний посібник призначений для формування у студентів системи теоретичних знань, прикладних умінь та практичних навичок з базових технологій програмування в мережах, основ ієрархічної структури об’єктів, об’єктно-орієнтованих концепцій програмування, а також формування твердих практичних навичок щодо розробки якісних мережних додатків. Навчальний посібник призначений для студентів вищих закладів освіти за спеціальністю №172 Телекомунікації та радіотехніка.

УДК 655.3

Зміст

Вступ

В посібнику наведено матеріал, що допомагає засвоїти всі лекційні теми навчальної дисципліни. Подано велику кількість ілюстративного матеріалу у вигляді відповідних діалогових вікон, рисунків і прикладів; більшу частину рисунків наведено в графічному вигляді, що істотно полегшує розуміння досліджуваних питань з основ мережевого програмування на мовах PHP та Java. Наведено приклади, спрямовані на підвищення ефективності процесу створення клієнт-серверних програм та додатків. Окремим розділом представлено лабораторний практикум, що призначений для закріплення теоретичних знань шляхом виконання конкретних практичних завдань.

Мета навчальної дисципліни: формування та засвоєння студентами базових технологій програмування в мережах, основ ієрархічної структури об'єктів, об'єктно-орієнтованих концеп-цій програмування, а також формування твердих практичних навичок щодо розробки якісних мережних додатків. Дисципліна логічно поєднана з курсами: «Інформатика», «Захист інформації в телекомунікаційних системах та мере-жах», «Телекомунікаційні та інформаційні мережі».

Студент повинен набути наступних **компетентностей та практичних навичок**:

Здатність розв'язувати складні задачі і проблеми під час професійної діяльності у галузі телекомунікації та радіотехніки й у суміжних областях (приладобудування, оптичний зв'язок, телемедицина, екологія тощо) або у процесі навчання за програмами вищого рівня, що передбачає проведення досліджень та/або здійснення інновацій та характеризується невизначеністю умов і вимог.

Здатність демонструвати і використовувати фундаментальні знання принципів побудови сучасних інфокомунікаційних систем та мереж, радіотехнічних систем та пристроїв, волоконно-оптичних та інших оптичних систем контролю, керування, збереження та перетворення інформації, перспективні напрямки їх розвитку.

Здатність самостійно проектувати інформаційно-телекомунікаційні мережі та системи із залученням фундаментальних знань оптичних та оптико-електронних систем та їх елементів з урахуванням усіх аспектів поставленої задачі.

Здатність оцінювати доцільність та можливість застосування нових методів і технологій в задачах синтезу інформаційних мереж та систем накопичення, збереження й обробки та захисту інформації.

Здатність використовувати типові та розробляти власні програмні продукти, орієнтовані на розв'язок задач проектування та розрахунок складових частин інформаційних мереж для оптимізації структури та синтезу нових інфокомунікаційних послуг.

Розділ 1. БАЗОВІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ В МЕРЕЖАХ

1.1. Поняття про розподілені інформаційні системи

На сьогоднішній день особливо актуальними є розподілені інформаційні системи. Це системи, головна ознака яких – наявність кількох центрів оброблення даних, що дає змогу виконувати паралельні обчислення. В загальному випадку вони є взаємозв'язаним набором автономних комп'ютерів, процесів або процесорів. Так як вузлами системи можуть бути процеси, то вони включають в себе і програмні засоби. В більшості випадків розподілена система складається з декількох процесорів, сполучених комутуючою апаратурою. Крім розподілених відомі ще і монолітні системи.

Розподілені інформаційні системи – децентралізовані системи, як правило гетерогенні. На противагу ним – монолітні системи працюють на одному пристрої, який не взаємодіє з сервером та іншими пристроями.

Розподілені системи виникли три десятиліття тому. При побудові інформаційних систем популярною була модель "хост-комп'ютер + термінали", що реалізовувалась на базі мейнфреймів (наприклад, IBM-360/370, або їх вітчизняних аналогів – комп'ютерів серії ЄС ЕОМ), або на базі так званих МІНІ-ЕОМ. Характерною особливістю такої системи була повна "неінтелектуальність" терміналів, що використовувались як робочі місця. Їх роботою керував хост-комп'ютер.

В нашому університеті така система функціонувала в 80÷90-их роках. На базі двох ЕОМ ЄС-1055 в режимі PRIMUS студентами на персональних терміналах в аудиторіях 210-1 та 214-1 розв’язували свої індивідуальні задачі на різних алгоритмічних мовах (FORTRAN, PL-1, Асемблер). Результати своїх розробок вони друкували на системних алфавітно-цифрових друкуючих пристроях, а графічну інформацію виводили на системний планшетний плотер.

Цей підхід мав безперечні на той час переваги. По-перше, користувачі такої системи могли спільно використовувати різні ресурси хост-комп'ютера (оперативну пам'ять, процесор) і досить дорогі для тих часів периферійні пристрої (друкуючі та графічні пристрої, пристрої введення з магнітних стрічок і гнучких дисків, дискові накопичувачі). Програмне забезпечення, яке тоді використовувалось, мало справу тільки з "локальними" ресурсами – з локальною файловою системою, локальною оперативною пам'яттю і т.д. Бурхливе зростання індустрії персональних комп'ютерів, що почалося, спочатку мало що змінило в ідеології побудови програмних систем – як і раніше в більшості своїй програми мали справу з локальними ресурсами. Правда, частина цих ресурсів була вже "псевдолокальною". Наприклад, файли на мережевому диску, хоч, як і раніше оброблялись безпосередньо самим вузлом, але при цьому він спочатку передавався по мережі. Виявилося, що традиційні підходи не працюють за таких умов. При збільшенні обсягу даних, що перероблюються, а також по мірі зростання їхньої вартості, прийшли до висновку, що довіряти їх обробку

клієнтським машинам не можна – будь-яка помилка на них (а чим більше клієнтів, тим більше вірогідність помилки) приводить або до втрати цілісності даних, або до їх блокувань в процесі роботи. Це в свою чергу означає зниження загальної продуктивності системи.

Наступним ключовим кроком стало розповсюдження ідеології клієнт-серверної обробки. Це були системи, в яких клієнт ніс відповідальність за відображення інтерфейсу користувача і виконання коду додатку, а роль сервера зазвичай доручалася системі управління базами даних (СУБД). Наприклад, замість того, щоб читати файл цілком і обробляти його, машина-клієнт передає машині-серверу запит, де вказує, яким чином файл має бути оброблений. Сервер обробляє запит клієнта і повертає йому результат.

З появою і подальшим ускладненням подібних систем особливої актуальності набуло поняття «програмного рівня».

Концепція рівнів – одна із загальновживаних моделей, що використовувались розробниками програмного забезпечення для розбиття складних систем на більш прості частини. В архітектурі комп'ютерних систем, наприклад, розрізняють рівні коду на мові програмування, функцій операційної системи, драйверів пристроїв, наборів інструкцій центрального процесора і внутрішньої логіки чіпів. У середовищі мережевої взаємодії протокол FTP працює на основі протоколу TCP, який, в свою чергу, функціонує «поверх» протоколу IP, розташованого «над» протоколом Ethernet.

При такому підході виділяють верхній рівень, що описує систему в цілому (внаслідок того, що в ньому ігнорують більшість малозначущих деталей). Під ним розташовується рівень, на якому робиться опис (реалізація) операцій, що використовуються на верхньому рівні і так далі. Таким чином, якщо дивитися знизу вгору, кожен рівень, розміщений нижче, забезпечує функціональність, яку використовує рівень, що розміщений вище.

Розбиття системи на рівні надає цілий ряд переваг.

- окремий рівень можна сприймати як одне ціле;
- можна вибирати альтернативну реалізацію базових рівнів;
- залежність між рівнями можна звести до мінімуму;
- більш нижчий рівень може одночасно служити основою для декількох більш вищих рівнів. Інакше для кожного протоколу високого рівня довелося б створювати власний протокол низького рівня;

Проте концепція розбиття на рівні має і певні недоліки. А саме:

- у випадку модифікації одного рівня часом потрібно внесення каскадних змін в інші;
- наявність надмірних рівнів знижує продуктивність системи, тому що при переході від одного рівня до іншого представлення моделі піддається відповідним перетворенням.

Проте найскладнішим при використанні архітектурних рівнів є визначення вмісту і меж відповідальності кожного рівня.

Розповсюдження технології клієнт – сервер допомогло вирішити багато старих проблем, але при цьому створило і багато нових. Однією з основних було

і залишається визначення межі між функціональністю клієнта та сервера. На практиці досить часто перенесення частини завдань з клієнта на сервер негативно позначається на загальній продуктивності системи. В той же час, перенесення частини навантаження з сервера до клієнта може привести до втрати централізації.

З ростом популярності систем клієнт/сервер набирала силу і технологія об'єктно-орієнтованого програмування, яка пропонувала перейти до системної архітектури з трьома рівнями: рівень представлення відводиться призначеному для користувача інтерфейсу, рівень предметної області призначений для опису основних функцій додатку, необхідних для досягнення поставленої перед ним мети, а третій рівень представляє джерело даних.

1.2. Архітектура клієнт-сервер

З появою поняття Web актуальними стали системи клієнт/сервер, де в ролі клієнта виступав би web–браузер. Інструментальні засоби конструювання Web–сторінок були у меншій мірі пов'язані з мовою баз даних SQL і тому більш підходили для реалізації третього рівня.

В даний час можна вважати, що бум технологій, пов'язаних з клієнт–серверною архітектурою все ще продовжується. Більшість сучасних інформаційних систем виконана за цією технологією.

Розподілені системи сьогодні досить широко використовуються, так як їх застосування є більш ефективним, ніж використання монолітних. Наведемо їх переваги.

Обмін інформацією. Необхідність обміну даними між різними вузлами зросла в шестидесятих, коли більшість основних університетів і компаній почали користуватися своїми власними мейнфреймами. Взаємодія між людьми з різних організацій полегшилася завдяки обміну даними між комп'ютерами цих організацій, і це дало зростання розвитку так званих глобальних мереж (WAN).

Комп'ютерна система, сполучена в глобальну мережу, як правило забезпечувалася всім необхідним користувачеві: резервними сховищами даних, дисками, багатьма застосовними програмами та принтерами.

Пізніше комп'ютери стали більш компактиними та дешевшими. Сьогодні одна організація в більшості випадків має багато комп'ютерів – часто один комп'ютер на одного працівника (робочу станцію). В цьому випадку також доцільно, щоб комп'ютери були сполучені для електронного обміну інформацією між персоналом фірми.

Розподілення ресурсів. З приходом більш дешевих комп'ютерів стало можливим забезпечення кожного співробітника організації особистим комп'ютером, але це не завжди можливо або доцільно для інших ресурсів (принтери, резервні сховища, блоки дисків і так далі). У цьому випадку кожен комп'ютер може використовувати спеціалізовані вузли – сервери, які забезпечують його даними і надають доступ до спеціалізованих ресурсів. Мережа, що сполучає комп'ютери в масштабі підприємства називається локальною обчислювальною мережею (LAN).

Причини, за якими організація встановлює мережу невеликих комп'ютерів, а не мейнфрейми – зниження вартості і розширюваність. По-перше, менші комп'ютери мають краще співвідношення ціна/продуктивність, ніж більші комп'ютери. По-друге, якщо потужність системи більше не задовільняє потреби, то мережа може бути розширена додаванням інших машин (файлових серверів, принтерів і робочих станцій). Якщо ж потужність монолітної системи стає незадовільною, залишається тільки повна її заміна.

Велика надійність завдяки реплікації. Розподілені системи мають потенціал надійності більший, ніж монолітні системи завдяки властивості їх часткового виходу з ладу. Це означає, що коли деякі вузли системи видуть з ладу, інші, як і раніше, функціонуватимуть і можуть взяти на себе завдання зіпсованих компонентів. Вихід з ладу монолітного комп'ютера діє на всю систему цілком і приводить до можливості продовжувати обчислення. З цієї причини розподілена архітектура при розробці високонадійних комп'ютерних систем є більш доцільною.

Високонадійна система як правило складається з декількох уніпроцесорів, які виконують застосовну програму. Правильне функціонування розподіленої системи за наявності пошкоджених компонент вимагає досить складної алгоритмічної підтримки.

Велика продуктивність завдяки розпаралелюванню. Наявність багатьох процесорів в розподіленій системі відкриває можливість зниження часу обробки завдяки розділенню роботи серед декількох процесорів.

Розподілення для забезпечення паралельного виконання часто застосовується при побудові обчислювальних систем, призначених для вирішення складних обчислювальних завдань. Мета такої розподіленої системи – зменшення часу виконання завдання.

Велика продуктивність завдяки балансуванню навантаження. Часто мотивом для створення розподіленої системи виступає збільшення загальної пропускної спроможності системи шляхом балансування навантаження складових її вузлів. При цьому підході завдання повністю потрапляє на найменш завантажену частину (вузол) системи. Як приклад можна привести будь-яку з систем пошуку в Internet, в якій запит користувача перенаправляється на найменш завантажений на даний момент веб-сервер.

Спрощення розробки завдяки спеціалізації. Розробка комп'ютерної системи може бути складною, особливо якщо потрібна значна функціональність. Розробка може бути часто спрощена розбиттям системи на модулі, кожен з яких відповідає за частину функціональності і взаємодіє з іншими модулями.

На рівні однієї програми модульність досягається визначенням абстрактних типів даних і процедур для різних завдань. Велика система представляється як набір взаємопов’язаних процесів. Модулі можуть бути виконані в рамках одного комп'ютера. Одночасно доцільно мати локальну мережу з різними типами комп'ютерів: один забезпечений спеціальним устаткуванням для обчислень, інший – графічним устаткуванням, третій – дисками і так далі.

Як приклад розподілених систем можна привести систему Інтернет. Із самого початку ця мережа розроблялася як розподілена система, здатна продовжувати функціонування при знищенні частини (можливо навіть великої) її складових (вузлів). В результаті з’явилася технологія об'єднання незалежних мереж за допомогою єдиного комунікаційного протоколу, що дозволяє динамічно перенастроювати маршрути передачі пакетів даних. З одного боку, Інтернет є яскравим прикладом системи, побудованої згідно архітектури P2P, – всі вузли в ньому незалежні. З іншого боку, якщо піднятися на рівень сервісів, то простежуються приклади практично всіх відомих архітектур.

Іншим прикладом розподіленої системи є Інтранет. Під інтранетом зазвичай розуміють мережі, об'єднані за якоюсь ознакою (мережі великого підприємства, наприклад). В інтранеті задіяні вузли, доступ до яких необхідний для його користувачів для здійснення певних потреб. Це вузли, що є серверами друку, баз даних, файл-сервера, пошти, сервера додатків та інше. Створення подібних мереж виникає внаслідок потреби в обміні інформацією та забезпечення сумісного доступу до розподілених ресурсів (принтерів та інших пристроїв, доступ до Інтернет, корпоративних даних та інше).

Інтранет часто називають «малим Інтернетом», так як він заснований переважно на тих же технологіях, що і «великий» Інтернет. У той же час йому притамана більша захищеність, яка забезпечується централізованим управлінням.

1.3. Кластерна технологія

Також прикладом розподілених систем можуть служити кластери. Під кластером як правило розуміють декілька обчислювальних вузлів, об'єднаних за допомогою деякої швидкісної технології передачі даних, що мають спеціальне програмне забезпечення. Комп'ютерні кластери в даний час набули великого поширення. Це пов'язано насамперед з тим, що через постійне зниження вартості на устаткування і появу останнім часом відповідного системного програмного забезпечення технологія створення кластерів стала загальнодоступною.

Завдяки застосуванню технології кластеризації можна вирішувати ряд задач. Ось деякі з них.

Перша технологія пов'язана з резервуванням деяких критичних сервісів. Для цього застосовуються кластери, налаштовані таким чином, що при виході з ладу одного з вузлів, що входять в кластер, сервіси, які обслуговуються цим вузлом, автоматично завантажуються на вузлі іншого кластера. Такий підхід дозволяє істотно мінімізувати час простою системи. Кластери, побудовані за таким принципом, називаються відмовостійкими.

Друга технологія вирішує шляхом кластеризації підвищення продуктивності системи. При такому підході один сервіс запускається на декількох вузлах кластера одночасно (при цьому реалізація може бути різною: або на кожному з вузлів запускається копія сервісу, або сервіс запускається на одному вузлі, а частина його процедур розміщується на інших вузлах, або інше). При цьому кількість одночасно оброблюваних завдань збільшується. Але слід

зазначити, що для забезпечення такої можливості сервіс має бути спеціальним чином спроектований. Кластери, завдяки яким вирішуються подібні завдання, називаються високопродуктивними.

Як правило, вимоги, що пред'являються кінцевою метою застосування кластерів, настільки різні, що кожен конкретний кластер здатний вирішувати тільки одну з них: або забезпечувати відмовостійкість, або підвищувати продуктивність системи.

1.4. Вимоги до сучасних програмних систем

Наведемо деякі вимоги, яким повинні задовільняти сучасні програмні системи, що розробляються. Більша частина з них справедлива не тільки до розподілених систем, а взагалі до всіх систем, що розробляються, у тому числі і монолітних. Проте, якщо «розподіленість» для монолітних застосувань може розглядатися як бажана характеристика, то для розподілених систем вона є обов'язковою.

Відкритість. Використання відкритих стандартів поряд з детальною документацією специфікацій та протоколів є важливим моментом при створенні будь-якого програмного продукту. Критичної важливості ці чинники набувають при створенні розподілених систем. На сьогодні поширеною є ситуація, коли додаток повинен функціонувати в гетерогенному середовищі. В цьому випадку особливо важливим є збереження незалежності програмного продукту від рішень, пов'язаних з конкретною платформою (з конкретним постачальником). Використання відкритих стандартів підвищує можливості інтеграції і розвитку програмних систем, а також підвищує вірогідність успішного повторного використання окремих програмних компонент і рішень.

Безпека. Це важлива властивість будь-якої програмної системи і розподіленої зокрема. Під безпекою зазвичай розуміють сукупність таких властивостей:

конфіденційність (забезпечення захисту від несанкціонованого доступу в систему);

цілісність (забезпечення цілісності і збереження даних);

доступність (забезпечення захисту при паралельному доступі до ресурсів).

Для розподілених систем, в яких часто дані пересилаються по мережі, важливим фактором для забезпечення конфіденційності є здібність шифрування повідомлень, що передаються по мережі.

Важливою проблемою при проектуванні розподілених систем є забезпечення якості обслуговування. Під якістю обслуговування розуміють якість угоди по виконанню запитів клієнта. Наприклад, запит повинен оброблятися не пізніше, ніж через 2 секунди після отримання або щось таке інше. Жорстким порушенням якості обслуговування є ситуація відмови в обслуговуванні, яка може виникнути у разі, якщо зловмисник атакує сервер запитами, які той не встигає обробляти. Вирішення цієї проблеми в загальному вигляді може бути дуже складним, проте існують рекомендації, що дозволяють обходити її в більшості випадків. Іншою важливою проблемою безпеки є

використання мобільного коду, що прийшов по мережі і виконується локально. Тут рішення полягає у використанні спеціального коду та застосуванні віртуальних машин.

Масштабованість. Це – одна з можливих переваг, що отримується при реалізації розподіленої системи. Саме можливих, а не дійсних, так як на практиці цілком вірогідна ситуація, коли архітектура розподіленої системи або не допускає зміни числа її вузлів через територіальні чинники, або через зниження продуктивності системи. Незважаючи на деякі обмеження, створення масштабованих систем є бажаним і в одночас складним. При створенні систем, що масштабуються, необхідно досліджувати їхню продуктивність на всіх етапах проектування та реалізації системи, починаючи від самих перших. Вимога масштабованості має завжди включатися в список формальних вимог до характеристик системи. Необхідно ретельно контролювати витоки ресурсів, досліджувати роботу системи, знаходячи і усуваючи вузькі місця. Існують емпіричні оцінки, того, як добитися масштабованості системи. Проте, є обмеження, які негативно впливають на продуктивність. Це можуть бути жорсткі обмеження середовища (наприклад, пропускна спроможність мережевого інтерфейсу), а деякі ресурси взагалі виключають масштабування.

Обробка помилок. Розподілені системи використовують певне середовище для комунікацій. Дуже часто в ролі такого середовища виступає мережа. На жаль, в розподілених системах часто виникають помилки, які пов’язані з тим, що, по-перше, сама по собі мережа є достатньо ненадійною, по-друге, тому, що архітектура розподіленої системи набагато складніша, ніж монолітної. У зв'язку з цим програмні засоби повинні виявляти ці помилки. Після виявлення помилки, вона має бути маскована, а система повинна спробувати продовжувати виконання процеса, для чого нерідко потрібно повернутися до стану, що мав місце до виникнення помилки.

Паралельність. Написання програм, що працюють в паралельному середовищі, складна задача. Проблеми паралелізму виникають всякий раз, коли декілька процесів або потоків обчислень роблять спроби маніпуляції одними і тими ж елементами даних. Сприйняття безлічі паралельних операцій ускладнене врахуванням всіх можливих сценаріїв розвитку подій, що здатні привести до тих або інших негативних наслідків, вкрай складно. Більше того, паралельні операції важко тестувати. Модель транзакцій дозволяє уникнути деяких труднощів: якщо ви маніпулюєте даними всередині транзакції, можна бути впевненим, що нічого поганого з ними не трапиться. Проте це не означає, що проблемами управління паралельними завданнями можна повністю нехтувати, тому що в багатьох випадках доводиться мати справу з даними, що модифікуються декількома транзакціями. Вищезгадана ситуація отримала назву автономного паралелізму.

Втім, труднощі пов'язані не тільки з паралелізмом. Іноді їх вносять керуючі системи. Наприклад, втрачені зміни або ефект неузгодженого читання, який виникає в ситуаціях, коли прочитуються дві начебто коректні порції даних, які, проте, не можуть існувати в один і той же момент часу.

Обидві проблеми виражаються у втраті достовірності інформації та некоректній поведінці системи, яких можна було б уникнути, якби два процеси не зверталися до одних і тих же даних одночасно. Основною проблемою будь-якої моделі програмування паралельних операцій є не тільки збереження коректності даних, але і забезпечення максимального ступеня паралелізму.

Більшість з цих проблем має відомі розв'язки, так як вони виникли ще на початку розвитку галузі. При написанні розподілених програм необхідно використовувати весь досвід, накопичений у даній області – використання критичних секцій та блокуючих змінних, правильна послідовність накладення блокувань та інше.

Прозорість. Під прозорістю розуміють приховування від користувача гетерогенної природи системи та представлення її на верхньому рівні як єдиної системи. Виділяють вісім видів прозорості:

прозорість доступу: доступ до локальних та віддалених ресурсів за допомогою однакових викликів;

прозорість розташування: доступ до ресурсів незалежно від їх фізичного розташування;

прозорість паралелізму: можливість декільком процесам паралельно працювати з ресурсами, не впливаючи один на одного;

прозорість реплікації: можливість декільком екземплярам одного ресурсу використовуватися без знання фізичних особливостей реплікації;

прозорість обробки помилок: захист програмних компонентів від відмов, що мали місце в інших програмних компонентах, відновлення після збоїв;

прозорість мобільності: можливість перенесення програмного додатку між платформами без його переробки;

прозорість продуктивності: можливість конфігурації системи з метою збільшення продуктивності при зміні складу платформи виконання;

прозорість масштабованості: можливість збільшення продуктивності без зміни структури програмної системи та алгоритмів, що використовуються.

Для розподілених систем найбільш важливими є прозорість доступу та фізичного розташування, оскільки вони мають критичне значення для належного використання розподілених ресурсів.

Керованість. Система використовується тільки в тому випадку, якщо вона керована. Для розподіленої системи задача управління може бути досить складною, бо для розподіленого ресурсу не може бути призначено єдиного центру керування. Це зумовлене тим, що відмова керуючого вузла веде за собою зупинку всієї системи. Інша проблема полягає в тому, що існують системи, які нікому не належать. За своєю сутністю вони є сукупністю більш дрібних або приватних систем, наприклад, Internet. А тому задача управління повинна вирішуватися у кожному конкретному випадку відповідним розв'язком.

Складність реалізації. Не дивлячись на те, що розподілені системи можуть володіти рядом переваг в порівнянні з монолітними, при їх створенні доводиться мати справу з рядом істотних труднощів. Наведемо деякі з них.

Залежність від вибраної архітектури. Як і у випадку проектування традиційного програмного додатку, питання вибору структурних рішень (шаблонів), що лежать в його основі є базовим, навіть можна вважати вирішальним. Так компоненти розподіленої системи часто вимушені взаємодіяти по мережі. Це значно уповільнює виконання додатку (майже на порядок) порівняно з монолітними системами, для яких взаємодія полягає у локальних викликах процедур. Внаслідок цього невдало вибрана структура інтерфейсів або компоновка модулів системи може привести до катастрофічних втрат продуктивності.

Крім того, через те, що різні модулі системи виконуються на незалежних вузлах і в рамках незалежних процесів, гостріше виявляються проблеми синхронізації доступу до ресурсів, а також всі пов'язані з цим питання, а саме: забезпечення транзакційної обробки, рівні ізоляції транзакцій та інше.

Гетерогенне середовище. Як правило різні частини розподіленого програмного комплексу можуть виконуватися на різних вузлах системи. Вони в свою чергу можуть відрізнятися апаратною, мережевою та програмною архітектурою. Це спричиняє серйозні проблеми в програмуванні таких систем. Навіть представлення такого фундаментального поняття як тип даних залежить від мови програмування, від апаратної архітектури і т.п. Якнайкращим виходом з цієї ситуації може вважатися або використання прийнятих і підтримуваних стандартів (протоколу TCP/IP для мережевої взаємодії або POSIX для системних викликів). Або використання спеціальних проміжних засобів (middleware), що маскують гетерогенність (Java RMI, CORBA.NET та інше). Цікавим вирішенням проблеми гетерогенності є використання віртуальних машин (наприклад, jvm), що здатні виконувати деякий незалежний від застосованої цільової системи код. При цьому можуть бути створені системи, які працюють і взаємодіють на всяких платформах. Поруч з цим вирішується ще ряд серйозних проблем, пов’язаних, наприклад, з безпекою. Але серйозним недоліком такого підходу є те, що при цьому знижується продуктивність за рахунок введення додаткового посередника – віртуальної машини. Проте, провідні фахівці на ринках Sun, IBM і Microsoft в даний час активно працюють в цьому напрямі, а це свідчить про перспективність даного підходу. Альтернативним методом є підхід, при якому всі модулі системи компілюються під платформи, на яких система використовуватиметься. Цей метод на сьогодні є дуже розповсюдженим.

Складність розгортання. На відміну від монолітного ресурсу, розподілений додаток має бути розділений на модулі розгортання. Це пояснюється тим, що різні частини розподіленого програмного ресурсу виконуватимуться на різних вузлах, які досить часто мають різні характеристики, а саме апаратну, програмну платформу та інше. На кожен вузол має бути коректно інстальований свій модуль. У випадку, коли в процесі роботи системи відбувається міграція програмних компонент між вузлами, необхідно враховувати особливості гетерогенного середовища.

Складність налагодження. Характерною особливістю розподілених систем є те, що стан, в якому перебуває система, є також «розподіленим». Це

обумовлене тим, що для його фіксації необхідно зібрати інформацію з декількох обчислювальних вузлів. Іншою проблемою є відтворюваність результатів виконання, оскільки процеси, які виконуються на різних обчислювальних вузлах, можуть виконуватися з різною швидкістю, а їх треба гармонізувати між собою.

Шаблони рішень. Все вищезазначене демонструє видиму складність створення розподіленої програмної системи в порівнянні з монолітною. Незважаючи на значні труднощі, що пов’язані з розробкою розподілених систем, такі системи не тільки розробляються, але і досить широко експлуатуються вже досить довго. Це означає, що знайдено підходи щодо вирішення перерахованих вище проблем. Відомо, що при вирішенні всякої складної задачі доцільно ознайомитися з тим, як подібні завдання розв’язувалися раніше.

При уважному дослідженні великих розподілених систем виявляється досить багато загального в їх будові. Це дозволяє говорити про наявність різної архітектури побудови розподілених систем, яка по суті і є таким загальним розв’язком (шаблоном). Крім того, розгляд архітектури, в якій виконуються програмні засоби, дозволяє їх класифікувати за цією ознакою. Оскільки з кожною архітектурою пов’язані характерні її особливості, цих рис набувають і програмні комплекси, реалізовані за відповідною архітектурою. Подібна класифікація на практиці виявляється досить корисною при виборі архітектури проектованої системи.

Насправді, знаючи вимоги до системи та відмінні риси різної архітектури побудови, можна вибрати серед них найбільш відповідну для кожного конкретного випадку.

При початковому моделюванні системи звертають увагу не тільки на розбиття системи на частини (декомпозицію), але і на взаємодію цих частин (модулів), які можливо виконуватимуться на різних вузлах. Вибір архітектури в більшості випадків породжує той або інший спосіб декомпозиції системи, а також в більшості випадків і способи взаємодії її частин. Добре продумана структура системи повинна забезпечувати надійність, керованість, гнучкість і при цьому дозволяти побудувати економічно ефективне рішення.

При описі архітектури функції окремих компонент, як правило, не вказують. Важливішими є розміщення компонент (з урахуванням мережевої топології, яка може мати вирішальне значення при виборі тієї або іншої архітектури), способи розподілу та управління даними, розподіл навантаження між компонентами, а також способи взаємодії. Взаємодія компонентів для розподілених систем є ключовим аспектом. Дуже часто саме особливостями взаємодії диктується зрештою вибір архітектури системи. Інколи, якщо мова йде про реальну ситуацію, модулі розподіленої системи зазвичай вимушені взаємодіяти через мережу того або іншого типу. Відомо, що час передачі даних в мережі на декілька порядків більший, ніж час звернення до даних в локальній пам'яті. Тому прагнуть побудувати систему так, щоб мінімізувати кількість та обсяг такої взаємодії, переносять найбільш активно взаємодіючі модулі на вузли, що з'єднані більш швидкісними каналами.

Взаємодія передбачає участь двох сторін – "клієнта" та "сервера". Відповідно, перший модуль буде клієнтом по відношенню до другого. На практиці ролі визначаються тільки на момент конкретної взаємодії. Так до модуля, що є ініціатором запиту ("клієнтом"), водночас може звернутися інший модуль, що змінить його роль з "клієнта" на "сервер") .

Як зазначалося вище, архітектуру можна розуміти як якийсь шаблон, деяку загальну ідею декомпозиції та взаємодії компонентів, яка до цього вже багато раз застосовувалася. Це дозволяє компенсувати зростаючу складність системи.

Інший спосіб подолання складності полягає в розбитті системи, що розробляється, на рівні. Ідея полягає в тому, що існує певне коло задач, які виникають перед розробниками постійно. Отже, потрібно створити деякий набір готових програмних рішень, достатньо загальних за своєю сутністю, щоб ними можна було б скористатися при нагоді. При використанні такої методики додаток розглядається як верхній рівень, що використовує функціональність більш нижчого рівня, який часто називають проміжним програмним забезпеченням (middleware).

Проміжне програмне забезпечення (Middleware) ОС та апаратне забезпечення – два нижніх рівні часто об'єднуються під загальним терміном платформа.

Middleware – програмне забезпечення між платформою та компонентами розподіленого додатка. Цей рівень не є обов'язковим, але його наявність досить часто бажана. Він реалізує задачі приховування (маскування) гетерогенності платформи та забезпечує зручну модель програмування.

Як приклад такого роду програмного забезпечення можна привести такі продукти:

Java (Sun) – технологія виконання проміжного байт-кода на віртуальній машині. Байт-код, що одного разу відкомпілювався, може виконуватися на будь-якій платформі, для якої реалізована JVM (Java Virtual Machine), без перекомпіляції;

.NET – технологія Microsoft, заснована на виконанні заздалегідь відкомпільованих в проміжний код компонент. Такі компоненти можуть виконуватися на будь-якій платформі, для якої реалізована підтримка бібліотеки виконання .NET;

CORBA – технологія, що розробляється консорціумом OMG, яка дозволяє викликати методи віддалених об'єктів. Об'єкти можуть бути створені з використанням різних мов програмування;

Remote Procedure Call (Sun),

Java Remote Method Invocation (Sun);

MPI, PVM та багато інших.

Проте, слід зазначити, що більшість з перелічених засобів, є не просто бібліотекою процедур. Досить часто для того, щоб використовувати один з цих засобів, програма має бути не тільки написана, але і спроектована спеціальним чином. По суті, виробники часто нав'язують ту або іншу модель архітектури, тому вибір проміжного програмного забезпечення перетворюється на дуже

серйозне рішення, поміняти яке в процесі реалізації проекту буде дуже складно, а іноді практично неможливо.

Багато сучасних засобів проміжного програмного забезпечення підтримують безліч корисних сервісів, таких як сервіс іменування, сервіс безпеки, підтримка транзакцій, підтримка довготривалого зберігання об'єктів, сервіс повідомлення про події та багато інших. Їх використання може значно прискорити розробку додатків, спростити їх налагодження та супровід. Крім того, їх використання може спростити подальшу інтеграцію з компонентами або навіть цілими системами інших розробників, так як ці сервіси часто використовують для взаємодії відомі стандарти.

1.5. Контрольні питання

1. Що таке інформаційні технології.
2. Що таке розподілені інформаційні системи.
3. Що таке монолітні інформаційні системи.
4. Порівняти між собою розподілені та монолітні системи.
5. Навести переваги та недоліки підходу “хост-комп'ютер + термінали”.
6. Сутність клієнт-серверної обробки.
7. Які переваги має концепція “програмних рівнів”.
8. Що можна віднести до верхнього рівня моделі рівнів.
9. Основні недоліки концепції рівнів.
10. В чому полягає сутність технології об’єктно-орієнтованого програмування.
11. В чому заключається позитивний вплив розподілених систем на обмін інформацією між різними вузлами.
12. Розкрийте основні переваги від розподілення ресурсів.
13. Чому надійність розподілених систем вища ніж монолітних.
14. Наслідки розпаралелювання для розподілених систем.
15. Як балансування навантаження здатне підвищувати продуктивність розподілених систем.
16. Навести приклади розподілених систем.
17. Що таке кластер в контексті розподілених систем.
18. Сутність технології кластеризації.
19. Які задачі можна реалізовувати завдяки технології кластеризації.
20. В чому полягає принцип відкритості розподілених систем.
21. Основні властивості безпеки системи.
22. В чому полягає сутність фактору конфіденційності при пересиланні інформації по мережі.
23. Розкрийте сутність принципа масштабованості розподілених систем.
24. Який механізм обробки помилок в розподілених системах.
25. Основні труднощі, пов’язані з паралельністю в розподілених системах.
26. Що таке прозорість розподіленої системи.
27. Які існують види прозорості розподілених систем.
28. Призначення проміжного програмного забезпечення.

Розділ 2. Основи мови PHP

2.1. Що таке PHP

PHP- це широко розповсюджена мова сценаріїв загального призначення з відкритим вихідним кодом.

Говорячи простіше, PHP це мова програмування, спеціально розроблена для написання web-додатків, що виконуються на Web-сервері.

Абревіатура PHP означає **"Hypertext Preprocessor (Препроцесор Гіпертексту)"**. Синтаксис мови бере початок з C, Java і Perl. PHP досить простий для вивчення. Перевагою PHP є надання web-розробникам можливості швидкого створення динамічних web-сторінок.

Важливою **перевагою** мови PHP перед такими мовами, як мов Perl і C полягає в можливості створення HTML документів із вбудованими командами PHP.

Значною відмінністю PHP від якого-небудь коду, що виконується на стороні клієнта, наприклад, JavaScript, є те, що PHP-скрипти виконуються на стороні сервера. Ви навіть можете конфігурувати свій сервер таким чином, щоб HTML-файли оброблялися процесором PHP, так що клієнти навіть не зможуть дізнатися, чи отримують вони звичайний HTML-файл або результат виконання скрипта.

PHP дозволяє створювати якісні Web-додатки за дуже короткі терміни, отримані продукти легко модифікуються і підтримуються в майбутньому.

PHP простий для освоєння, і разом з тим здатний задовольнити інтереси професійних програмістів.

Мова PHP постійно удосконалюється, і їй, напевно забезпечене довге домінування в області мов web-програмування, принаймні, найближчим часом.

Можливості PHP

PHP може все. **Головна область застосування** PHP - це написання скриптів, що працюють на стороні сервера; таким чином, PHP здатний виконувати все те, що виконує будь-яка інша програма CGI, наприклад, обробляти дані форм, генерувати динамічні сторінки або відсилати й приймати cookies. Але PHP здатний виконувати ще і багато інших завдань.

Існують три основні області застосування PHP.

- *Створення скриптів для виконання на стороні сервера.* PHP найбільш широко використовується саме таким чином. Все, що вам знадобиться, це інтерпретатор PHP (у вигляді програми CGI або серверного модуля), вебсервер і браузер. Для того щоб можна було переглядати результати виконання PHP-скриптів в браузері, потрібен працюючий веб-сервер і встановлений PHP. У випадку, якщо ви просто експериментуєте, ви цілком можете використовувати свій домашній комп'ютер замість сервера.
- *Створення скриптів для виконання в командному рядку.* Ви можете створити PHP-скрипт, здатний запускатися незалежно від веб-сервера та броузера. Все, що вам буде потрібно – **парсер** PHP. Такий спосіб використання PHP ідеально підходить для скриптів, які повинні виконуватися регулярно, наприклад, за допомогою cron (на платформах * nix або Linux) або за допомогою

планувальника завдань (Task Scheduler) на платформах Windows. Ці скрипти також можуть бути використані в задачах простої обробки текстів.

- *Створення звичайних програм, що виконуються на стороні клієнта.* Можливо, PHP є не найкращою мовою для створення подібних додатків, але, якщо ви дуже добре знаєте PHP і хотіли б використати деякі його можливості у своїх клієнт-додатках, ви можете використовувати PHP-GTK для створення таких додатків. Подібним чином ви можете створювати і крос-платформні додатки. PHP-GTK є розширенням PHP і не поставляється разом з дистрибутивом PHP.

PHP доступний для більшості операційних систем, включаючи **Linux**, багато модифікації **Unix**(такі, як HP-UX, Solaris і OpenBSD), Microsoft Windows, Mac OS X, RISC OS, та багатьох інших. Також в PHP включена підтримка більшості сучасних веб-серверів, таких, як Apache, Microsoft Internet Information Server, Personal Web Server, серверів Netscape і iPlanet, сервера Oreilly Website Pro, Caudium, Xitami, OmniHTTPd та багатьох інших. Для більшості серверів PHP поставляється в якості модуля, для інших, що підтримують стандарт CGI, PHP може функціонувати як процесор CGI.

Таким чином, вибираючи PHP, ви отримуєте свободу вибору операційної системи і веб-сервера. Крім того, у вас з'являється вибір між використанням процедурного або об'єктно-орієнтованого програмування або ж їх поєднання. Багато бібліотек коду і великі програми (включаючи бібліотеку PEAR) написані тільки з використанням ООП.

PHP здатний генерувати не тільки HTML. Доступно формування зображень, файлів PDF і навіть роликів Flash (з використанням libswf і Ming), що створюються «на льоту». PHP також здатний генерувати будь-які текстові дані, такі, як XHTML та інші XML-файли. PHP може здійснювати автоматичну генерацію таких файлів і зберігати їх у файловій системі вашого сервера замість того, щоб віддавати клієнту, організовуючи, таким чином, кеш динамічного наповнення, розташований на стороні сервера.

Інструментарій

Мінімальна програма

Традиційно, знайомство з мовою програмування починають з горезвісної програми "Hello, World!". Що ж, ми не будемо відступати від цієї традиції, і напишемо нашу першу програму на PHP!

Отже, беремо редактор PHP-коду, і напишемо наступний PHP код:

```
<? Php
echo "Hello, World!";
?>
```

Перш, ніж запустити програму, її потрібно встановити на сервері. Для цього збережіть написаний PHP-скрипт під назвою **start.php**. Потім скопіюйте його в каталог **DocumentRoot**вашого сервера. Тепер наберіть в адресному рядку вашого браузера **http://localhost/start.php** і, якщо все встановлено і налаштовано правильно, ви побачите текст Hello, World!

Синтаксис

Ми приступаємо до вивчення основних елементів синтаксису мови PHP. Розглянемо способи поділу інструкцій і створення коментарів, змінні, константи, типи даних і оператори.

Основний синтаксис

Перше, що потрібно знати щодо синтаксису PHP, - це те, як він вбудовується в HTML-код, як інтерпретатор дізнається, що це код на мові PHP. В прикладах ми найчастіше будемо використовувати варіант <? Php?>, І іноді скорочений варіант <? ?>.

Поділ інструкцій

Програма на **PHP** (та й на будь-якій іншій мові програмування) - це набір команд (інструкцій). Оброблювачу програми (парсеру) необхідно якось відрізняти одну команду від іншої. Для цього використовуються спеціальні символи - роздільники. У PHP інструкції поділяються так само, як і у Сі або Perl, - кожен вираз закінчується крапкою з комою.

Закриваючий тег «?>» Має на увазі кінець інструкції, тому перед ним крапку з комою не ставлять. Наприклад, два наступних фрагмента коду еквівалентні:

```
<? Php
echo "Hello, world!"; / / крапка з комою
// В кінці команди
// Обов'язкова
?>
<? Php
echo "Hello, world!" ?>
<! - Крапка з комою
опускається з-за "?>" ->
```

2.2. Коментарі, змінні, константи й оператори.

Часто при написанні програм виникає необхідність робити будь-які коментарі до коду, які ніяк не впливають на сам код, а тільки пояснюють його. Це важливо при створенні великих програм і у випадку, якщо кілька людей працюють над однією програмою. При наявності коментарів у програмі в її коді розібратися набагато простіше. Крім того, якщо вирішувати задачу по частинах, недороблені частини рішення також зручно коментувати, щоб не забути про них надалі. В усіх мовах програмування передбачена можливість включати коментарі в код програми. PHP підтримує кілька видів коментарів: у стилі Сі, C ++ і оболонки Unix. Символи // і # позначають початок однорядкових коментарів, /* і */ - відповідно початок і кінець багаторядкових коментарів.

Приклад 1. Використання коментарів в PHP

```
<? Php
echo "Мене звуть Вася";
// Це однорядковий коментар
// У стилі C ++
echo "Прізвище моє Петров";
```

```
/* Це багаторядковий коментар.
Тут можна написати кілька рядків.
При виконанні програми все, що
знаходиться тут, буде проігнороване. */
echo "Я вивчаю PHP";
# Це коментар в стилі
# Оболонки Unix
?>
```

Змінні, константи й оператори

Важливим елементом кожної мови є змінні, константи й оператори. Розглянемо, як виділяються і обробляються ці елементи в PHP.

Змінні

Змінна у PHP позначається знаком долара, за яким слідує її ім'я. Наприклад: $My_var

Ім'я змінної чутливо до регістру, тобто змінні **$my_var** і **$My_var різні**.

Імена змінних відповідають тим же правилам, що й інші найменування в PHP: правильне ім'я змінної має починатися з букви або символу підкреслення з подальшим в будь-якій кількості літерами, цифрами або символами підкреслення.

У PHP змінні завжди присвоювалися за значенням. Тобто коли ви присвоюєте вираз змінної, всі значення оригінального виразу копіюються в цю змінну. Це означає, наприклад, що після присвоєння однієї змінної значення іншої, зміна однієї з них не впливає на значення іншої.

Константи

Для зберігання постійних величин, тобто таких величин, значення яких не змінюється в ході виконання скрипта, використовуються константи. Такими величинами можуть бути математичні константи, паролі, шляхи до файлів і т.п. Основна відмінність константи від змінної полягає в тому, що їй не можна присвоїти значення більше одного разу і її значення не можна анулювати після її оголошення. Крім того, у константи немає приставки у вигляді знаку долара і її не можна визначити простим присвоєнням значення. Як же тоді можна визначити константу? Для цього існує спеціальна функція define (). Її синтаксис такий:

```
define ("Імя_константи",
"Значення_константи",
[Нечутливість _до_регістру])
```

За замовчуванням імена констант чутливі до регістру. Для кожної константи це можна змінити, вказавши в якості значення аргументу Нечутливість _до_регістру значення True. Існує правило, за яким імена констант завжди пишуться у верхньому регістрі.

Отримати значення константи можна, вказавши її ім'я. На відміну від змінних, не потрібно випереджати ім'я константи символом $. Крім того, для отримання значення константи можна використовувати функцію constant () з ім'ям константи в якості параметра.

Приклад 3. Константи в PHP

```
<? Php
// Визначаємо константу
// PASSWORD
define ("PASSWORD", "qwerty");
// Визначаємо регістро незалежну
// Константу PI зі значенням 3.14
define ("PI", "3.14", True);
// Виведемо значення константи PASSWORD,
// Тобто qwerty
echo (PASSWORD);
// Теж виведе qwerty
echo constant ("PASSWORD");
echo (password);
/ * Виведе password і попередження,
оскільки ми ввели регістрочутливу
константу PASSWORD * /
echo pi;
// Виведе 3.14, оскільки константа PI
// Регістронезалежна за визначенням
?>
```

Крім змінних, які декларуються користувачем, про які ми тільки що розповіли, в PHP існує ряд констант, що визначаються самим інтерпретатором. Наприклад, константа **__FILE__** зберігає ім'я файлу програми (і шлях до нього), яка виконується в даний момент, **__FUNCTION__**містить ім'я функції, **__CLASS__** - ім'я класу, **PHP_VERSION** - версія інтерпретатора PHP. Повний список зумовлених констант можна отримати, прочитавши посібник з PHP.

Оператори

Оператори дозволяють виконувати різні дії зі змінними, константами і виразами. Вираз можна визначити як все, що завгодно, що має значення. Змінні і константи - це основні і найбільш прості форми виразів. Існує безліч операцій (і відповідних їм операторів), які можна робити з виразами. Розглянемо деякі з них докладніше.

Таблиця 1. Арифметичні оператори

Позначення	Назва	Приклад
+	Додавання	$a + $b
-	Віднімання	$a - $b
*	Множення	$a * $b
/	Ділення	$a / $b
%	Залишок від ділення	$a % $b

Таблиця 2. Строкові оператори

Позначення	Назва	Приклад
.	Конкатенація (додавання рядків)	$c = $a . $b (це рядок, що складається з $a і $b)

Таблиця 3. Оператори присвоювання

Позначення	Назва	Опис	Приклад
=	Присвоєння	Змінній ліворуч від оператора буде присвоєне значення, отримане в результаті виконання яких-небудь операцій або змінній/константи із правої сторони	$a = ($b = 4) +5; ($a буде дорівнювати 9, $b буде дорівнювати 4)
+=	Скорочення	Додає до змінної число й потім присвоює їй отримане значення	$a += 5; (еквівалентно $a = $a + 5;)
.=	Назва	Скорочено позначає комбінацію операцій конкатенації й присвоювання (спочатку додається рядок, потім отримана рядок записується в змінну)	$b = "Привіт "; $b .= "всім"; (еквівалентно $b = $b . "всім";) У результаті: $b="Привіт всім"

Таблиця 4. Логічні оператори

Позначення	Назва	Опис	Приклад
And	і	$a і $b істинні (True)	$a and $b
&&	і		$a && $b

Or	Або	Хоча б одна зі змінних $a або $b істина (можливо, що й обидві)	$a or $b
||	Або		$a || $b
Xor	Що виключає або	Одна зі змінних істина. Випадок, коли вони обидві істині, виключається	$a xor $b
!	Інверсія (NOT)	Якщо $a=True, то !$a=False і навпаки	! $a

Таблиця 5. Оператори порівняння

Позначення	Назва	Опис	Приклад
==	Рівність	Значення змінних рівні	$a == $b
===	Еквівалентність	Рівні значення і типи змінних	$a === $b
!=	Нерівність	Значення змінних не рівні	$a != $b
<>	Нерівність		$a <> $b
!==	Нееквівалентність	Змінні не еквівалентні	$a !== $b
<	Менше		$a < $b
>	Більше		$a > $b
<=	Менше або дорівнює		$a <= $b
>=	Більше або дорівнює		$a >= $b

Таблиця 6. Оператори інкремента й декремента

Позначення	Назва	Опис	Приклад
++$a	Пре-Інкремент	Збільшує $a на одиницю й повертає $a	

$a++	Пост-інкремент	Повертає $a, потім збільшує $a на одиницю
--$a	Пре-декремент	Зменшує $a на одиницю й повертає $a
$a--	Пост-декремент	Повертає $a, потім зменшує $a на одиницю

Типи даних

PHP підтримує вісім простих типів даних.

Чотири скалярних типи:

- boolean (логічні дані)
- integer (цілі числа)
- float (число з плаваючою крапкою або 'double')
- string (рядки)

Два змішаних типу:

- array (масиви)
- object (об'єкти)

І два спеціальних типи:

- resource (ресурси)
- NULL (порожній тип)

Існують також кілька **псевдотипів:**

- mixed (змішаний тип)
- number (числа)
- callback (зворотного виклику)

У PHP не прийнято явне оголошення типів змінних. Переважно, це робить сам інтерпретатор під час виконання програми в залежності від контексту, в якому використовується змінна. Розглянемо по порядку всі перераховані типи даних.

Тип boolean (логічний тип)

Цей найпростіший тип, що висловлює істинність значення, тобто змінна цього типу може мати лише два значення - істина TRUE або брехня FALSE.

Щоб визначити логічний тип, використовують ключове слово TRUE або FALSE. Обидва регістронезалежні.

Приклад 4. Логічний тип

```
<? Php
$Test = True;
?>
```

Логічні змінні використовуються в різних управляючих конструкціях (циклах, умовах тощо). Мати логічний тип, тобто приймати тільки два значення, істину, чи брехню, можуть також і деякі оператори (наприклад, оператор рівності). Вони також використовуються в керуючих конструкціях для перевірки будь-яких умов. Наприклад, в умовній конструкції перевіряється істинність

значення оператора або змінної і залежно від результату перевірки виконуються ті чи інші дії. Тут умова може бути істинно або хибно, що якраз і відображає змінна і оператор логічного типу.

Приклад 5. Використання логічного типу

```
<? Php
// Оператор '==' перевіряє рівність
// І повертає
// Булеве значення
if ($ know == False) {// якщо $know
// Має значення
// False
echo "Вивчай PHP!";
}
if (!$know) {/ / те ж саме, що
// І вище, тобто перевірка
// Чи має $know значення
// False
echo "Вивчай PHP!";
}
/* Оператор == перевіряє, чи збігається
значення змінної $know з рядком
"Вивчити PHP". Якщо співпадає, то
повертає true, інакше - false.
Якщо повернуто true, то виконується
те, що всередині фігурних дужок * /
if ($know == "Вивчити PHP")
{Echo "Почав вивчати";}
?>
```

Тип integer (цілі)

Цей тип задає число з множини цілих чисел Z = {..., -2, -1, 0, 1, 2, ...}. Цілі можуть бути вказані у десятковій, шістнадцятковій або вісімковій системі числення, за бажанням з попереднім знаком «-» або «+».

Якщо ви використовуєте вісімкову систему числення, ви повинні перед числом ставити 0 (нуль), для використання шістнадцяткової системи потрібно поставити перед числом 0x.

```
<? Php
# Десяткове число
$A = 1234;
# Від'ємне число
$A = -123;
# Вісімкове число (еквівалентно
# 83 у десятковій системі)
$A = 0123;
# Шістнадцяткове число (еквівалентно
```

```
# 26 у десятковій системі)
$A = 0x1A;
?>
```

Розмір цілого залежить від платформи, хоча, як правило, максимальне значення близько двох мільярдів (це 32-бітове знакове). Беззнакові цілі PHP не підтримує.

Якщо ви визначите число, що перевищує межі цілого типу, воно буде інтерпретовано як число з плаваючою крапкою. Також якщо ви використовуєте оператор, результатом роботи якого буде число, що перевищує межі цілого, замість нього буде повернуто число з плаваючою крапкою.

У PHP не існує оператора ділення цілих. Результатом 1/2 буде число з плаваючою крапкою 0.5. Ви можете навести значення до цілого, що завжди округлює його в меншу сторону, або використовувати функцію round (), округлюються значення за стандартними правилами. Для перетворення змінної до конкретного типу потрібно перед змінною вказати в дужках потрібний тип. Наприклад, для перетворення змінної $a = 0.5 до цілого типу необхідно написати (integer) (0.5) або (integer) $a або використовувати скорочений запис (int) (0.5). Можливість явного приведення типів за таким принципом існує для всіх типів даних (звичайно, не завжди значення одного типу можна перевести в інший тип). Ми не будемо заглиблюватися у всі тонкощі приведення типів, оскільки PHP робить це автоматично залежно від контексту.

Тип float (числа з плаваючою крапкою)

Числа з плаваючою крапкою (вони ж числа подвійної точності або дійсні числа) можуть бути визначені за допомогою будь-якого з наступних синтаксисів:

```
<? Php
$ A = 1.234;
$ B = 1.2e3;
$ C = 7E-10;
?>
```

Розмір числа з плаваючою крапкою залежить від платформи, хоча максимум, як правило, ~ 1.8e308 з точністю близько 14 десяткових цифр.

Тип string (рядки)

Рядок - це набір символів. У PHP символ - це те ж саме, що байт, це означає, що існує рівно 256 різних символів. Це також означає, що PHP не має вбудованої підтримки Unicode. У PHP практично не існує обмежень на розмір рядків, тому немає абсолютно ніяких причин турбуватися про їх довжину.

Рядок у PHP може бути визначений **трьома різними** способами:

- за допомогою одинарних лапок;
- за допомогою подвійних лапок;
- heredoc-синтаксисом.

Одинарні лапки

Найпростіший спосіб **визначити рядок** - це розмістити його в одинарні лапки «'». Щоб використовувати одинарні лапки всередині рядка, як і в багатьох інших мовах, перед нею необхідно поставити символ зворотної косої межі «\»,

тобто екранувати її. Якщо зворотній слеш повинен йти перед одинарними лапками або бути в кінці рядка, необхідно продублювати його «\\ '».

Якщо всередині рядка, розміщеного в одинарні лапки, зворотній слеш «\» зустрічається перед будь-яким іншим символом (відмінним від «\» і «'»), то він розглядається як звичайний символ і виводиться, як і всі інші. Тому зворотну косу риску необхідно екранувати, тільки якщо вона знаходиться в кінці рядка, перед останньою лапкою.

У PHP існує ряд комбінацій символів, які починаються з символу зворотнього косого слеша. Їх називають управляючими послідовностями, і вони мають спеціальні значення, про які ми розповімо трохи пізніше. Так от, на відміну від двох інших синтаксисів, змінні і керуючі послідовності для спеціальних символів, що зустрічаються в рядках, взятих в одинарні лапки, не обробляються.

Приклад 6. Використання керуючих послідовностей

```
<? Php
echo 'Також ви можете додавати до рядка
символ нового рядка таким чином,
це нормально ';
// Виведе: Щоб вивести ' треба
// Перед нею поставити \
echo 'Щоб вивести \' треба перед '.
'Нею поставити \\';
// Виведе: Ви хочете видалити C: \ *.*?
echo 'Ви хочете видалити C: \ \ *.*?';
// Виведе: Це не вставить: \ n новий рядок
echo 'Це не вставить: \ n новий рядок';
// Виведе: Змінні $ expand також
// $Either не підставляються
echo 'Змінні $ expand також $ either'.
'Не підставляються';
?>
```

Подвійні лапки

Якщо рядок помістити у подвійні лапки «" », то PHP розпізнає більшу кількість керуючих послідовностей для спеціальних символів. Деякі з них наведені в таблиці 7.

Послідовність	Значення
\n	Новий рядок (LF або 0x0A (10) в ASCII)
\r	Повернення каретки (CR або 0x0D (13) в ASCII)
\t	Горизонтальна табуляція (HT або 0x09 (9) в ASCII)
\\	Зворотній слеш

\$	Знак долара
\"	Подвійна лапки

Найважливішою властивістю рядків у подвійних лапках є обробка змінних.

Heredoc

Інший спосіб визначення рядків - це використання **heredoc-синтаксису**. У цьому випадку рядок повинен починатися з символу <<, після якого йде ідентифікатор. Закінчується рядок цим самим ідентифікатором. Закриваючий ідентифікатор повинен починатися в першому стовпці рядка. Крім того, ідентифікатор повинен відповідати тим же правилам іменування, що і всі інші позначки в PHP: містити тільки буквено-цифрові символи і знак підкреслення та починатися з цифри або знака підкреслення.

Heredoc-текст веде себе так само, як і рядок в подвійних лапках, при цьому їх не маючи. Це означає, що вам немає необхідності екранувати лапки в heredoc, але ви як і раніше можете використовувати перераховані вище керуючі послідовності. Змінні всередині heredoc теж обробляються.

Приклад 7. Використання heredoc-синтаксису

```
<? Php
$Str = <<
Приклад рядка, що охоплює кілька
рядків, з використанням
heredoc-синтаксису
EOD;
// Тут ідентифікатор – EOD використовується як маркер завершення даних.
    $Name = 'Вася';
echo <<
Мене звуть "$name".
EOD;
// Це виведе "Мене звуть" Вася ".
?>
```

Зауваження: Підтримка heredoc була додана з PHP 4.

2.3. Масиви та списки в PHP

Масиви (arrays) - це впорядковані набори даних, що представляють собою список однотипних елементів.

Існує два типи масивів, що розрізняються за способом ідентифікації елементів.

1. У масивах першого типу елемент визначається індексом у послідовності. Такі масиви називаються простими масивами.
2. Масиви другого типу мають асоціативну природу, і для звернення до елементів використовуються ключі, логічно пов'язані зі значеннями. Такі масиви називають асоціативними масивами.

Важливою особливістю PHP є те, що PHP, на відміну від інших мов, дозволяє створювати масиви будь-якої складності безпосередньо в тілі програми (скрипта).

Масиви можуть бути як одновимірними, так і багатовимірними. При зверненні до елементів простих індексованих масивів використовується цілочисельний індекс, що визначає позицію заданого елемента.

Прості одномірні масиви:

Узагальнений синтаксис елементів простого одновимірного масиву:

$Ім'я [індекс];

Масиви, індексами яких є числа, які починаються з нуля – це списки:

```
<? Php
/ / Простий спосіб ініціалізації масиву
$Names [0] = "Апельсин";
$Names [1] = "Банан";
$Names [2] = "Груша";
$Names [3] = "Помідор";
// Тут: Names - ім'я масиву, а 0, 1, 2, 3 - індекси масиву
?>
```

Доступ до елементів простих масивів (списків) здійснюється наступним чином:

```
<? Php
// Простий спосіб ініціалізації масиву
$Names [0] = "Апельсин";
$Names [1] = "Банан";
$Names [2] = "Груша";
$Names [3] = "Помідор";
// Тут: Names - ім'я масиву, а 0, 1, 2, 3 - індекси масиву

/ /Виведемо елементи масивів в браузер:
echo $Names [0]; // Значення елемента масиву names з індексом 0
echo $Names [3]; // Значення елемента масиву names з індексом 3
// Виводить:
// Апельсин
// Помідор
?>
```

З технічної точки зору різниці між простими масивами і списками немає.

Прості масиви можна створювати, не вказуючи індекс нового елемента масиву, це за вас зробить PHP. Ось приклад:

```
<? Php
// Простий спосіб ініціалізації масиву, без вказівки індексів
$Names [] = "Апельсин";
$Names [] = "Банан";
$Names [] = "Груша";
$Names [] = "Помідор";
```

```
// PHP автоматично присвоїть індекси елементів масиву, починаючи з 0

// Виводимо елементи масивів в браузер:
echo $Names [0]; // Висновок елемента масиву names з індексом 0
echo $Names [3]; // Висновок елемента масиву names з індексом 3
// Виводить:
// Апельсин
// Помідор
?>
```

У розглянутому прикладі ви можете додавати елементи масиву Names простим способом, тобто не вказуючи індекс елемента масиву:

```
$Names [] = "Яблуко";
```

Новий елемент простого масиву (списку) буде додано в кінець масиву. Надалі, з кожним новим елементом масиву, індекс буде збільшуватися на одиницю.

Прості багатовимірні масиви:

Узагальнений синтаксис елементів багатовимірного простого масиву:

```
$Ім'я [індекс1] [індекс2] .. [індексN];
```

Приклад простого багатовимірного масиву:

```
<? Php
// Багатомірний простий масив:
$Arr [0] [0] = "Овочі";
$Arr [0] [1] = "Фрукти";
$Arr [1] [0] = "Абрикос";
$Arr [1] [1] = "Апельсин";
$Arr [1] [2] = "Банан";
$Arr [2] [0] = "Огірок";
$Arr [2] [1] = "Помідор";
$Arr [2] [2] = "Гарбуз";

// Виводимо елементи масиву:
echo "<h3>". $Arr [0] [0 ].":</ h3> ";
for ($q = 0; $q <= 2; $q + +) {
echo $Arr [2] [$ q]. "";
}
echo "<h3>". $Arr [0] [1 ].":</ h3> ";
for ($w = 0; $w <= 2; $w + +) {
echo $Arr [1] [$ w]. "<br>";
}
?>
```

Асоціативні масиви в PHP

У PHP індексом масиву може бути не тільки число, але і рядок. Причому на такий рядок не накладаються ніякі обмеження: він може містити пробіли, довжина такого рядка може бути будь-яка.

Асоціативні масиви особливо зручні в ситуаціях, коли елементи масиву зручніше пов'язувати зі словами, а не з числами.

Отже, масиви, індексами яких є рядки, називаються асоціативними масивами.

Одномірні асоціативні масиви:

Одномірні асоціативні масиви містять тільки один ключ (елемент), відповідний конкретному індексу асоціативного масиву. Наведемо **приклад:**

```
<? Php
// Асоціативний масив
$Names ["Іванов"] = "Іван";
$Names ["Сидоров"] = "Микола";
$Names ["Петров"] = "Петро";
// У даному прикладі: прізвища - ключі асоціативного масиву
//, А імена - елементи масиву Names
?>
```

Доступ до елементів одновимірних асоціативних масивів здійснюється так само, як і до елементів звичайних масивів, і називається доступом по ключу:

```
echo $Names ["Іванов"];
```

Багатовимірні асоціативні масиви:6

Багатовимірні асоціативні масиви можуть містити кілька ключів, які відповідають конкретним індексам асоціативного масиву. Розглянемо приклад багатовимірного асоціативного масиву:

```
<? Php
// Багатомірний масив
$A ["Ivanov"] = array ("name" => "Іванов І.І.", "age" => "25", "email" =>
"ivanov@mail.ru");
$A ["Petrov"] = array ("name" => "Петров П.П.", "age" => "34", "email" =>
"petrov@mail.ru");
$A ["Sidorov"] = array ("name" => "Сидоров С.С.", "age" => "47", "email" =>
"sidorov@mail.ru");
?>
```

Багатовимірні масиви схожі на записи у мові Pascal або структури в мові С.

Доступ до елементів багатовимірного асоціативного масиву здійснюється наступним чином:

```
echo $A ["Ivanov"] ["name"]; // Виводить Іванов І.І.
echo $A ["Petrov"] ["email"]; // Виводить petrov@mail.ru
```

Як ви вже помітили, для створення багатовимірного асоціативного масиву ми використовували спеціальну функцію array, ми її розглянемо пізніше, коли будемо розглядати операції над масивами.

Асоціативні багатовимірні масиви можна створювати і класичним способом, хоча це не так зручно:

```
<? Php
// Багатомірний асоціативний масив
```

```
$A ["Ivanov"] ["name"] = "Іванов І.І.";
$A ["Ivanov"] ["age"] = "25";
$A ["Ivanov"] ["email"] = "ivanov@mail.ru";

$A ["Petrov"] ["name"] = "Петров П.П.";
$A ["Petrov"] ["age"] = "34";
$A ["Petrov"] ["email"] = "petrov@mail.ru";

$A ["Sidorov"] ["name"] = "Сидоров С.С.";
$A ["Sidorov"] ["age"] = "47";
$A ["Sidorov"] ["email"] = "sidorov@mail.ru";

// Отримуємо доступ до ключів багатовимірного асоціативного масиву
echo $A ["Ivanov"] ["name "]."< br>"; // Виводить Іванов І.І.
echo $A ["Sidorov"] ["age "]."< br>"; // Виводить 47
echo $A ["Petrov"] ["email "]."< br>"; // Виводить petrov@mail.ru
?>
```

Тип object (об'єкти)

Об'єкти - тип даних, що прийшов з об'єктно-орієнтованого програмування (ООП). Згідно з принципами ООП, клас - це набір об'єктів, що володіють певними властивостями і методами роботи з ним, а об'єкт відповідно - екземпляр класу. Наприклад, програмісти - це клас людей, які пишуть програми, вивчають комп'ютерну літературу і, крім того, як всі люди, мають ім'я та прізвище. Тепер, якщо взяти одного конкретного програміста, Васю Іванова, то можна сказати, що він є об'єктом класу програмістів, має ті ж властивості, що й інші програмісти, теж має ім'я, пише програми і т.п.

У PHP для доступу до методів об'єкта використовується оператор ->. Для ініціалізації об'єкту використовується вираз new, що створює в змінній екземпляр об'єкта.

Приклад 8. Об'єкти в PHP

```
<? Php
// Створюємо клас людей
class Person
{
// Метод, який навчає людину PHP
function know_php ()
{
echo "Тепер я знаю PHP";
}
}
$Bob = new Person; // створюємо об'єкт
// Класу людина
$ Bob -> know_php (); // навчаємо його PHP
?>
```

Більш докладно реалізацію принципів ООП в мові PHP ми розглянемо в одній з наступних лекцій.

Тип resource (ресурси)

Ресурс - це спеціальна змінна, що містить посилання на зовнішній ресурс (наприклад, з'єднання з базою даних). Ресурси створюються та використовуються спеціальними функціями (наприклад, mysql_connect (), pdf_new () і т.п.).

Тип Null

Спеціальне значення NULL говорить про те, що змінна не має значення.

Змінна вважається NULL, якщо:

- їй була присвоєна константа NULL ($var = NULL);
- їй ще не було присвоєно будь-яке значення;
- вона була вилучена за допомогою unset ().

Існує тільки одне значення типу NULL - регістронезалежне ключове слово NULL.

2.4. Керуючі конструкції, Цикли.

Оператор if. Це один з найважливіших операторів багатьох мов, включаючи PHP. Він дозволяє виконувати фрагменти коду в залежності від умови. Структуру оператора if можна представити наступним чином:

if (вираз) блок_виконання

Тут вираз є будь-який правильний PHP-вираз (тобто все, що має значення). У процесі обробки скрипта вираз перетвориться до логічного типу. Якщо в результаті перетворення значення виразу істинно (True), то виконується блок_виконання. В іншому випадку блок_виконання ігнорується. Якщо блок_виконання містить кілька команд, то він повинен бути укладений у фігурні дужки {}.

Правила перетворення виразу до логічного типу:

1. Правила перетворення виразу до логічного типу:

- логічне False
- цілий нуль (0)
- дійсний нуль (0.0)
- порожній рядок і рядок "0"
- масив без елементів
- об'єкт без змінних (детально про об'єкти буде розказано в одній з наступних лекцій)
- спеціальний тип NULL

2. Всі інші значення перетворюються в TRUE.

Приклад 9. Умовний оператор if

```
<?
$Names = array ("Іван", "Петро", "Семен");
if ($Names[0] == "Іван") {
```

```
echo "Привіт, Ваня!";
$Num = 1;
$Account = 2000;
}
if ($num) echo "Іван перший у списку!";
$Bax = 30;
if ($Account > 100 * $Bax +3)
echo "Цей рядок не з'явиться
на екрані, так як умова не виконана ";
?>
```

Оператор else. Ми розглянули тільки одну, основну частину оператора if. Існує кілька розширень цього оператора. Оператор else розширює if на випадок, якщо вираз, що перевіряється в if є невірним, і дозволяє виконати будь-які дії за таких умов.

Структуру оператора if, розширеного за допомогою оператора else, можна представити таким чином:

```
if (вираз) блок_виконання
else блок_виконання1
```

Цю конструкцію if ... else можна інтерпретувати приблизно так: якщо виконана умова (тобто вираз = true), то виконуємо дії з блоку_виконання, інакше - дії з блоку_виконання1. Використовувати оператор else не обов'язково.

Подивимося, як можна змінити попередній приклад, з огляду на необхідність здійснення дій в разі невиконання умови.

Приклад 10. Оператор else

```
<?
$Names = array ("Іван", "Петро", "Семен");
if ($Names [0] == "Іван") {
echo "Привіт, Ваня!";
$Num = 1;
$Account = 2000;
}Else {
echo "Привіт, $Names [0].
А ми чекали Ваню: (";
}
if ($Num) echo "Іван перший у списку!";
else echo "Іван НЕ перший у списку?!";
$Bax = 30;
if ($Account > 100 * $ bax +3)
echo "Цей рядок не з'явиться на екрані,
так як умова не виконана ";
else echo "Зате з'явиться цей рядок!";
?>
```

Оператор elseif. Ще один спосіб розширення умовного оператора if - використання оператора elseif. Elseif - це комбінація else і if. Як і else, він розширює if для виконання різних дій у тому випадку, якщо умова, що перевіряється в if, невірно. Але на відміну від else, альтернативні дії будуть виконані, тільки якщо elseif-умова є вірним. Структуру оператора if, розширеного за допомогою операторів else і elseif, можна представити таким чином:

if (вираз) блок_виконання
elseif (вираз1) блок_виконання1
...
else блок_ виконанняN

Операторів elseif може бути відразу кілька в одному if-блоці. Elseif-твердження буде виконано, тільки якщо попереднє if-умова є False, всі попередні elseif-умови є False, а дане elseif-умова - True.

Приклад 11. Оператор elseif

```
<?
$Names = array ("Іван", "Петро", "Семен");
if ($Names [0] == "Іван") {
// Якщо перше ім'я в масиві Іван
echo "Привіт, Ваня!";
} Elseif ($Names [0] == "Петро") {
// Якщо перше ім'я
// Не Іван, а Петро
echo "Привіт, Петя!";
} Elseif ($Names [0] == "Семен") {
// Якщо перше ім'я не
// Іван, не Петро, а Семен
echo "Привіт, Сеня!";
} Else {
// Якщо перше ім'я не Іван,
// Не Петро І не Семен
echo "Привіт, $ Names [0]. А ти хто такий?";
}
?>
```

Оператор switch. Ще одна конструкція, що дозволяє перевіряти умови і виконувати в залежності від цього різні дії, - це **switch**. У залежності від того, яке значення має змінна, він перемикається між різними блоками дії. switch дуже схожий на оператор if ... elseif ... else або набір операторів if. Структуру switch можна записати наступним чином:

```
switch (вираз чи змінна) {
case значення1:
блок_дій1
break;
```

```
case значення2:
блок_дій2
break;
 ...
default:
блок_дій_при_замовчуванні
}
```

На відміну від if, тут значення виразу не приводиться до логічного типу, а просто порівнюється зі значеннями, перерахованими після ключових слів case (значення1, значення2 і т.д.). Якщо значення виразу співпало з якимсь варіантом, то виконується відповідний блок_дій - від двокрапки після значення, що співпало до кінця switch або до першого оператора break, якщо такий знайдеться. Якщо значення виразу не співпало з жодним із варіантів, то виконуються дії за умовчанням (блок_дій_при_замовчуванні), що знаходяться після ключового слова default. Вираз в switch обчислюється тільки один раз, а в операторі elseif - кожен раз, тому, якщо вираз досить складний, то switch працює швидше.

Приклад 11 можна переписати з використанням **switch** наступним чином:

```
<?
$ Names = array ("Іван", "Петро", "Семен");
switch ($Names [0]) {
case "Іван":
echo "Привіт, Ваня!";
break;
case "Петро":
echo "Привіт, Петя!";
break;
case "Семен":
echo "Привіт, Сеня!";
break;
default:
echo "Привіт, $Names [0].
А як Вас звати? ";
} ?>
```

Якщо в цьому прикладі опустити оператор break, наприклад, в case "Петро":, то, якщо змінна виявиться рівною рядку "Петро", після виведення на екран повідомлення "Привіт, Петя!" програма піде далі і виведе також повідомлення "Привіт, Сеня!" і тільки потім, зустрівши break, продовжить своє виконання за межами switch.

Для конструкції switch, як і для if, можливий альтернативний синтаксис, де відкривається switch фігурна дужка замінюється двокрапкою, а закриває - endswitch; відповідно.

2.4. Керуючі конструкції, Цикли.

У PHP існує кілька конструкцій, що дозволяють виконувати повторювані дії в залежності від умови. Це цикли while, do .. while, foreach та for. Розглянемо їх більш докладно.

while

Структура:

while (вираз) {блок_виконання}

або

while (вираз): блок_виконання endwhile;

While - простий цикл. Він наказує PHP виконувати команди блоку_виконання до тих пір, поки вираз обчислюється як True (тут, як і в if, відбувається приведення вислову до логічного типу). Значення виразу перевіряється щоразу на початку циклу, так що, навіть якщо його значення змінилося в процесі виконання блоку_виконання, цикл не буде зупинено до кінця ітерації (тобто поки всі команди блоку_виконання не будуть виконані).

Приклад 12. Оператор while

```
<?
// Ця програма надрукує всі парні цифри
$I = 1;
while ($I <10) {
if ($I % 2 == 0) print $ i;
// Друкуємо цифру, якщо вона парна
$I + +;
// І збільшуємо $I на одиницю
}
?>
```

do ... while

Цикли do .. while дуже схожі на цикли while, з тією лише різницею, що істинність висловлювання перевіряється наприкінці циклу, а не на початку. Завдяки цьому блок_виконання циклу **do ... while** гарантовано виконується хоча б один раз.

Структура:

do {блок_виконання} while (вираз);

Приклад 13. Оператор do .. while

```
// Ця програма надрукує число 12, незважаючи на те
// Що умова циклу не виконано
$I = 12;
do {
if ($I % 2 == 0) print $I;
// Якщо число парне, то друкуємо його
$I++;
// Збільшуємо число на одиницю
} While ($I <10)
?>
```

for

Це найскладніші цикли в PHP. Вони нагадують відповідні цикли C.

Структура:

for (вираз1; вираз2; вираз3) {блок_виконання}
або
for (вираз1; вираз2; вираз3): блок_виконання endfor;

Тут, як ми бачимо, умова складається одразу з трьох виразів. Перший вираз вираз1 обчислюється безумовно один раз на початку циклу. На початку кожної ітерації обчислюється вираз2. Якщо він є True, то цикл продовжується і виконуються всі команди блоку_виконання. Якщо вираз2 обчислюється як False, то виконання циклу зупиняється. В кінці кожної ітерації (тобто після виконання всіх команд блоку_виконання) обчислюється вираз3.

Кожне з виразів 1, 2, 3 може бути порожнім. Якщо вираз2 є порожнім, то це значить, що цикл повинен виконуватися невизначений час (у цьому випадку PHP вважає це вираз завжди істинним). Це не так марно, як здається, адже цикл можна зупиняти, використовуючи оператор break.

Наприклад, всі парні цифри можна вивести з використанням циклу for таким чином:

```
<? Php
for ($i = 0; $ i <10; $i + +) {
if ($i% 2 == 0) print $i;
// Друкуємо парні числа
} ?>
```

Якщо опустити другий вираз (умова $ i <10), то таку ж задачу можна вирішити, зупиняючи цикл оператором break.

```
< ? Php
for ($i = 0;; $ i + +) {
if ($i> = 10) break;
// Якщо $i більше або дорівнює 10,
// То припиняємо роботу циклу
if ($i% 2 == 0) print $ i;
// Якщо число парне,
// То друкуємо його
} ?>
```

Можна опустити всі три вирази. У цьому випадку просто не буде задано початкове значення лічильника $i і воно не буде змінюватися кожного разу наприкінці циклу. Всі ці дії можна записати у вигляді окремих команд або в блоці_виконання, або перед циклом:

```
<? Php
$i = 2; // задаємо початкове значення лічильника
for (;;) {
if ($i> = 10) break;
// Якщо $i більше або дорівнює 10,
// То припиняємо роботу циклу
if ($ i% 2 == 0) print $i;
```

```
// Якщо число парне,
// То друкуємо його
$i+ +; // збільшуємо лічильник на одиницю
}
?>
```

У третє вираз конструкції for можна записувати через кому відразу кілька найпростіших команд. Наприклад, якщо ми хочемо просто вивести всі цифри, то програму можна записати зовсім просто:

```
<? Php
for ($i = 0; $i <10; print $i, $i + +)
/ * Якщо блок_виконання не містить команд
або містить тільки одну команду,
фігурні дужки, в які він укладений,
можна опускати * /
?>
```

foreach

Ще одна корисна конструкція. Вона з'явилася тільки в PHP4 і призначена виключно для роботи з масивами.

Синтаксис:

```
foreach ($array as $value) {блок_виконання}
або
foreach ($array as $key => $value)
{Блок_виконання}
```

У першому випадку формується цикл по всіх елементах масиву, заданого змінною $array. На кожному кроці циклу значення поточного елемента масиву записується в змінну $value, і внутрішній лічильник масиву пересувається на одиницю (так що на наступному кроці буде записано наступний елемент масиву). Усередині блоку_виконання значення поточного елемента масиву може бути отримано за допомогою змінної $value. Виконання блоку_виконання відбувається стільки разів, скільки елементів в масиві $array.

Друга форма запису на додаток до перерахованого вище на кожному кроці циклу записує ключ поточного елемента масиву в змінну $key, яку теж можна використовувати в блоці_виконання.

Приклад 14. Оператор foreach

Коли foreach починає виконання, внутрішній покажчик масиву автоматично встановлюється на перший елемент.

```
<? Php
$Names = array ("Іван", "Петро", "Семен");
foreach ($Names as $ val) {
echo "Привіт, $val <br>";
// Виведе всім вітання
}
foreach ($Names as $k => $val) {
// Крім привітання,
```

```
// Виведемо номера в списку, тобто ключі
echo "Привіт, $val!
Ти в списку під номером $k <br> ";
}
?>
```

2.5. Оператори передачі управління, Оператори включення.

Іноді потрібно негайно завершити роботу циклу або окремої його ітерації. Для цього використовують оператори break та continue.

Break

Приклад 15. Оператор break

```
<? Php
$I = 1;
while($ i) {
$N = rand (1,10);
// Генеруємо довільне число
// Від 1 до 10
echo "$I: $N ";
// Виводимо номер ітерації і
// Згенероване число
if ($N == 5) break;
/* Якщо було створене число 5,
то припиняємо роботу циклу. У цьому випадку
все, що знаходиться після цього рядка
всередині циклу, не буде виконана * /
echo "Цикл працює <br>";
$I++;
}
echo "<br> Число ітерацій циклу $ i"; ?>
```

Результатом роботи цього скрипта буде приблизно наступне:

```
1:7 Цикл працює
2:2 Цикл працює
3:5
Число ітерацій циклу 3
```

Трохи змінимо наш скрипт:

```
<? Php
$i = 1;
while ($i) {
$n = rand (1,10);
// Генеруємо довільне число
// Від 1 до 10
switch ($n) {
case 5:
echo "<font color=blue>
```

```
Вихід з switch (n = $n) ";
break 1;
// Припиняємо роботу switch
// (Один break циклу)
case 10:
echo "<font color=red>
Вихід з switch і
while (n = $n) ";
break 2;
// Припиняємо роботу switch і while
// (Два break циклів)
default:
echo "switch працює (n = $n),";
}
echo "while працює - крок $i <br>";
$i+ +;
}
echo "<br> Число ітерацій циклу $i";
?>
```

continue

Іноді потрібно не повністю припинити роботу циклу, а тільки почати його нову ітерацію. Оператор continue дозволяє пропустити подальші інструкції з блоку_виконання будь-якого циклу і продовжити виконання з нового кола. continue можна використовувати з числовим аргументом, який вказує, скільки ітерацій циклу повинні завершити роботу.

Замінимо в прикладі попереднього параграфа оператор break на continue. Крім того, обмежимо кількість кроків циклу трьома.

```
<? Php
$i = 1;
while ($i <= 4) {
$n = rand (1,10);
// Генеруємо довільне число
// Від 1 до 10
echo "$i: $n";
// Виводимо номер ітерації і
// Згенероване число
if ($ n == 5) {
echo "Нова ітерація <br>";
continue;

/* Якщо було створене число 5,
то починаємо нову ітерацію циклу,
$i не збільшується */
}
```

```
echo "Цикл працює <br>";
$i+ +;
}
$i--;
echo "<br> Число ітерацій циклу $ i";
?>
```

Результатом роботи цього скрипта буде

```
1:10 Цикл працює
2:5 Нова ітерація
2:1 Цикл працює
3:1 Цикл працює
Число ітерацій циклу 4
```

Зауважимо, що після виконання оператора continue робота циклу не закінчується. У прикладі лічильник циклу не змінюється в разі отримання числа 5, оскільки він перебуває після оператора continue. Фактично за допомогою continue ми намагаємося уникнути ситуації, коли буде створене число 5. Тому можна було просто написати, замінивши оператор continue на перевірку істинності висловлювання:

```
<? Php
$I = 1;
while ($I <4) {
$N = rand (1,10);
// Генеруємо довільне число
// Від 1 до 10
if ($N! == 5) {
echo "$ i: $ n <br>";
// Виводимо номер ітерації
// І згенероване число
$I + +;
}
}
?>
```

У PHP існує одна особливість використання оператора continue - в конструкціях switch він працює так само, як і break. Якщо switch знаходиться всередині циклу й треба почати нову ітерацію циклу, слід використовувати continue 2.

Include. Оператор include дозволяє включати код, що міститься у вказаному файлі, і виконувати його стільки разів, скільки програма зустрічає цей оператор. Включення може здійснюватися будь-яким з перерахованих способів:

```
include 'ім'я_файлу';
include $file_name;
include ("ім'я_файлу");
```

Приклад 16. Нехай у файлі params.inc у нас зберігається набір якихось параметрів і функцій. Кожного разу, коли нам потрібно буде використовувати ці

параметри (функції), ми будемо додавати до тексту нашої основної програми команду include 'params.inc'.

Приклад 16. Використання оператора включення include
params.inc

```
<? Php
$User = "Вася";
$Today = date ("d.m.y");
/ * Функція date () повертає дату
і час (тут - дату в форматі
день.місяць.рік) * /
?>
```

include.php

```
 <? Php
include ("params.inc");
/* Змінні $User і $Today задані у файлі
params.inc. Тут ми теж можемо ними
користуватися завдяки команді
include ("params.inc") * /

echo "Привіт, $User! <br>";
// Виведе "Привіт, Вася!"
echo "Сьогодні $Today";
// Виведе, наприклад, "Сьогодні 24.01.11"
?>
```

Зауважимо, що використання оператора include еквівалентно простий вставці змістовної частини файлу params.inc в код програми include.php. Справа в тому, що в момент вставки файлу відбувається перемикання з режиму обробки PHP в режим HTML. Тому код всередині файлу, який потрібно обробити як PHP-скрипт, повинен бути укладений у відповідні теги.

Пошук файлу для вставки відбувається за такими правилами.

1. Спочатку ведеться пошук файлу в include_path поточної робочої директорії.
2. Якщо файл не знайдений, то пошук виконується в include_path директорії поточного скрипта.
3. Параметр include_path, визначається у файлі налаштувань PHP, задає імена директорій, в яких потрібно шукати файли, що включаються.

Наприклад, ваш include_path це поточна робоча директорія - / www /. В основний файл include.php ви включаєте файл my_dir / a.php, який у свою чергу включає b.php. Тоді парсер насамперед шукає файл b.php в директорії / www /, і якщо такого немає, то в директорії / www / my_dir /.

Якщо файл включений з допомогою include, то код що міститься в ньому успадковує область видимості змінних. Будь-які змінні включеного файлу

будуть доступні в файлі з цього рядка і далі. Відповідно, якщо include з'являється всередині функції файлу, який викликає, то код, що міститься в включеному файлі, буде вести себе так, як ніби він був визначений всередині функції. Таким чином, він успадкує область видимості цієї функції. Хоча ми і не знайомилися ще з поняттям функції, все ж наводимо тут ці відомості в розрахунку на інтуїтивне його розуміння.

Приклад 17. Область видимості при використанні include .Нехай файл для вставки params.inc залишиться таким же, а include.php буде наступним:

```
<? Php
function Footer () {
// Оголошуємо функцію з ім'ям Footer
include ("params.inc");
/* Включаємо файл params.inc.
Тепер його змінними можна користуватися,
але тільки всередині функції */
$str = "Сьогодні: $today <br>";
$str .= "<a
href = 'mailto: help@intuit.ru'> Сторінку
створив $user ";
echo "$str";
}
Footer();
// Викликаємо функцію Footer (). Отримаємо:
// Сьогодні: 08.07.05
// Сторінку створив Вася

echo "$user, $today";
// Нічого не виведе, так як
// Ці змінні видно тільки
// Всередині функції
 ?>
```

Крім локальних файлів, за допомогою include можна включати і зовнішні файли, вказуючи їх url-адреси. Дана можливість контролюється директивою url_fopen_wrappers у файлі налаштувань PHP і за замовчуванням, як правило, включена. Але у версіях PHP для Windows до PHP 4.3.0 ця можливість не підтримується зовсім, незалежно від url_fopen_wrappers.

include () - це спеціальна мовна конструкція, тому при використанні всередині блоків її потрібно укладати у фігурні дужки.

Приклад 18. Використання include ()

```
<? Php
/* Це невірний запис. Отримаємо помилку.
Ми ж вставляємо не одну команду,
а декілька, вони тільки записані
в іншому файлі */
```

```
if ($condition) include ("first.php");
else include ("second.php");
// А ось так правильно.
if ($ condition) {include ("first.php");}
else {include ("second.php");}
?>
```

При використанні include можливо два види помилок - помилка вставки (наприклад, не можна знайти вказаний файл, невірно написана сама команда вставки тощо) або помилка виконання (якщо помилка міститься у файлі, що вставляється). У будь-якому випадку при помилці в команді include виконання скрипта не завершується.

Require. Цей оператор діє приблизно так само, як і #include в C+ +. Все, що ми говорили про include, лише за деякими винятками, справедливо і для require. require також дозволяє включати в програму і виконувати який-небудь файл. Основна відмінність require і include полягає в тому, як вони реагують на виникнення помилки. Як вже говорилося, include видає попередження, і робота скрипта триває. Помилка в require викликає фатальну помилку роботи скрипта і припиняє його виконання.

Умовні оператори на require () не впливають. Якщо рядок, в якому з'являється цей оператор, не виконується, то жоден рядок коду з файлу теж не виконується. Цикли також не впливають на require (). Код, що міститься у файлі який є об'єктом циклу,вставляється але вставка відбувається тільки один раз.

У реалізаціях PHP до версії 4.0.2 використання require () означало, що інтерпретатор обов'язково спробує прочитати файл.

require, як і include, при використанні всередині умовних блоків потрібно укладати у фігурні дужки.

Альтернативний синтаксис

PHP пропонує альтернативний синтаксис для деяких своїх керуючих структур, а саме для if, while, for, foreach і switch. У кожному разі відкриваючу фігурну дужку потрібно замінити на двокрапку (:), а закриваючу - на endif;, endwhile, і т.д. відповідно.

Наприклад, синтаксис оператора if можна записати таким чином: if (вираз): блок_виконання endif;

Сенс залишається тим же: якщо умова, що записана в круглих дужках оператора if, виявилась істиною, буде виконуватися весь код, від двокрапки ":" до команди endif;. Використання такого синтаксису корисно при встроюванні php в html-код.

Приклад 19. Використання альтернативного синтаксису

```
<? Php
$ Names = array ("Іван", "Петро", "Семен");
if ($Names [0] == "Іван"):
?>
Привіт, Ваня!
```

```
<? Php
endif;
?>
```

Якщо використовуються конструкції else і elseif, то також можна задіяти альтернативний синтаксис:

```
<? Php
$a = 1;
if ($a == 5):
print "a дорівнює 5";
print "...";
elseif ($ a == 6):
print "a дорівнює 6";
print "!!!";
else:
print "a не дорівнює ні 5, ні 6";
endif;
?>
```

2.6. Функції користувача, символічні і жорсткі посилання.

Для чого потрібні функції? Щоб відповісти на це питання, потрібно зрозуміти, що взагалі являють собою функції. У програмуванні, як і в математиці, функція є відображення безлічі її аргументів на безліч її значень. Тобто функція для кожного набору значень аргументу повертає якісь значення, що є результатом її роботи. Навіщо потрібні функції, спробуємо пояснити на прикладі. Класичний приклад функції у програмуванні - це функція, що обчислює значення факторіала числа. Тобто ми задаємо їй число, а вона повертає нам його факторіал. При цьому не потрібно для кожного числа, факторіал якого ми хочемо отримати, повторювати один і той самий код - досить просто викликати функцію з аргументом, рівним цьому числу.

Функція обчислення факторіала натурального числа

```
<? Php
function fact ($n) {
if ($n == 0) return 1;
else return $fact = $ n * fact ($ n-1);
}
echo fact (3);
// Можна було б написати echo (3 * 2);
// Але якщо число велике,
echo fact (50);
// То зручніше користуватися функцією,
// Ніж писати echo (50 * 49 * 48 *...* 3 * 2);
?>
```

Таким чином, коли ми здійснюємо дії, в яких простежується залежність від будь-яких даних, і при цьому нам знадобиться виконувати такі ж дії, але з

іншими вихідними даними, зручно використовувати механізм функцій - оформити блок дій у вигляді тіла функції, а змінні дані - як її параметр.

Подивимося, як у загальному вигляді виглядає оголошення функції. Функція може бути визначена за допомогою наступного синтаксису:

function Імя_функціі (параметр1, параметр2, ... параметрN) {
Блок_дій
return "значення повертається функцією";
}

Якщо прямо так написати в php-програмі, то працювати нічого не буде. По-перше, Імя_функції і імена параметрів функції (параметр1, параметр2 і т.д.) повинні відповідати правилам найменування в PHP (і українських символів в них краще не використовувати). Імена функцій нечутливі до регістру. По-друге, параметри функції - це змінні мови, тому перед назвою кожної з них повинен стояти знак $. Ніяких крапок ставити в списку параметрів не можна. По-третє, замість слів блок_дій в тілі функції повинен знаходитися будь-який правильний PHP-код (не обов'язково залежати від параметрів). І нарешті, після ключового слова return має йти коректний php-вираз (що-небудь, що має значення). Крім того, у функції може і не бути параметрів, як і значення, що повертається. Приклад правильного оголошення функції - функція обчислення факторіала, наведена вище.

Як відбувається виклик функції? Вказується ім'я функції і в круглих дужках список значень її параметрів, якщо такі є:

<? Php
Імя_функції ("значення_для_параметра1",
"Значення_для_параметра2 ",...);
// Приклад виклику функції - виклик функції
// Обчислення факторіала наведено вище,
// Там для обчислення факторіала числа 3
// Ми писали: fact (3);
// Де fact - ім'я викликається функції,
// А 3 - значення її параметра з ім'ям $ n
?>

Коли можна викликати функцію? Функцію можна викликати після її визначення, тобто в будь-якому рядку програми нижче блоку function f_name (){...}. У PHP3 це було дійсно так. Але вже в PHP4 такої вимоги немає. Вся справа в тому, як інтерпретатор обробляє одержуваний код. Єдиний виняток становлять функції, які визначаються умовно (всередині умовних операторів або інших функцій). Коли функція визначається таким чином, її визначення повинно передувати її виклику.

Приклад 20. Визначення функції всередині умовного оператора

<? Php
$Make = true;
/* Тут не можна викликати Make_event ();
тому що вона ще не існує, але можна

```
викликати Save_info () * /

Save_info ("Вася", "Іванов",
"Я вибрав курс по PHP");

if ($Make) {
// Визначення функції Make_event ()
function Make_event () {
echo "<p> Хочу вивчати Python <br>";
}
}
// Тепер можна викликати Make_event ()
Make_event ();
// Визначення функції Save_info
function Save_info ($first, $last, $message) {
echo "<br> $ message <br>";
echo "Ім'я:". $first. "". $last. "<br>";
}
Save_info ("Федя", "Федоров",
"А я вибрав Lisp");
// Save_info можна викликати і тут
?>
```

Якщо функція одного разу визначена в програмі, то перевизначити або видалити її пізніше не можна. Незважаючи на те, що імена функцій нечутливі до регістру, краще викликати функцію з того ж імені, яким вона була задана у визначенні.

Приклад 21. Визначення функції усередині функції

```
<? Php
/* Не можна зберегти дані, тобто викликати
функцію DataSave () до того, як виконана
перевірка їх правильності, тобто викликана
функція DataCheck () */

DataCheck ();
DataSave ();

function DataCheck () {
// Перевірка правильності даних
function DataSave () {
// Зберігаємо дані
}
}
?>
```

Розглянемо докладніше аргументи функцій, їх призначення та використання.

Аргументи функцій

У кожній функції може бути список аргументів. За допомогою цих аргументів у функцію передається різна інформація (наприклад, значення числа, факторіал якого треба підрахувати). Кожен аргумент являє собою змінну або константу.

За допомогою аргументів дані у функцію можна передавати трьома різними способами. Це передача аргументів значенням (використовується за замовчуванням), по посиланню й завдання значення аргументів за замовчуванням. Розглянемо ці способи докладніше.

Коли аргумент передається у функцію за значенням, зміна значення аргументу всередині функції не впливає на його значення поза функцією. Щоб дозволити функції змінювати аргументи, їх потрібно передавати по посиланню. Для цього у визначенні функції перед ім'ям аргументу слід написати знак амперсанд «&».

Приклад 22. Передача аргументів за посиланням

```
<? Php
// Напишемо функцію, яка б додавала
// До рядка слово checked
function add_label (& $data_str) {
$data_str .= "checked";
}
$str = "<input type = radio name = article";
// Нехай є такий рядок
echo $str. "> <br>";
// Виведе елемент форми -
// Не вибрану радіо кнопку
add_label ($str);
// Викличемо функцію
echo $ str. "> <br>";
// Це виведе вже вибрану
// Радіо кнопку
?>
```

У функції можна визначати значення аргументів, які використовуються за замовчуванням. Значення за замовчанням має бути константним виразом, а не змінною і не представником класу або викликом іншої функції.

У нас є функція, яка створює інформаційне повідомлення, підпис до якого змінюється в залежності від значення переданого їй параметра. Якщо значення параметра не задано, то використовується підпис "Оргкомітет".

Приклад 23. Значення аргументів за замовчуванням

```
<? Php
function Message ($ sign = "Оргкомітет.") {
// Тут параметр sign має за замовчуванням значення "Оргкомітет"
```

```
echo "Наступне зібрання відбудеться завтра. <br>";
echo $sign. "<br>";
}
Message ();
// Викликаємо функцію без параметра.
// У цьому випадку підпис - це Оргкомітет
Message ("З повагою, Вася");
// У цьому випадку підпис
// Буде "З повагою, Вася."
?>
```

Результатом роботи цього скрипта буде:

```
Наступне зібрання відбудеться завтра.
Оргкомітет.
Наступне зібрання відбудеться завтра.
З повагою, Вася.
```

Якщо у функції декілька параметрів, то ті аргументи, для яких задаються значення за замовчуванням, повинні бути записані після всіх інших аргументів у визначенні функції. В іншому випадку з'явиться помилка, якщо ці аргументи будуть опущені при виклику функції.

Наприклад, ми хочемо внести опис статті в каталог. Користувач повинен ввести такі характеристики статті, як її назва, автор та короткий опис. Якщо користувач не вводить ім'я автора статті, вважаємо, що це Іванов Іван.

```
<? Php
function Add_article ($title, $description,
$author = "Іванов Іван") {
echo "Заносимо в каталог статтю: $title,";
echo "автор $author";
echo "<br> Короткий опис:";
echo "$description <hr>";
}
Add_article ("Інформатика і ми",
"Це стаття про інформатику ...",
"Петров Петро");
Add_article ("Хто такі хакери",
"Це стаття про хакерів ...");
?>
```

У результаті роботи скрипта одержимо наступне

```
Заносимо в каталог статтю: Інформатика і ми,
автор Петров Петро.
Короткий опис:
Це стаття про інформатику ...
Заносимо в каталог статтю: Хто такі хакери,
автор Іванов Іван.
```

Короткий опис:
Це стаття про хакерів ...

Якщо ж ми напишемо ось так:

```
<? Php
function Add_article ($author = "Іванов Іван",
$title, $description) {
// ... Дії як у попередньому прикладі
}
Add_article ("Хто такі хакери",
"Це стаття про хакерів ...");
?>
```

То в результаті отримаємо:

```
Warning: Missing argument 3 for
add_article () in
c: \ users \ nina \ tasks \ func \ def_bad.php
on line 2
```

Списки аргументів змінної довжини

В PHP4 можна створювати функції зі змінним числом аргументів. Тобто ми створюємо функцію, не знаючи заздалегідь, зі скількома аргументами її викличуть. Для написання такої функції ніякого спеціального синтаксису не потрібно. Все робиться за допомогою вбудованих функцій func_num_args (), func_get_arg (), func_get_args ().

Функція func_num_args () повертає число аргументів, переданих в поточну функцію. Ця функція може використовуватися тільки усередині тіла функції користувача. Якщо вона з'явиться поза функцією, то інтерпретатор видасть попередження.

Приклад 24. Використання функції func_num_args ()

```
<? Php
function DataCheck () {
$N = func_num_args ();
echo "Кількість аргументів функції $N";
}
DataCheck ();
// Виведе рядок
// "Число аргументів функції 0"
DataCheck (1,2,3);
// Виведе рядок
// "Число аргументів функції 3"
?>
```

Функція func_get_arg (ціле номер_аргумента) повертає аргумент зі списку переданих у функцію аргументів, порядковий номер якого заданий параметром номер_аргумента. Аргументи функції починаються з нуля. Як і func_num_args (), ця функція може використовуватися тільки усередині тіла якої-небудь функції.

Номер_аргумента не може перевищувати число аргументів, переданих у функцію. Інакше буде згенероване попередження, і функція func_get_arg () поверне False.

Створимо функцію для перевірки типу даних її аргументів. Вважаємо, що перевірка пройшла успішно, якщо перший аргумент функції - ціле число, другий - рядок.

Приклад 25. Функція для перевірки типу даних, її аргументів

```
<?
function DataCheck () {
$Check = true;
$n = func_num_args ();
// Число аргументів,
// Переданих у функцію
/* Перевіряємо, чи є першим
переданий аргумент цілим числом */
if ($ n> = 1) if (! is_int (func_get_arg (0)))
$Check = false;
/* Перевіряємо, чи є другий
переданий аргумент рядком */
if ($n> = 2)
if (!is_string (func_get_arg (1)))
$Check = false;
return $Check;
}

if (DataCheck (123, "text"))
echo "Перевірка пройшла успішно <br>";
else echo "Дані не задовольняють
умовам <br> ";
if (DataCheck (324))
echo "Перевірка пройшла успішно <br>";
else echo "Дані не задовольняють умовам <br>";
?>
```

Результатом роботи буде наступне.

```
Перевірка пройшла успішно
Перевірка пройшла успішно
```

Функція func_get_args () повертає масив, що складається зі списку аргументів, переданих функції. Кожен елемент масиву відповідає аргументу, переданому функції. Якщо функція використовується поза тілом функції, то генерується попередження.

Перепишемо попередній приклад, використовуючи цю функцію. Будемо перевіряти, чи є цілим числом кожен парний аргумент, переданий функції:

```
<?
function DataCheck () {
```

```
$Check = true;
$N = func_num_args ();
// Число аргументів,
// Переданих у функцію

$Args = func_get_args ();
// Масив аргументів функції
for ($i = 0; $i <$ n; $i + +) {
$V = $ args [$ i];
if($ i% 2 == 0) {
if(!is_int ($V)) $ check = false;
// Перевіряємо,
// Чи є парний аргумент цілим
}
}
return $ check;
}
if (DataCheck (array ("text", 324)))
echo "Перевірка пройшла успішно <br>";
else echo "Дані не задовольняють
умовам <br> ";
?>
```

Як бачимо, комбінації функцій func_num_args (), func_get_arg () і func_get_args () використовується для того, щоб функції могли мати змінний список аргументів. Ці функції були додані тільки в PHP 4. У PHP3 для того, щоб домогтися подібного ефекту, можна використовувати в якості аргументу функції масив. Наприклад, ось так можна написати скрипт, який перевіряє, чи є кожен непарний параметр функції цілим числом:

```
<?
function DataCheck ($params) {
$Check = true;
$N = count ($ params);
// Число аргументів,
// Переданих у функцію

for ($i = 0; $i <$ n; $i + +) {
$V = $params [$ i];
if ($ i% 2! == 0) {
// Перевіряємо, чи є непарний
// Аргумент цілим
if (! is_int ($V)) $Check = false;
}
}
return $Check;
```

```
}
if (DataCheck (array ("text", 324)))
echo "Перевірка пройшла успішно <br>";
else echo "Дані не задовольняють умовам <br>";
?>
```

Використання змінних всередині функції

Глобальні змінні

Щоб використовувати всередині функції змінні, задані поза нею, ці змінні потрібно оголосити як глобальні. Для цього в тілі функції слід перерахувати їх імена після ключового слова global:

```
global $ var1, $ var2;
```

Приклад 26. Глобальні змінні

```
<?
$a = 1;
function Test_g () {
global $a;
$a = $a * 2;
echo 'в результаті роботи функції $ a =', $a;
}
echo 'поза функції $a =', $a, ',';
Test_g ();
echo "<br>";
echo 'поза функції $a =', $a, ',';
Test_g (); ?>
```

У результаті роботи цього скрипта одержимо:

```
поза функції $a = 1, в результаті роботи
функції $a = 2
поза функції $a = 2, в результаті роботи
функції $a = 4
```

Коли змінна оголошується як глобальна, фактично створюється посилання на глобальну змінну. Тому такий запис еквівалентний наступному (масив $GLOBALS містить всі змінні, глобальні щодо поточної області видимості):

```
$Var1 = & $GLOBALS ["var1"];
$Var2 = & $GLOBALS ["var2"];
```

Це означає що видалення змінної $var1 не видаляє глобальної змінної $GLOBALS ["var1"].

Статичні змінні

Щоб використовувати змінні тільки всередині функції, при цьому зберігаючи їх значення і після виходу з функції, потрібно оголосити ці змінні як статичні. Статичні змінні видно тільки всередині функції але вони не втрачають свого значення, якщо виконання програми виходить за межі функції. Оголошення таких змінних проводиться за допомогою ключового слова static:

```
static $var1, $var2;
```

Статичній змінній може бути присвоєно будь-яке значення, але не посилання.

Приклад 27. Використання статичної змінної

```
<?
function Test_s () {
static $a = 1;
// Можна присвоювати вираз або посилання
$a = $a * 2;
echo $a;
}
Test_s (); // виведе 2
echo $a; // нічого не виведе, так як
// $a доступна тільки
// Всередині функції
Test_s (); // всередині функції $a = 2, тому
// Результатом роботи функції
// Буде число 4
?>
```

Значення, що повертаються

Всі функції, наведені вище в якості прикладів, виконували будь-які дії. Окрім подібних дій, будь-яка функція може повертати як результат своєї роботи якесь значення. Це робиться за допомогою return. Значення, що повертається може бути будь-якого типу, включаючи списки і об'єкти. Коли інтерпретатор зустрічає команду return у тілі функції, він негайно припиняє її виконання і переходить на той рядок, з якої була викликана функція.

Наприклад, складемо функцію, яка повертає вік людини. Якщо людина не померла, то вік вважається відносно поточного року.

```
<? Php
/* Якщо другий параметр передається він сприймається
як true, то він розглядається як
дата смерті, */

function Age ($birth, $is_dead) {
if ($is_dead) return $is_dead-$ birth;
else return date ("Y") - $birth;
}
echo Age (1971, false); // для 2009 року виведе 38
echo Age (1971, 2001); // виведе 30
?>
```

У цьому прикладі можна було і не використовувати функцію return, а просто замінити її функцією виведення echo. Проте якщо ми все-таки робимо так, що функція повертає якесь значення (у даному випадку вік людини), то в програмі ми можемо присвоїти будь-якій змінній значення цієї функції:

```
$An_age = Age (1981, 2004);
```

У результаті роботи функції може бути повернуто лише одне значення. Кілька значень можна отримати, якщо повертати список значень (одновимірний масив). Припустимо, ми хочемо отримати повний вік людини з точністю до дня.

```
<? Php
function Full_age ($ b_day, $ b_month, $ b_year)
{
$y = date ("Y");
$m = intval (date ("m"));
$d = intval (date ("d"));
$B_month = intval($ b_month);
$B_day = intval($ b_day);
$B_year = intval($ b_year);

$day = ($b_day> $d? 30 - $b_day + $d: $d - $b_day);
$tmpMonth = ($b_day> $d? -1: 0);
$month = ($b_month> $m + $tmpMonth? 12 - $ b_month +
$tmpMonth + $m: $m + $tmpMonth - $b_month);
$tmpYear = ($b_month> $m + $tmpMonth? -1: 0);
if ($b_year> $y + $tmpYear)
{
$year = 0; $month = 0;$ day = 0;
}
else
{
$year = $y + $tmpYear - $b_year;
}
return array ($day, $month, $year);
}
$Age = Full_age ("29", "06", "1986");
echo "Вам $age [2] років, $ age[1] місяців і $age [0] днів";
?>
```

Коли функція повертає кілька значень для їх обробки в програмі, зручно використовувати мовну конструкцію list (), яка дозволяє однією дією присвоїти значення одразу кільком змінним. Наприклад, у попередньому прикладі, залишивши без зміни функцію, обробити повернене значення можна так:

```
<?
// Виклик функції Full_age ()
list ($ay, $ month, $ year) = Full_age ("07","08", "1974");
echo "Вам $year років, $month місяців і $day днів ";
?>
```

Взагалі конструкцію list () можна використовувати для присвоєння змінним значень елементів будь-якого масиву.

Приклад 28. Використання list ()

```
<?
$Arr = array ("first", "second");
list ($ a, $ b) = $Arr;
// Змінній $a присвоюється перше
// Значення масиву, $b - друге
echo $a, "", $b;
// Виведе рядок «first second»
?>
```

Повернення посилання

У результаті своєї роботи функція також може повертати посилання на будь-яку змінну. Це може стати в нагоді, якщо потрібно використовувати функцію для того, щоб визначити, якій змінній повинно бути присвоєно посилання. Щоб отримати з функції посилання, потрібно при оголошенні перед її ім'ям написати знак амперсанду (&) і щоразу при виклику функції перед її ім'ям теж писати амперсанд (&). Зазвичай функція повертає посилання на будь-яку глобальну змінну (або її частину - посилання на елемент глобального масиву), посилання на статичну змінну (або її частину) або посилання на один з аргументів, якщо він був також переданий по посиланню.

Приклад 29. Повернення посилання

```
<?
$a = 3; $ b = 2;
function & ref ($par) {
global $a, $b;
if ($par% 2 == 0) return $b;
else return $a;
}
$var = & ref (4);
echo $var, "i", $b, "<br>";
// Виведе 2 і 2
$b = 10;
echo $ var, "i", $ b, "<br>";
// Виведе 10 і 10 ?>
```

При використанні синтаксису посилань на змінну $var нашого прикладу не копіюється значення змінної $b повернутої функцією $ref, а створюється посилання на цю змінну. Тобто тепер змінні $var і $b ідентичні і будуть змінюватися одночасно.

Змінні функції

PHP підтримує концепцію змінних функцій. Це означає, що якщо ім'я змінної закінчується круглими дужками, то PHP шукає функцію з таким же ім'ям і намагається її виконати.

Приклад 30. Використання змінних функцій

```
<?
/* Створимо дві прості функції:
```

```
Add_sign - додає підпис до рядка і
Show_text - виводить рядок тексту * /

function Add_sign ($string,
$sign = "З повагою, Петро") {
echo $string. "". $sign;
}
function Show_text () {
echo "Відправити повідомлення поштою <br>";
}
$Func = "Show_text";
// Створюємо змінну зі значенням,
// Рівним імені функції Show_text
$Func ();
// Це викличе функцію Show_text
$Func = "Add_sign";
// Створюємо змінну зі значенням,
// Рівним імені функції Add_sign
$Func ("Привіт всім <br>");
// Це викличе функцію
// Add_sign з параметром "Привіт усім"
?>
```

У цьому прикладі функція Show_text просто виводить рядок тексту. Здавалося навіщо для цього створювати окрему функцію, якщо існує спеціальна функція echo (). Справа в тому, що такі функції, як echo (), print (), unset (), include () і т.п. не можна використовувати в якості змінних функцій. Тобто якщо ми напишемо:

```
<?
$Func = "echo";
$Func ("TEXT");
?>
```

то інтерпретатор виведе помилку:

```
Fatal error: Call to undefined function:
echo () in
c: \ users \ nina \ tasks \ func \ var_f.php on line 2
```

Тому для того, щоб використовувати будь-яку з перерахованих вище функцій як змінну функцію, потрібно створити власну функцію, що ми і зробили в попередньому прикладі.

Символічні і жорсткі посилання

Хоча в PHP немає такого поняття, як покажчик, все ж таки існує можливість створювати посилання на інші змінні. Існує два різновиди посилань: жорсткі і символічні (змінні) (перші часто називають просто посиланнями). Жорсткі посилання з'явилися в PHP версії 4 (в третій версії існували лише символічні посилання).

Посилання в PHP - це засіб доступу до вмісту однієї змінної під різними іменами. Вони не схожі на покажчики мови Сі і не є псевдонімами таблиці символів. У PHP ім'я змінної і її вміст - це різні речі, тому один вміст може мати різні імена. Найближча аналогія - імена файлів Unix і медіа - імена змінних є елементами каталогів, а вміст змінних це самі файли. Посилання в PHP - аналог жорстких посилань (hardlinks) у файлових системах Unix.

Жорсткі посилання в PHP

Жорстке посилання представляє собою просто змінну, яка є синонімом іншої змінної. Багаторівневі посилання (тобто, посилання на посилання змінної, як це можна робити, наприклад, у Perl) не підтримуються. Так що не варто сприймати жорсткі посилання серйозніше, ніж синоніми.

Щоб створити жорстке посилання, потрібно використовувати оператор & (амперсанд). Наприклад:

```
$a = 10;
$b = & $a; // тепер $b - те ж саме, що і $a
$b = 0; // насправді $a = 0
echo "b = $b, a = $a"; / / Виводить: "b = 0, a = 0"
```

Посилатися можна не тільки на змінні, але й на елементи масиву (цим жорсткі посилання вигідно відрізняються від символічних). Наприклад:

```
$a = array ('a' => 'aaa', 'b' => 'bbb');
$b = & $a ['b']; // тепер $b - те ж, що й елемент з індексом 'b' масиву
$b = 0; / / насправді $a ['b'] = 0;
echo $a ['b']; // Виводить 0
```

Втім, елемент масиву, для якого планується створити символічне посилання, може і не існувати. Як у наступному випадку:

```
$a = array ('a' => 'aaa', 'b' => 'bbb');
$b = & $a ['c']; // тепер $b - те ж, що й елемент з індексом 'c' масиву
echo "Елемент з індексом 'c': (". $a ['c']. ")";
```

В результаті виконання розглянутого скрипта, хоча посиланням $b і не було нічого присвоєно, в масиві $a створиться новий елемент з ключем c і значенням - порожній рядком (ми можемо це визначити за результатом роботи echo). Тобто, жорстке посилання на самому ділі не може посилатися на неіснуючий об'єкт, а якщо робиться така спроба, то об'єкт створюється.

Примітка: Якщо прибрати рядок, в якій створюється жорстке посилання, то буде виведено повідомлення про те, що елемент з ключем c не визначено в масиві $a.

Жорсткі посилання зручно застосовувати при передачі параметрів для функції користувача і повернення значення з неї.

Символічні посилання (змінні на змінні)

Символічне посилання - це всього лише рядкова змінна, що зберігає ім'я іншої змінної. Щоб дістатися до значення змінної, на яку посилається символічне посилання, необхідно застосувати додатковий знак $ перед ім'ям посилання. Розглянемо приклад:

```
$A = 10;
$B = 20;
$C = 30;
$P = "A"; / / або $P = "B" або $P = "C" (присвоюємо $p ім'я іншої змінної)
echo $$P; // виводить змінну, на яку посилається $P, тобто $A
$$P = 100; / / присвоює $A значення 100
```

Ми бачимо, що для того, щоб використовувати звичайну рядкову змінну як посилання, потрібно перед нею поставити ще один символ $. Це говорить інтерпретатору, що треба взяти не значення самої $P, а значення змінної, ім'я якої зберігається у змінній $P.

Символічні посилання (змінні на змінні) використовуються досить рідко.

Жорсткі посилання і призначені для функцій користувача

Передача значень за посиланням

Ви можете передавати змінні в функцію користувача по посиланню, якщо ви хочете дозволити функції модифікувати свої аргументи. У такому випадку, функція користувача зможе змінювати аргументи. Синтаксис такий:

```
<? Php
function foo (& $var)
{
$var + +;
}

$a = 5;
foo ($a);
// $a тут дорівнює 6
?>
```

Зауважте, що у виклику функції відсутній знак посилання - він є тільки у визначенні функції. Цього достатньо для коректної передачі аргументів за посиланням.

Ще один цікавий приклад:

```
<? Php
function funct (& $string)
{
$string .= 'а ця всередині.';
}
$str = 'Цей рядок за межами функції,';
funct ($str);
echo $ str; // Виведе 'Цей рядок за межами функції, а ця всередині.'
?>
```

За посиланням можна передавати:

- Змінні, наприклад foo ($ a)
- Оператор new, наприклад foo (new foobar ())

- Лінки, які повертаються функцією, наприклад:

```
<? Php
function & bar ()
{
$a = 5;
return $a;
}
foo (bar ());
?>
```

Будь-яке інше вираження не повинно передаватися по посиланню, так як результат не визначений. Наприклад, наступна передача за посиланням є неправильною:

```
<? Php
function bar () // Операція & відсутня
{
$a = 5;
return $a;
}
foo (bar ());

foo ($a = 5); // Вираз, а не змінна
foo (5); // Константа, а не змінна
?>
```

Повернення значень за посиланням

Розглянемо ще одну можливість функцій користувача PHP - повернення посилань.

Повернення за посиланням використовується в тих випадках, коли ви хочете використовувати функцію для вибору змінної, з якою має бути пов'язана дане посилання. При поверненні за посиланням використовуйте такий синтаксис:

```
<? Php
function & find_var ($param)
{
/ * ... код ... * /
return $found_var;
}

$foo = & find_var ($ bar);
$foo-> x = 2;
?>
```

У цьому прикладі встановлюється властивість об'єкта, повернутого функцією find_var, а не його копії, як було б без використання посилань.

Ще один приклад повернення значень користувача функції за посиланням:

```
<? Php
$a= 100;
/* Далі йде функція, яка повертає посилання */
function & s () {
global $a;
// Повертаємо посилання на змінну $a
return $a;
}
// Надаємо посилання змінної $b
$b = & s ();
$b = 0;
echo $a; // Виводить 0
?>
```

Видалення посилань (скидання посилань)

При видаленні посилання, просто розривається зв'язок імені та вмісту змінної. Це не означає, що вміст змінної буде зруйновано. Наприклад:

```
<? Php
$a = 1;
$b = & $a;
unset ($a);
?>
```

Цей код не скине $b, а тільки $a.

І все ж, жорстке посилання - не абсолютно точний синонім об'єкта, на який він посилається. Справа в тому, що оператор Unset (), виконаний для жорсткого посилання, не видаляє об'єкт, на який він посилається, а всього лише розриває зв'язок між посиланням і об'єктом.

Отже, жорстке посилання і змінна (об'єкт), на яку він посилається, абсолютно рівноправні, але зміна однієї тягне зміну іншої. Оператор Unset () розриває зв'язок між об'єктом і посиланням, але об'єкт видаляється тільки тоді, коли на нього ніхто вже не посилається.

Розділ 3. Розробка Web-сторінок за допомогою мови PHP

3.1. Основи клієнт-серверних технологій

PHP - це скриптова мова, оброблювана сервером. Якщо мова йде про сервер, мимоволі спливає в пам'яті поняття клієнта. Все тому, що ці два поняття нерозривно пов'язані. Об'єднує їх комп'ютерна архітектура клієнт-сервер. Зазвичай, коли говорять «сервер», мають на увазі сервер в архітектурі клієнт-сервер, а коли говорять «клієнт» - мають на увазі клієнт в цій же архітектурі. Так що ж це за архітектура? Суть її в тому, щоб розділити функції між двома підсистемами: клієнтом, який відправляє запит на виконання будь-яких дій, і сервером, який виконує цей запит. Взаємодія між клієнтом і сервером відбувається за допомогою стандартних спеціальних протоколів, таких як TCP / IP. Насправді протоколів дуже багато, вони розрізняються за рівнями. Ми розглянемо тільки протокол прикладного рівня HTTP (трохи пізніше), оскільки для вирішення наших програмістських завдань потрібен тільки він. А поки повернемося до клієнт-серверної архітектури і розберемося, що ж таке клієнт і що таке сервер.

Сервер являє собою набір програм, які контролюють виконання різних процесів. Відповідно, цей набір програм встановлений на якомусь комп'ютері. Часто комп'ютер, на якому встановлено сервер, і називають сервером. Основна функція комп'ютера-сервера - по запиту клієнта запустити який-небудь певний процес і відправити клієнту результати його роботи.

Клієнтом називають будь-який процес, який користується послугами сервера. Клієнтом може бути як користувач, так і програма. Основне завдання клієнта - виконання програми та здійснення зв'язку з сервером, коли цього потребує програма. Тобто клієнт повинен надавати користувачеві інтерфейс для роботи з додатком, реалізовувати логіку його роботи і при необхідності відправляти завдання серверу.

Взаємодія між клієнтом і сервером починається з ініціативи клієнта. Клієнт запитує вид обслуговування, встановлює сеанс, отримує потрібні йому результати і повідомляє про закінчення роботи.

Послугами одного сервера найчастіше користується декілька клієнтів одночасно. Тому кожен сервер повинен мати досить велику продуктивність і забезпечувати безпеку даних.

Логічно встановлювати сервер на комп'ютері, що входить в яку-небудь мережу, локальну або глобальну. Однак можна встановлювати сервер і на окремому комп'ютері (тоді він буде одночасно і клієнтом і сервером).

Існує безліч типів серверів. Ось лише деякі з них.

- **Відеосервер**

Такий сервер спеціально пристосований до обробки зображень, зберігання відеоматеріалів, відеоігор і т.п. У зв'язку з цим комп'ютер, на якому

встановлений відеосервер, повинен мати високу продуктивність і велику пам'ять.

- **Пошуковий сервер** призначений для пошуку інформації в Internet.
- **Поштовий сервер** надає послуги у відповідь на запити, надіслані електронною поштою.
- **Сервер WWW** призначений для роботи в Internet.
- **Сервер баз даних** виконує обробку запитів до баз даних.
- **Сервер захисту даних** призначений для забезпечення безпеки даних (містить, наприклад, засоби для ідентифікації паролів).
- **Сервер додатків** призначений для виконання прикладних процесів. З одного боку взаємодіє з клієнтами, одержуючи завдання, а з іншого - працює з базами даних, підбираючи необхідні для обробки дані.
- **Сервер віддаленого доступу** забезпечує колективний віддалений доступ до даних.
- **Файловий сервер** забезпечує функціонування розподілених ресурсів, надає послуги пошуку, зберігання, архівування даних і можливість одночасного доступу до них декількох користувачів.

Зазвичай на комп'ютері-сервері працює відразу кілька програм-серверів. Одна займається електронною поштою, інша розподілом файлів, третя rjynhjk.' web-сторінки.

З усіх типів серверів нас в основному цікавить сервер WWW. Часто його називають web-сервером, http-сервером або навіть просто сервером. Що являє собою web-сервер? По-перше, це сховище інформаційних ресурсів. По-друге, ці ресурси зберігаються і надаються користувачам відповідно до стандартів Internet (такими, як протокол передачі даних HTTP). Як надаються дані у відповідності з цим протоколом, ми розглянемо трохи пізніше. Робота з документами web-сервера здійснюється за допомогою браузера (наприклад, IE, Opera або Mozilla), який відсилає серверу запити, створені відповідно до протоколу HTTP. У процесі виконання завдання сервер може зв'язуватися з іншими серверами.

В якості прикладів web-серверів можна навести сервер Apache групи Apache, Internet Information Server (IIS) компанії Microsoft, SunOne фірми Sun Microsystems, WebLogic фірми BEA Systems, IAS (Inprise Application Server) фірми Borland, WebSphere фірми IBM, OAS (Oracle Application Server).

Все, що ми коли-небудь будемо говорити о web-серверах, орієнтоване на Apache, якщо не вказано інший. А тепер, як було обіцяно, звернемося до протоколу HTTP.

3.2. Протокол HTTP і способи передачі даних на сервер

Internet побудований за багаторівневим принципом, від фізичного рівня, пов'язаного з фізичними аспектами передачі двійкової інформації, і до прикладного рівня, що забезпечує інтерфейс між користувачем і мережею.

HTTP (HyperText Transfer Protocol, протокол передачі гіпертексту) - це протокол прикладного рівня, розроблений для обміну гіпертекстової інформацією в Internet.

HTTP надає набір методів для вказівки цілей запиту, що відправляється серверу. Ці методи основані на узгодженості посилань, де для вказівки ресурсу, до якого має бути застосований даний метод, використовується універсальний ідентифікатор ресурсів (Universal Resource Identifier) у вигляді місцезнаходження ресурсу (Universal Resource Locator, URL) або у вигляді його універсального імені (Universal Resource Name , URN).

Повідомлення по мережі при використанні протоколу HTTP передаються у форматі, схожому з форматом поштового повідомлення Internet (RFC-822) або з форматом повідомлень MIME (Multipurpose Internet Mail Exchange).

HTTP використовується для комунікацій між різними користувацькими програмами та програмами-шлюзами, що надають доступ до існуючих Internet-протоколів, таких як SMTP (протокол електронної пошти), NNTP (протокол передачі новин), FTP (протокол передачі файлів), Gopher і WAIS. HTTP розроблений для того, щоб дозволяти таким шлюзам через проміжні програми-сервери (proxy) передавати дані без втрат.

Протокол реалізує принцип запит / відповідь. Запитуюча програма-клієнт ініціює взаємодію з відповідною програмою-сервером, і надсилає запит, який містить:

- метод доступу;
- адресу URI;
- версію протоколу;
- повідомлення (схоже за формою на MIME) з інформацією про тип переданих даних, інформацією про клієнта, що послав запит, і, можливо, із змістовною частиною (тілом) повідомлення.

Відповідь сервера містить:

- рядок стану, в яку входить версія протоколу і код повернення (успіх або помилка);
- повідомлення (у формі, схожій на MIME), до якого входить інформація сервера, метаінформація (тобто інформація про зміст повідомлення) і тіло повідомлення.

У протоколі не вказується, хто повинен відкривати і закривати з'єднання між клієнтом і сервером. На практиці з'єднання, як правило, відкриває клієнт, а сервер після відправки відповіді ініціює його розриває.

Форма запиту клієнта

Клієнт відсилає серверу запит в одній з двох форм: у повній або скороченій. Запит у першій формі називається відповідно повним запитом, а в другій формі - простим запитом.

Простий запит містить метод доступу та адресу ресурсу. Формально це можна записати так:

<Простий-Запит>: = <Метод> <символ пробіл> <Запитуваний-URI> <символ нового рядка>

В якості методу можуть бути вказані GET, POST, HEAD, PUT, DELETE та інші. Про найбільш поширені з них ми поговоримо трохи пізніше. В якості запитуваної URI найчастіше використовується URL-адреса ресурсу.

Приклад простого запиту:
GET http://phpbook.info/

Тут **GET** - це метод доступу, тобто метод, який повинен бути застосований до запитуваного ресурсу, а http://phpbook.info/ - це URL-адреса запитуваного ресурсу.

Повний запит містить рядок стану, кілька заголовків (заголовок запиту, загальний заголовок або заголовок запису) і, можливо, тіло запиту. Формально загальний вигляд повного запиту можна записати так:

<Повний запит>: = <Рядок Стану> (<Загальний заголовок> | <Тема запиту> | <Заголовок змісту>) <Символ нового рядка> [<Зміст запиту>]

Квадратні дужки тут позначають необов'язкові елементи заголовка, через вертикальну риску перераховані альтернативні варіанти. Елемент <Рядок стану> містить метод запиту та URI ресурсу (як і простий запит) і, крім того, використовувану версію протоколу HTTP. Наприклад, для виклику зовнішньої програми можна задіяти наступний рядок стану:

POST http://phpbook.info/cgi-bin/test HTTP/1.0

У даному випадку використовується метод POST і протокол HTTP версії 1.0.

В обох формах запиту важливе місце займає URI запитуваного ресурсу. Найчастіше URI використовується у вигляді URL-адреси ресурсу. При зверненні до сервера можна застосовувати як повну форму URL, так і спрощену.

Повна форма містить тип протоколу доступу, адресу сервера ресурсу та адресу ресурсу на сервері (рис 1).

У скороченій формі опускають протокол і адресу сервера, вказуючи лише місце розташування ресурсу від кореня сервера. Повну форму використовують, якщо можливе пересилання запиту іншого сервера. Якщо ж робота відбувається тільки з одним сервером, то частіше застосовують скорочену форму.

Рис. 1. Повна форма URL

Методи

Метод повідомляє про мету запиту клієнта. Протокол HTTP підтримує досить багато методів, але реально використовуються тільки три: POST, GET і HEAD. Метод GET дозволяє отримати будь-які дані, ідентифіковані за допомогою URI в запиті ресурсу. Якщо URI вказує на програму, то повертається результат роботи програми, а не її текст (якщо, звичайно, текст не є результат її роботи). Додаткова інформація, необхідна для обробки запиту, вбудовується в сам запит (у рядок статусу). При використанні методу GET у поле тіла ресурсу

повертається власне затребувана інформація (текст HTML-документа, наприклад).

Існує різновид методу GET - **умовний GET.** Цей метод повідомляє серверу про те, що на запит потрібно відповісти, тільки якщо виконуються умови, що міститься в полі if-Modified-Since заголовка запиту. Якщо говорити більш точно, то тіло ресурсу передається у відповідь на запит, якщо цей ресурс змінювався після дати, зазначеної в if-Modified-Since.

Метод **HEAD** аналогічний методу GET, тільки не повертає тіло ресурсу і не має умовного аналога. Метод HEAD використовують для отримання інформації про ресурс. Це може стати в нагоді, наприклад, при вирішенні завдання тестування гіпертекстових посилань.

Метод **POST** розроблений для передачі на сервер такої інформації, як анотації ресурсів, новини і поштові повідомлення, дані для додавання в базу даних, тобто для передачі інформації великого обсягу і досить важливої. На відміну від методів GET і HEAD, в POST передається тіло ресурсу, яке і є інформацією, одержану з полів форм або інших джерел введення.

До цих пір ми тільки теоретизували, знайомилися з основними поняттями. Тепер варто навчитися використовувати все це на практиці. Далі в лекції ми розглянемо, як посилати запити серверу і як обробляти його відповіді.

3.3. Використання HTML-форм для передачі даних на сервер

Як передавати дані серверу? Для цього в мові HTML є спеціальна конструкція - **форми.** Форми призначені для того, щоб отримувати від користувача інформацію. Наприклад, вам потрібно знати логін і пароль користувача для того, щоб визначити, на які сторінки сайту його можна допускати. Або вам необхідні особисті дані користувача, щоб була можливість з ним зв'язатися. Форми якраз і застосовуються для введення такої інформації. У них можна вводити текст або вибирати підходящі варіанти зі списку. Дані, записані у форму, відправляються для обробки спеціальною програмою (наприклад, скриптом на PHP) на сервері. Залежно від введених користувачем даних ця програма може формувати різні web-сторінки, відправляти запити до бази даних, запускати різні додатки і т.п.

Розберемося з синтаксисом HTML-форм. Можливо, багато хто з ним знайомий, але ми все ж повторимо основні моменти, оскільки це важливо.

Отже, для створення форми мовою HTML використовується тег FORM. Всередині нього знаходиться одна або декілька команд INPUT. За допомогою атрибутів action і method тега FORM задаються ім'я програми, яка буде обробляти дані форми, і метод запиту, відповідно. Команда INPUT визначає тип і різні характеристики запитуваної інформації. Надсилання даних форми відбувається після натискання кнопки input типу submit. Створимо форму для реєстрації учасників заочної школи програмування.

<h2> Форма для реєстрації учасників </ h2>
<form action="1.php" method=POST>

При відправленні запиту буде використаний метод POST ->
Ім'я
 <input type = text name = "first_name"
value = "Ваше ім'я">
Прізвище
 <input type=text name="last_name">
E-mail
 <input type=text name="email">
<p>
Виберіть курс, який ви б хотіли відвідувати:
<input type=radio name="kurs" value="PHP"> PHP
<input type=radio name="kurs" value="Lisp"> Lisp
<input type=radio name="kurs" value="Perl"> Perl
<input type=radio name="kurs" value="Unix"> Unix
<P> Що ви хочете, щоб ми знали про вас?
<textarea name="comment" cols=32 rows=5>
<P> <Input name = "confirm" type = checkbox
checked> Підтвердити отримання
<input type=submit value="Відправити">
<input type=reset value="Відмініти">

Після обробки браузером цей файл буде виглядати приблизно так:

Рис. 1. Приклад html-форми

Ось так створюються і виглядають HTML-форми. Будемо вважати, що ми навчилися чи згадали, як їх створювати. Як ми бачимо, у формі можна вказувати метод передачі даних. Подивимося, що буде відбуватися, якщо вказати метод GET або POST, і в чому буде різниця.

Для методу GET

Для методу GET При відправці даних форми за допомогою методу GET вміст форми додається до URL після знака запитання у вигляді пар імен = значення, об'єднаних за допомогою амперсанда &:

action?name1=value1=name2=value2&name3=value3

Тут action - це URL-адреса програми, яка повинна обробляти форму (це або програма, задана в атрибуті action тега form, або сама поточна програма, якщо цей атрибут опущений). Імена name1, name2, name3 відповідають іменам елементів форми, а value1, value2, value3 - значення цих елементів. Всі спеціальні

символи, включаючи = і &, в іменах або значеннях цих параметрів будуть закодовані. Тому не варто використовувати в назвах або значеннях елементів форми ці символи і символи кирилиці в ідентифікаторах.

Якщо в полі для введення ввести який-небудь службовий символ, то він буде переданий в його шістнадцятковому коді, наприклад, символ $ заміниться на% 24. Так само передаються і кирилиця.

Для полів введення тексту і пароля (це елементи input з атрибутом type = text і type = password), значенням буде те, що введе користувач. Якщо користувач нічого не вводить в таке поле, то в рядку запиту буде присутній елемент name =, де name відповідає імені цього елемента форми.

Для кнопок типу **checkbox** і **radio button** значення **value** визначається атрибутом VALUE в тому випадку, коли кнопка відзначена. Не зазначені кнопки при складанні рядка запиту ігноруються повністю. Кілька кнопок типу checkbox можуть мати один атрибут NAME (і різні VALUE), якщо це необхідно. Кнопки типу radio button призначені для одного з усіх запропонованих варіантів і тому повинні мати однаковий атрибут NAME і різні атрибути VALUE.

У принципі створювати HTML-форму для передачі даних методом GET не обов'язково. Можна просто додати рядок URL потрібні змінні та їх значення.

http://phpbook.info/test.php?id=10&user=pit

У зв'язку з цим у передачі даних методом GET є один істотний недолік - будь-хто може підробити значення параметрів. Тому не радимо використовувати цей метод для доступу до захищених паролем сторінок, для передачі інформації, що впливає на безпеку роботи програми або сервера. Крім того, не варто застосовувати метод GET для передачі інформації, яку не дозволено змінювати користувачеві.

Незважаючи на всі ці недоліки, використовувати метод GET досить зручно при налагодженні скриптів (можна бачити значення й імена переданих змінних) і для передачі параметрів, які не впливають на безпеку.

Для методу POST

Вміст форми кодується точно так само, як для методу **GET**, але замість додавання рядка до **URL**вміст запиту надсилається блоком даних як частина операції POST. Якщо присутній атрибут**ACTION**, то значення **URL**, яке там знаходиться, визначає, куди посилати цей блок даних. Цей метод, як уже зазначалося, рекомендується для передачі великих за обсягом блоків даних.

Інформація, введена користувачем і відправлена серверу за допомогою методу POST, подається на стандартне введення програми, зазначеної в атрибуті action, чи поточного скрипту, якщо цей атрибут опущений. Довжина посилається файлу передається у змінній оточення CONTENT_LENGTH, а тип даних - у змінній CONTENT_TYPE.

Передати дані методом POST можна тільки за допомогою HTML-форми, оскільки дані передаються в тілі запиту, а не в заголовку, як у GET. Відповідно і змінити значення параметрів можна, тільки змінивши значення, введене в форму. При використанні POST користувач не бачить чи передаються серверу дані.

Основна перевага POST запитів - це їхня велика безпека і функціональність у порівнянні з GET-запитами. Тому метод POST частіше використовують для передачі важливої інформації, а також інформації великого обсягу. Тим не менш не варто цілком покладатися на безпеку цього механізму, оскільки дані POST запиту також можна підробити, наприклад створивши html-файл на своїй машині і заповнивши його потрібними даними. Крім того, не всі клієнти можуть застосовувати метод POST, що обмежує варіанти його використання.

При відправці даних на сервер будь-яким методом передаються не тільки самі дані, введені користувачем, але і ряд змінних, які називаються змінними середовища, що характеризують клієнта, історію його роботи, шляхи до файлів і т.п. Ось деякі із змінних оточення:

- REMOTE_ADDR - IP-адреса хоста (комп'ютера), що відправляє запит;
- REMOTE_HOST - ім'я хоста, з якого надіслано запит;
- HTTP_REFERER - адреса сторінки, що посилається на поточний скрипт;
- REQUEST_METHOD - метод, який був використаний при відправці запиту;
- QUERY_STRING - інформація, яка перебуває в URL після знака питання;
- SCRIPT_NAME - віртуальний шлях до програми, яка повинна виконуватися;
- HTTP_USER_AGENT - інформація про браузер, який використовує клієнт

3.4. Обробка запитів за допомогою PHP

До цих пір ми згадували тільки, що запити клієнта обробляються на сервері за допомогою спеціальної програми. Насправді цю програму ми можемо написати самі, в тому числі і на мові PHP, і вона буде робити з отриманими даними все, що ми захочемо. Для того, щоб написати цю програму, необхідно познайомитися з деякими правилами і інструментами, запропонованими для цих цілей PHP.

Усередині PHP-скрипта є декілька способів отримання доступу до даних, переданим клієнтом по протоколу HTTP. До версії PHP 4.1.0 доступ до таких даних здійснювався за іменами переданих змінних (нагадаємо, що дані передаються у вигляді пар «ім'я змінної, символ« = », значення змінної»). Таким чином, якщо, наприклад, було передано first_name = Nina, то всередині скрипта з'являлася змінна $first_name зі значенням Nina. Якщо потрібно розрізняти, яким методом були передані дані, то використовувалися асоціативні масиви $HTTP_POST_VARS і $HTTP_GET_VARS, ключами яких були імена переданих змінних, а значеннями - відповідно значення цих змінних. Таким чином, якщо пара first_name = Nina передана методом GET, то $ HTTP_GET_VARS ["first_name"] = "Nina".

Використовувати в програмі імена переданих змінних безпосередньо небезпечно. Тому було вирішено починаючи з PHP 4.1.0 задіяти для звернення до змінних, переданих за допомогою HTTP-запитів, спеціальний масив - $ _REQUEST. Цей масив містить дані, передані методами POST і GET, а також за допомогою HTTP cookies. Це Суперглобальний асоціативний масив, тобто його значення можна отримати в будь-якому місці програми, використовуючи як ключ ім'я відповідної змінної (елементу форми).

Приклад 2. Припустимо, ми створили форму для реєстрації учасників заочної школи програмування, як у наведеному вище прикладі. Тоді у файлі 1.php, що обробляє цю форму, можна написати наступне:

```
<? Php
$str = "Здрастуйте,
". $_REQUEST [" First_name "]."
". $_REQUEST [" Last_name "]."! <br> ";
$str .= "Ви обрали для вивчення курс по
". $_REQUEST [" Kurs "];
echo $str;
?>
```

Тоді, якщо в формі ми ввели ім'я «Вася», прізвище «Петров» і вибрали серед усіх курсів курс по PHP, на екрані браузера отримаємо таке повідомлення:

Здравствуйте, Вася Петров!
Ви обрали для вивчення курс по PHP

Після введення масиву $_REQUEST масиви $HTTP_POST_VARS і $HTTP_GET_VARS для однорідності були перейменовані в $_POST і $_GET відповідно, але самі вони з вжитку не зникли з міркувань сумісності з попередніми версіями PHP. На відміну від своїх попередників, масиви $_POST і $_GET стали суперглобальними, тобто доступними безпосередньо і всередині функцій і методів.

Наведемо приклад використання цих масивів. Припустимо, нам потрібно обробити форму, що містить елементи введення з іменами first_name, last_name, kurs (наприклад, форму form.html, наведену вище). Дані були передані методом POST, і дані, передані іншими методами, ми обробляти не хочемо. Це можна зробити наступним чином:

```
<? Php
$Str = "Здрастуйте,
". $_POST [" First_name "]."
". $_POST [" Last_name "]."! <br> ";
$Str .= "Ви обрали для вивчення курс по".
$_POST ["Kurs"];
echo $Str;
?>
```

Тоді на екрані браузера, якщо ми ввели ім'я «Вася», прізвище «Петров» і вибрали серед усіх курсів курс по PHP, побачимо повідомлення, як у попередньому прикладі:

Здравствуйте, Вася Петров!
Ви обрали для вивчення курс по PHP

Для того, щоб зберегти можливість обробки скриптів більш ранніх версій, ніж PHP 4.1.0, була введена директива register_globals, що дозволяє чи забороняє доступ до змінних безпосередньо за їхніми іменами. Якщо у файлі налаштувань PHP параметр register_globals = On, то до змінних, переданим сервера методами GET і POST, можна звертатися просто за їхніми іменами (тобто можна писати $first_name). Якщо ж register_globals = Off, то потрібно писати $_REQUEST ["first_name"] або $_POST ["first_name"], $_GET ["first_name"], $HTTP_POST_VARS ["first_name"], $HTTP_GET_VARS ["first_name"]. З точки зору безпеки цю директиву краще відключати (тобто register_globals = Off). При включеній директиві register_globals перераховані вище масиви також будуть містити дані, передані клієнтом.

Іноді виникає необхідність дізнатися значення якої-небудь змінної оточення, наприклад метод, що використовувався при передачі запиту або IP-адреса комп'ютера, що відправив запит. Отримати таку інформацію можна за допомогою функції getenv(). Вона повертає значення змінної оточення, ім'я якої передано їй як параметр.

```
<?
getenv ('REQUEST_METHOD');
// Поверне використаний метод
echo getenv ('REMOTE_ADDR');
// Виведе IP-адресу користувача,
// Послав запит
?>
```

Все, що записано в URL після знака запитання, можна отримати за допомогою команди

getenv ('QUERY_STRING');

Завдяки цьому можна методом GET передавати дані в якому-небудь іншому вигляді. Наприклад, вказувати тільки значення декількох параметрів через знак плюс, а в скрипті розбивати рядок запиту на частини або можна передавати значення лише одного параметра. У цьому випадку в масиві $_GET з'явиться порожній елемент з ключем, рівним цьому значенню (всього рядка запиту), причому символ «+», що зустрівся в рядку запиту, буде замінений на підкреслення «_».

Методом POST дані передаються тільки за допомогою форм, і користувач (клієнт) не бачить, які саме дані відправляються серверу. Щоб їх побачити, хакер повинен підмінити нашу форму на власну. Тоді сервер відправить результати обробки неправильної форми не туди, куди потрібно. Щоб цього уникнути, можна перевіряти адресу сторінки, з якої були надіслані дані. Це можна зробити знову ж за допомогою функції getenv ():

getenv ('HTTP_REFERER');

Приклад обробки запиту за допомогою PHP

Нагадаємо, в чому полягало завдання, і уточнимо його формулювання. Потрібно написати форму для реєстрації учасників заочної школи програмування і після реєстрації відправити учаснику повідомлення. Ми назвали це повідомлення універсальним листом, але воно буде трохи відрізнятися від того листа, який ми склали вище. Тут ми не будемо відправляти що-небудь по електронній пошті, щоб не уподібнюватися спамерам, а просто згенеруємо це повідомлення і виведемо його на екран браузера. Початковий варіант форми реєстрації ми вже наводили вище. Змінимо його таким чином, щоб кожен хто реєструється, міг вибрати скільки завгодно курсів для відвідування, і не будемо підтверджувати отримання реєстраційної форми.

```
<h2> Форма для реєстрації студентів </ h2>
<form action="1.php" method=POST>
Ім'я <br> <input type = text name = "first_name"
value = "Ваше ім'я"> <br>
Прізвище <br> <input type=text name="last_name"> <br>
E-mail <br> <input type=text name="email"> <br>
lt;p> Виберіть курс, який ви б хотіли відвідувати: lt;br>
lt;input type=checkbox name='kurs[]' value='PHP'> PHP <br>
lt;input type=checkbox name='kurs[]' value='Lisp'> Lisp <br>
lt;input type=checkbox name='kurs[]' value='Perl'> Perl <br>
lt;input type=checkbox name='kurs[]' value='Unix'> Unix <br>
lt;P> Що ви хочете, щоб ми знали про вас? <BR>
lt;textarea name="comment" cols=32 rows=5> </ textarea>
lt;input type=submit value="Відправте">
lt;input type=reset value="Відмінити">
</ Form>
```

Тут все досить просто і зрозуміло. Єдине, що можна відзначити, - це спосіб передачі значень елемента checkbox. Коли ми пишемо в імені елемента kurs [], це означає, що перший зазначений елемент checkbox буде записаний в перший елемент масиву kurs, другий зазначений checkbox - у другий елемент масиву і т.д. Можна, звичайно, просто дати різні імена елементів checkbox, але це ускладнить обробку даних, якщо курсів буде багато.

Скрипт, який все це буде розбирати і обробляти, називається 1.php (форма посилається саме на цей файл, що записано в її атрибуті action). За замовчуванням використовується для передачі метод GET, але ми вказали POST. За отриманими даними від зареєстрованої людини, скрипт генерує відповідне повідомлення. Якщо людина вибрала якісь курси, то йому виводиться повідомлення про час їх проведення та про лекторів, які їх читають. Якщо людина нічого не вибрав, то виводиться повідомлення про наступні збори заочної школи програмістів (ЗШП).

```
<?
// Створимо масиви відповідностей курс - час його
// Проведення та викладач курсу
$times = array ("PHP" => "14.30", "Lisp" => "12.00",
```

```
"Perl" => "15.00", "Unix" => "14.00");
$lectors = array ("PHP" => "Василь Васильович",
"Lisp" => "Іван Іванович", "Perl" => "Петро Петрович", "Unix" => "Семен
Семенович");
define ("SIGN", "З повагою, адміністрація");
// Визначаємо підпис листа як константу
define ("MEETING_TIME", "18.00");
// Задаємо час зборів студентів
$date = "12 травня"; // задаємо дату проведення лекцій
// Починаємо складати текст повідомлення
$str = "Здрастуйте, шановний". $_POST ["First_name"]. "". $_POST ["Last_name
"]."!<br>";
$str .= "<br> Повідомляємо Вам, що";
$kurses = $_POST ["kurs"]; // збережемо в цій змінній
// Список вибраних курсів
if (!isset ($kurses)) {// якщо не обраний жоден курс
$event = "наступні збори студентів";
$str .= "$event відбудеться $date". MEETING_TIME. "<br>";
} Else {// якщо хоча б один курс вибраний
$event = "обрані Вами лекції відбудуться $date <ul>";
// Функція count обчислює число елементів у масиві
$lect = "";
for ($i = 0; $i <count ( kurses); $i++) {
// Для кожного обраного курсу
$k = $kurses [$i]; // запам'ятовуємо назву курсу
$lect = $lect. "<li> Лекція з $k в $times [$k]";
// Складаємо повідомлення
$lect .= "(Ваш лектор, $lectors [$k])";
}
$event = $event. $lect. "</ Ul>";
$str .= "$event";
}
$str .= "<br>". SIGN; // додаємо підпис
echo $str; // виводимо повідомлення на екран
?>
```

3.5. Суперглобальні масиви

Розглянемо роботу суперглобальних змінних в PHP.

У PHP існує кілька суперглобальний змінних, а точніше суперглобальних масивів:

- $_SERVER
- $_GET
- $_POST
- $_FILES

- $_COOKIE
- $_SESSION
- $_REQUEST
- $_ENV

Суперглобальний масив $_SERVER

Масив являє собою інформацію про заголовки, шляхи та розміщення скриптів. Записи в цьому масиві створюються веб-сервером. Не існує гарантій, що веб-сервер сформує цей масив з усіма параметрами. Даний масив містить такі елементи:

PHP_SELF: ім'я файлу, що в даний час виконується PHP-скриптом. Наприклад при виконанні скрипта http://phpprogs.ru/test/guestbook2/ даний елемент буде приймати значення / test/guestbook2/index.php.

argv: список аргументів, переданих скрипту. Якщо використовує в командному рядку, то отримуєте масив значень, якщо використовується $_GET, то буде містити рядок запиту.

argc: містить число параметрів переданих сценарієм (якщо запуск був з командного рядка).

GATEWAY_INTERFACE: параметр повертає версію CGI, яку використовує веб-сервер.

SERVER_ADDR: елемент містить IP адресу сервера, де виконується скрипт.

SERVER_NAME: елемент містить ім'я веб-сервера, де виконується скрипт.

SERVER_SOFTWARE: ідентифікаційний рядок веб-сервера, який повертається у відповідь при запитах.

SERVER_PROTOCOL: ім'я та версія протоколу HTTP.

REQUEST_METHOD: використовуваний метод запиту до веб-сервера (POST, GET, HEAD, PUT).

REQUEST_TIME: відмітка про час початку запиту (починаючи з PHP 5.1.0).

QUERY_STRING: рядок запиту до веб-сторінки, якщо вона існує, за допомогою якого був здійснений доступ до сторінки

DOCUMENT_ROOT: коренева директорія, з якої виконується скрипт.

HTTP_ACCEPT: зміст заголовка ACCEPT, якщо він є.

HTTP_ACCEPT_CHARSET: зміст заголовка ACCEPT-CHARSET, якщо він є. Наприклад 'iso-8859-1, *, utf-8'.

HTTP_ACCEPT_ENCODING: зміст заголовка ACCEPT-ENCODING, якщо він є. Наприклад 'gzip'.

HTTP_ACCEPT_LANGUAGE: зміст заголовка ACCEPT-LANGUAGE, якщо він є. Наприклад 'en'.

HTTP_ACCEPT_CONNECTION: зміст заголовка ACCEPT-CONNECTION, якщо він є. Наприклад 'Keep-Alive'.

HTTP_HOST: зміст заголовка HOST, тобто він є.

HTTP_REFERER: адреса сторінки, з якої на поточну сторінку перейшло програмне забезпечення користувача. Не всі ПЗ користувача передають цей параметр, а деякиі ПЗ навіть змінюють його. Отже, даному параметру довіряти не можна.

HTTP_USER_AGENT: цей параметр містить інформацію про клієнт користувача (ПО користувача), який звертається до сторінки. Наприклад 'Mozilla/4.5 [RU] (X11; U; Linux 2.2.9 i586). Також цю інформацію Ви можете отримати з функції get_browser ().

HTTPS: параметр містить інформацію, якщо запит був зроблений через HTTPS.

REMOTE_ADDR: IP-адреса користувача, з якого він переглядає сторінку.

REMOTE_HOST: ім'я хоста користувача, з якого він переглядає цю сторінку.

REMOTE_POST: порт, який використовується для з'єднання з веб-сервером.

SCRIPT_FILENAME: абсолютний шлях до поточного скрипта.

SERVER_ADMIN: значення SERVER_ADMIN, взяте з конфігураційного файлу Apache.

SERVER_PORT: порт веб-сервера, використаний для передачі даних по HTTP. За замовчуванням 80.

SERVER_SIGNATURE: рядок, що містить версію веб-сервера і ім'я віртуального хоста.

PATH_TRANSLATED: базовий шлях до поточного сценарія.

SCRIPT_NAME: містить шлях та ім'я поточного скрипта.

REQUEST_URI: URI для поточної сторінки.

PHP_AUTH_DIGEST: якщо PHP працює як модуль Apache, то параметр використовується як реквізити по протоколу HTTP для перевірки автентичності.

PHP_AUTH_USER: якщо PHP працює як модуль Apache або IIS, то параметр містить ім'я користувача при аутентифікації по протоколу HTTP.

PHP_AUTH_PW: якщо PHP працює як модуль Apache або IIS, то параметр містить пароль користувача при аутентифікації по протоколу HTTP.

AUTH_TYPE: якщо PHP працює як модуль Apache або IIS, то параметр містить тип аутентифікації по протоколу HTTP.

Суперглобальний масив $_GET

Масив $_GET представляє собою асоціативний масив елементів, переданих за допомогою HTTP GET запитів поточному PHP-скрипту. Немає необхідності оголошувати масив $_GET всередині функції користувача командою "global $_GET;", тому що даний масив є суперглобальний.

Суперглобальний масив $_POST

Масив $_POST представляє собою асоціативний масив елементів, переданих за допомогою HTTP POST запитів поточному PHP-скрипту. Немає

необхідності оголошувати масив $_POST всередині функції користувача командою "global $_POST;", тому що даний масив є суперглобальний.

Суперглобальний масив $_FILES

Масив $_FILES представляє собою асоціативний масив елементів, переданих за допомогою HTTP POST запитів поточному PHP-скрипту. Немає необхідності оголошувати масив $_FILES всередині функції користувача командою "global $_FILES;", тому що даний масив є суперглобальний.

Суперглобальний масив $_COOKIE

Масив $_COOKIE представляє собою асоціативний масив елементів, переданих за допомогою HTTP COOKIE запитів поточному PHP-скрипту. Немає необхідності оголошувати масив $_COOKIE всередині функції користувача командою "global $_COOKIE;", тому що даний масив є суперглобальний.

Суперглобальний масив $_SESSION

Даний асоціативний масив містить змінні сесії, доступні для даного скрипта. Немає необхідності оголошувати масив $_SESSION всередині функції користувача командою "global $_SESSION;", тому що даний масив є суперглобальний.

Суперглобальний масив $ _REQUEST

Масив $_REQUEST є об'єднаним асоціативним масивом, який включає в себе масиви $_GET, $_POST, $_FILES. Немає необхідності оголошувати масив $_REQUEST всередині функції користувача командою "global $_REQUEST;", тому що даний масив є суперглобальний.

Суперглобальний масив $ _ENV

$_ENV Представляє собою асоціативний масив, який містить значення змінних з середовища, в якій працює інтерпретатор PHP. Немає необхідності оголошувати масив $_ENV всередині функції користувача командою "global $_ENV;", тому що даний масив є суперглобальний.

PHP і Cookies

Cookies - це механізм зберігання даних броузером віддаленого комп'ютера для ідентифікації відвідувачів і зберігання параметрів веб-сторінок (наприклад, змінних).

Наведемо приклад використання Cookies на конкретному прикладі.

Припустимо, нам потрібно написати лічильник відвідування сайту. Нам потрібно знати, яке число відвідувань сайту здійснювалося кожним конкретним користувачем.

Дану задачу можна вирішити двома способами. Перший з них полягає у веденні обліку IP-адрес користувачів. Для цього потрібна база даних всього з однієї таблиці, приблизна структура якої така:

IP-адреса	Число відвідувань
210.124.134.203	7
212.201.78.207	14
83.103.203.73	3

Коли користувач заходить на сайт, нам потрібно визначити його IP-адресу, знайти в базі даних інформацію про його відвідини, збільшити лічильник і вивести його в браузер відвідувача. Написати обробник (скрипт) подібної процедури нескладно. Проте при використанні такого методу у нас з'являються проблеми наступного характеру:

- Для кожної IP-адреси потрібно вести облік в одній таблиці, яка може бути дуже великою. А з цього випливає, що ми нераціонально використовуємо процесорний час і дисковий простір;
- У більшості домашніх користувачів IP-адреси є динамічними. Тобто, сьогодні у нього адреса 212.218.78.124, а завтра - 212.218.78.137. Таким чином, велика вірогідність ідентифікувати одного користувача кілька разів.

Можна використовувати другий спосіб, який набагато легший в реалізації і є більш ефективним. Ми встановлюємо в Cookie змінну, яка буде зберігатися на диску віддаленого користувача. Ця змінна буде зберігати інформацію про відвідування. Вона буде зчитуватися скриптом при зверненні користувача до сервера. Вигода такого методу ідентифікації очевидна. По-перше, нам не потрібно зберігати безліч непотрібної інформації про IP-адреси. По-друге, нас не цікавлять динамічні IP-адреси, оскільки дані про відвідання зберігаються конкретно у кожного відвідувача сайту.

Тепер зрозуміло, для чого ми можемо використовувати Cookie - для зберігання невеликої за обсягом інформації у клієнта (відвідувача) сайту, наприклад: налаштування сайту (колір фону сторінок, мова, оформлення таблиць і.т.д.), а також іншої інформації.

Файли Cookies представляють собою звичайні текстові файли, які зберігаються на диску у відвідувачів сайтів. Файли Cookies і містять ту інформацію, яка була в них записана сервером.

Програмування Cookies

Приступимо до програмування Cookies.

Для установки Cookies використовується функція SetCookie (). Для цієї функції можна вказати шість параметрів, один з яких є обов'язковим:

- name - задає ім'я (рядків), закріплене за Cookie;
- value - визначає значення змінної (рядок);
- expire - час "життя" змінної (ціле число). Якщо цей параметр не вказувати, то Cookie будуть "жити" до кінця сесії, тобто до закриття браузера. Якщо час вказано, то, коли він настане, Cookie самознищиться.
- path - шлях до Cookie (рядок);
- domain - домен (рядок). Як значення встановлюється ім'я хоста, з якого Cookie був записаний;
- secure - передача Cookie через захищене HTTPS-з'єднання.

Зазвичай використовуються тільки три перші параметра.

Приклад установки Cookies:

```
<? Php

// Встановлюємо Cookie до кінця сесії:
```

```
SetCookie ("Test", "Value");

// Встановлюємо Cookie на одну годину після установки:
SetCookie ("My_Cookie", "Value", time () +3600);

?>
```

При використанні Cookies необхідно мати на увазі, що Cookies повинні встановлюватися до першого виводу інформації в браузер (наприклад, оператором echo або якої-небудь іншої функції). Тому бажано встановлювати Cookies на самому початку скрипта. Cookies встановлюються за допомогою певного заголовка сервера, а якщо скрипт виводить що-небудь, то це означає, що починається тіло документа. У результаті Cookies не будуть встановлені і може бути виведено попередження. Для перевірки успішності установки Cookies можна використовувати такий метод:

```
<? Php
// Встановлюємо Cookie до кінця сесії:
// В разі успішного встановлення Cookie, функція SetCookie повертає TRUE:
if (SetCookie ("Test", "Value")) echo "<h3> Cookies успішно встановлені! </
h3>";
?>
```

Функція SetCookie () повертає TRUE у випадку успішної установки Cookie. У випадку, якщо Cookie встановити не вдається SetCookie () поверне FALSE і можливо, попередження (залежить від налаштувань PHP). Приклад невдалого встановлення Cookie:

```
<? Php
// Cookies встановити не вдасться, оскільки перед відправкою
// Заголовка Cookie ми виводимо в браузер рядок 'Hello':
echo "Hello";
// Функція SetCookie поверне FALSE:
if (SetCookie ("Test", "Value")) echo "<h3> Cookie успішно встановлено! </
h3>";
else echo "<h3> Cookie встановити не вдалося! </ h3>";
// Виводить 'Cookie встановити не вдалося!'.
?>
```

Cookie встановити не вдалося, оскільки перед посилкою заголовка Cookie ми вивели в браузер рядок "Hello".

Читання значень Cookies

Отримати доступ до Cookies та їх значенням досить просто. Вони зберігаються в суперглобальних масивах і $_COOKIE і $HTTP_COOKIE_VARS.

Доступ до значень здійснюється за ім'ям встановлених Cookies, наприклад:

```
echo $_COOKIE ['my_cookie'];
// Виводить значення встановленої Cookie 'My_Cookie'
```

Приклад встановлення Cookie і подальшого його читання:

```
<? Php
// Встановлюємо Cookie 'test' зі значенням 'Hello' на одну годину:
setcookie ("test", "Hello", time () +3600);
// При наступному запиті скрипта виводить 'Hello':
echo @$ _COOKIE ['test']; ?>
```

У розглянутому прикладі при першому зверненні до скрипта встановлюється Cookie "test" із значенням "hello". При повторному зверненні до скрипта буде виведено значення Cookie "test", тобто рядок "Hello".

При читанні значень Cookies звертайте увагу на перевірку існування Cookies, наприклад, використовуючи оператор isset (). Або шляхом заборони виводу помилок оператором @

А ось приклад, як побудувати лічильник числа завантажень сторінки за допомогою Cookies:

```
<? Php
// Перевіряємо, чи був вже встановлений Cookie 'Mortal',
// Якщо так, то читаємо його значення,
// І збільшуємо значення лічильника звернень до сторінки:
if (isset ($ _COOKIE ['Mortal'])) $cnt = $_COOKIE ['Mortal'] +1;
else $ cnt = 0;
// Встановлюємо Cookie 'Mortal' із значенням лічильника,
// З часом "життя" до 18/07/29,
// Тобто на дуже довгий час:
setcookie ("Mortal", $ cnt, 0x6FFFFFFF);
// Виводить число відвідувань (завантажень) цієї сторінки:
echo "<p> Ви відвідували цю сторінку <b> ".@$_ COOKIE ['Mortal']."</ b> раз
</ p>";
?>
```

Видалення Cookies

Іноді виникає необхідність видалення Cookies. Зробити це нескладно, необхідно лише знову встановити Cookie з ідентичним ім'ям і порожнім параметром. Наприклад:

```
<? Php
// Видаляємо Cookie 'Test':
SetCookie ("Test ","");
?>
```

Створення масиву Cookies і його читання

Ми можемо створити масив Cookies, а потім прочитати масив Cookies та значення цього масиву:

```
<? Php
// Передаємо масив Cookies:
setcookie ("cookie [1]", "Перший");
setcookie ("cookie [2]", "Другий");
setcookie ("cookie [3]", "Третій");
```

```
// Після перезавантаження сторінки ми відобразимо
// Вміст масиву Cookies 'cookie':
if (isset ($_COOKIE ['cookie'])) {
foreach ($_COOKIE ['cookie'] as $name => $value) {
echo "$ name: $ value <br>";
}
}
?>
```

Переваги використання Cookies незаперечні. Проте існують і деякі проблеми їх використання. Перша з них полягає в тому, що відвідувач може блокувати Cookies браузером або просто видалити всі Cookies або їх частину. Таким чином, ми можемо мати деякі проблеми в ідентифікації таких відвідувачів.

Робота з сесією

Етапи роботи з сесіями

1. Установка імені сесії (не обов'язково).
2. Створення сесії.
3. Реєстрація змінних сесії і їх використання.
4. Видалення змінних сесії.
5. Знищення сесії.

Створення сесії

Перше, що потрібно зробити для роботи з сесіями (якщо вони вже налаштовані адміністратором сервера), це запустити механізм сесій. Якщо в налаштуваннях сервера змінна session.auto_start встановлена в значення "0" (якщо session.auto_start = 1, то сесії запускаються автоматично), то будь-який скрипт, в якому потрібно використовувати дані сесії, повинен починатися з команди

session_start ();

Отримавши таку команду, сервер створює нову сесію або відновлює поточну, ґрунтуючись на ідентифікаторі сесії, переданому запитом. Як це робиться? Інтерпретатор PHP шукає змінну, в якій зберігається ідентифікатор сесії (за умовчанням це PHPSESSID) спочатку в cookies, потім в змінних, переданих за допомогою POST-і GET-запитів. Якщо ідентифікатор знайдений, то користувач вважається ідентифікованим, проводиться заміна всіх URL і виставлення cookies. В іншому випадку користувач вважається новим, для нього генерується новий унікальний ідентифікатор, потім проводиться заміна URL і виставлення cookies.

Команду session_start () потрібно викликати у всіх скриптах, у яких доведеться використовувати змінні сесії, причому до виведення яких-небудь даних в браузер. Це пов'язано з тим, що cookies виставляються тільки до виведення інформації на екран.

Отримати ідентифікатор поточної сесії можна за допомогою функції session_id ().

Для наочності сесії можна задати ім'я за допомогою функції session_name ([імя_сессії]). Робити це потрібно ще до ініціалізації сесії. Отримати ім'я поточної сесії можна за допомогою цієї ж функції, викликаної без параметрів: session_name ();

Ім'я сесії - це ім'я параметра, в якому зберігається ідентифікатор сесії.

Приклад створення сесії;

```
<?
session_start ();
// Створюємо нову сесію або
// Відновлюємо поточну
echo session_id ();
// Виводимо ідентифікатор сесії
?>
<html>
<head> <title> My home page </ title> </ head>
... // Домашня сторінка
</ Html>
<?
echo session_name ();
// Виводимо ім'я поточної сесії.
// У даному випадку це PHPSESSID
?>
```

Якщо ж створити файл authorize.php, то значення змінних, що виводяться (id сесії та її ім'я) будуть такими ж, якщо перейти на нього з index.php і не закривати перед цим вікно браузера, тоді ідентифікатор сесії зміниться.

Реєстрація змінних сесії

Однак від самого ідентифікатора та імені сесії нам користі для вирішення наших завдань небагато. Ми ж хочемо передавати і зберігати протягом сесії наші власні змінні (наприклад, логін і пароль). Для того щоб цього досягти, потрібно просто зареєструвати свої змінні:

session_register (ім’я_змінної1, ім’я_змінної2, ...);

Увага! session_register є застарілою і у версії PHP 6.0 взагалі вилучена. Її використання вкрай не бажано. Замість session_register слід використовувати $_SESSION.

Зауважимо, що реєструються не значення, а імена змінних. Зареєструвати змінну достатньо один раз на будь-якій сторінці, де використовуються сесії. Імена змінних передаються функції session_register () без знаку $. Усі зареєстровані таким чином змінні стають глобальними (тобто доступними з будь-якої сторінки) протягом даної сесії роботи з сайтом.

Зареєструвати змінну також можна, просто записавши її значення в асоціативний масив $ _SESSION, тобто написавши

$_SESSION ['Ім'я_змінної'] = 'Значення_змінної';

У цьому масиві зберігаються всі зареєстровані (тобто глобальні) змінні сесії.

Доступ до таких змінних здійснюється за допомогою масиву $_SESSION ['ім'я_змінної'] (або $HTTP_SESSION_VARS ['ім'я_змінної'] для версії PHP 4.0.6 і більш ранніх). Якщо ж в налаштуваннях php включена опція register_globals, то до сесійних змінних можна звертатися ще й як до звичайних змінних, наприклад так: $ім'я_змінної.

Якщо register_globals = off (відключені), то користуватися session_register () для реєстрації змінних переданих методами POST або GET, не можна, тобто це просто не працює. І взагалі, не рекомендується одночасно використовувати обидва методи реєстрації змінних, $_SESSION і session_register ().

Приклад реєстрація змінних

Зареєструємо логін і пароль, які вводяться користувачем на сторінці авторизації.

```
<?
session_start ();
// Створюємо нову сесію або
// Відновлюємо поточну
if (! isset ($_GET ['go'])){
echo "<form>
Login: <input type=text name=login>
Password: <input type = password
name = passwd>
<input type=submit name=go value=Go>
</ Form> ";
} Else {
$_SESSION ['Login']=$_ GET [' login '];
// Реєструємо змінну login
$_SESSION ['Passwd']=$_ GET [' passwd '];
// Реєструємо змінну passwd
// Тепер логін і пароль - глобальні
// Змінні для цієї сесії
if ($_GET ['login']==" pit "&&
$_GET ['Passwd']==" 123 ") {
Header ("Location: secret_info.php");
// Перенаправляємо на сторінку
// Secret_info.php
} Else echo "Невірний введення,
спробуйте ще раз <br> ";
}
print_r ($_SESSION);
// Виводимо всі змінні сесії
?>
```

Вміст файлу authorize.php

Тепер, потрапивши на сторінку secret_info.php, та й на будь-яку іншу сторінку сайту, ми зможемо працювати з введеними користувачем логіном і паролем, які будуть зберігатися в масиві $ _SESSION. Таким чином, якщо змінити код секретної сторінки (зауважте, ми перейменували її в secret_info.php) так:

```
<? Php
session_start ();
// Створюємо нову сесію або
// Відновлюємо поточну
print_r ($ _SESSION);
// Виводимо всі змінні сесії
?>
<html>
<head> <title> Secret info </ title> </ head>
<body>
<p> Тут я хочу ділитися секретами
з одним Петром.
</ Body>
</ Html>
```

Вміст файлу secret_info.php

То ми отримаємо в браузері на секретній сторінці наступне:

```
Array ([login] => pit [passwd] => 123)
Тут я хочу ділитися секретами
з одним Петром.
```

У результаті отримаємо список змінних, зареєстрованих на authorize.php і, власне, саму секретну сторінку.

Що це нам дає? Припустимо, хакер хоче прочитати секрети Васі і Петі. І він якось дізнався, як називається секретна сторінка (або сторінки). Тоді він може спробувати просто ввести її адресу в рядку браузера, минаючи сторінку авторизації (введення пароля). Щоб уникнути такого проникнення в наші таємниці, потрібно дописати всього пару рядків у код секретних сторінок:

```
<? Php
session_start ();
// Створюємо нову сесію або
// Відновлюємо поточну
print_r ($ _SESSION);
// Виводимо всі змінні сесії
if (!($_ SESSION ['login'] == pit &&
$ _SESSION ['Passwd'] == 123))
// Перевіряємо правильність
// Пароля-логіна
Header ("Location: authorize.php");
// Якщо помилка, то перенаправляємо на
// Сторінку авторизації
```

```
?>
<html>
<head> <title> Secret info </ title> </ head>
... / / Тут розташовується
/ / Секретна інформація:)
</ Html>
```

Код 2-ї версії secret_info.php

Видалення змінних сесії

Крім наіичок реєстрації змінних сесії (тобто робити їх глобальними протягом всього сеансу роботи), корисно також вміти видаляти такі змінні і сесію в цілому.

Функція **session_unregister** (ім'я_змінної) видаляє глобальну змінну з поточної сесії (тобто видаляє її з списку зареєстрованих змінних). Якщо реєстрація проводилася за допомогою $_SESSION ($HTTP_SESSION_VARS для версії PHP 4.0.6 і більш ранніх), то використовують мовну конструкцію unset (). Вона не повертає ніякого значення, а просто знищує зазначені змінні.

Увага! session_unregister застаріла! Слід використовувати unset ($_SESSION ['ім'я змінної']);

Де це може стати в нагоді? Наприклад, для знищення даних про відвідувача (зокрема, логіна і пароля) після його виходу з секретної сторінки. Якщо правильні логін і пароль збережуться і вікно браузера після відвідування сайту не закрили, то будь-який інший користувач цього комп'ютера зможе прочитати закриту інформацію.

Приклад знищення змінних сесії

У файл secret_info.php додамо рядок для виходу на головну сторінку:

```
<? Php
// ... php код
?>
<html>
<head> <title> Secret info </ title> </ head>
... // Тут розташовується
// Секретна інформація:)
<a href="index.php"> На головну </ a>
</ Html>
```

У Index.php знищимо логін і пароль, введені раніше:

Вміст файлу index.php

```
<?
session_start ();
session_unregister ('passwd');
// Знищуємо пароль
unset ($ _SESSION ['login']);
// Знищуємо логін
print_r ($ _SESSION);
// Виводимо глобальні змінні сесії
```

```
?>
<html>
<head> <title> My home page </ title> / head>
....// Домашня сторінка
</ Html>
```

Тепер, щоб потрапити на секретну сторінку, потрібно буде знову вводити логін і пароль.

Для того щоб скинути значення всіх змінних сесії, можна використовувати функцію session_unset ();

Знищити поточну сесію цілком можна командою session_destroy (); Вона не скидає значення глобальних змінних сесії і не видаляє cookies, а знищує всі дані, асоційовані з поточною сесією.

Приклад знищення сесії і глобальних змінних

```
<?
session_start (); // ініціюємо сесію
$test = "Змінна сесії";
$_SESSION ['Test'] = $test;
// Реєструємо змінну $test.
// Якщо register_globals = on,
// То можна використовувати
// Session_register ('test');
print_r ($ _SESSION);
// Виводимо всі глобальні змінні
echo session_id ();
// Виводимо ідентифікатор сесії
echo "<hr>";
session_unset ();
// Знищуємо всі глобальні
// Змінні сесії
print_r ($_SESSION);
echo session_id ();
echo "<hr>";
session_destroy (); // знищуємо сесію
print_r ($ _SESSION);
echo session_id ();
?>
```

У результаті роботи цього скрипта будуть виведені три рядки: у першому - масив з елементом test і його значенням, а також ідентифікатор сесії, у другому - порожній масив і ідентифікатор сесії, в третьому - порожній масив. Таким чином, видно, що після знищення сесії знищується і її ідентифікатор, і ми більше не можемо ні реєструвати змінні, ні взагалі проводити які-небудь дії з сесією.

Розділ 4. Технології програмування мовою Java

4.1. Огляд можливостей мови Java

Мова Java – це об'єктно–орієнтована, незалежна від платформи мова програмування, яка використовується для розробки розподілених застосувань, що працюють в мережі Internet.

Проект Java був представлений корпорацією Sun Microsystems в 1995 році. Система програмування Java дозволяє використовувати World Wide Web для реалізації невеликих інтерактивних прикладних програм – аплетів. Вони розміщуються на серверах Internet, транспортуються по мережі, автоматично встановлюються і запускаються на стороні клієнта як частина документа WWW. Аплет використовує обмежений доступ до ресурсів комп'ютера клієнта, тому він без ризику пошкодження даних на диску може надати довільний мультимедійний інтерфейс, виконувати складні обчислення та інше.

Другим видом програм Java є додатки, що являють собою переносимі коди, які можуть виконуватися на будь–якому комп'ютері, незалежно від його архітектури. Код, що генерується при цьому, представляє собою набір інструкцій для виконання на інтерпретаторі віртуального коду – віртуальній Java-машині (JVM – Java Virtual Machine).

Широкого поширення набули також сервлети та JSP (Java Server Pages), що надають клієнтам можливість доступу до баз даних і додатків на сервері.

Мова Java побудована на синтаксисі мови C++, проте об'єктна модель використана з мови Smalltalk. З цього виходить, що вся схожість з C++ тільки зовнішня. Основні відмінності від від інших мов програмування пов'язані з необхідністю зменшення розмірів програм і збільшення вимог до безпеки переносимих додатків, що працюють в мережі. Java не має вказівного типу даних, що можливо в мовах типу C++, Pascal, а тому збільшує стан захисту пам’яті, звільнившись від роботи з довільними адресами в ній через вказівники. В мові Java змінилися способи обчислень з плаваючою арифметикою, тому, щоб забезпечити переносимість коду між версіями мови, введено ключове слово strictfp, яке вказує компілятору як виконувати арифметичні дії для чисел з плаваючою комою по моделі обчислень попередньої версії.

Системна бібліотека класів мови містить класи та пакети, що реалізують різні базові можливості мови. Методи класів, включених в ці бібліотеки, викликаються з JVM під час інтерпретації Java-програми. В Java всі об'єкти програми розташовані в динамічній пам'яті і доступні за об'єктними посиланнями, які в свою чергу зберігаються в стеку. Це рішення виключило безпосередній доступ до пам'яті, але ускладнило роботу з елементами масивів.

Необхідно відзначити, що об'єктні посилання мови Java містять інформацію про клас об'єктів, на які вони посилаються, а тому вони не вказівники, а дескриптори об'єктів. Наявність дескрипторів дозволяє JVM виконувати перевірку сумісності типів на фазі інтерпретації коду, генеруючи виключення у разі помилки.

В мові Java переглянута і концепція динамічного розподілу пам'яті. В ній відсутні способи звільнення динамічно виділеної пам'яті. Замість цього реалізована система автоматичного звільнення пам'яті, виділеної за допомогою оператора new.

В Java-програмах специфікація класу та його реалізація завжди містяться в одному й тому ж файлі.

Мова Java не підтримує перевантаження операторів і typedef, беззнакові цілі (якщо не рахувати таким тип char), а також використання аргументів по замовчуванню. В Java відсутнє множинне наслідування, існують конструктори, але відсутні деструкції (застосовується автоматична збірка сміття), не використовується оператор goto і слово const, хоч вони є зарезервованими словами мови.

Найбільш істотні нові можливості, що з'явилися в Java, це інтерфейси та багагопоточність (можливість одночасного виконання частин програми).

4.2. Базові типи даних. Оператори. Управляючі конструкції.

У мові Java визначено вісім базових типів даних, розмір кожного з яких залишається незмінним незалежно від платформи. Беззнакових типів в Java не існує.

Ключове слово	Тип	Розмір	Значення, які може зберігати	Значення по замовчуванням
Byte	Байт	8 бит	значення в діапазоні від -2^7 до 2^7-1	0
short	коротке ціле	16 бит	значення в діапазоні від -2^{16} до $2^{16}-1$	0
Int	Ціле	32 бит	значення в діапазоні від -2^{32} до $2^{32}-1$	0
Long	довге ціле	64 бит	значення в діапазоні від -2^{64} до $2^{64}-1$	0
Float	плаваюча крапка	32 бит	значення в діапазоні від$1.7*10^{-38}$ до $1.7*10^{38}$	0.0f
double	подвоєна точність	64 бит	значення в діапазоні від –1.40239846E–45 до 3.40282347E+38	0.0d
Char	Символьний	16 бит	букви, цифри, символи, засновані на 16–бітовому наборі символів Unicode от /u0000 до /uffff	'0x0'
boolean	логічний (булевою)	8 бит	true або false	false

Тип char використовує формат UNICODE завдовжки два байти, що дозволяє використовувати безліч наборів символів, включаючи ієрогліфи.

В Java використовуються:

цілочисельні літерали: 1024,

восьмеричні значення: 015,

шістнадцяткові значення: 0x51.

Цілочисельні літерали створюють значення типу int . Якщо необхідно визначити довгий літерал типу long, в кінці вказується символ L (наприклад: oxffffl). Літерали дійсних чисел записуються з фіксованою крапкою 1.918 або в експоненціальній формі 0.112E–05 і відносяться до типу double. Якщо необхідно визначити літерал типу float, то в кінці слід додати символ F(1.918f). Символьні літерали обмежуються апострофами (‘ a’, ‘ n’, ‘141’, ‘u005a’). Рядки беруться в лапки і є об'єктами (“alfa”). За літерали вважаються булеві значення true і false, а також null – значення по замовчуванню для об'єктів. Літерали у арифметичних виразах автоматично приводяться до типу перетворення. Java автоматично розширює тип кожного byte або short операнда до int. Для звуження перетворень необхідно проводити явне перетворення типу значення. Наприклад, byte b=(byte) 35;

Оператори. Операції над цілими числами: +, – *, %, /, ++,-- та бітові операції &, |, ^, ~ аналогічні операціям більшості мов програмування. Ділення на нуль цілочисельного типу викликає виняткову ситуацію, переповнення не контролюється.

Операції над числами з плаваючою крапкою практично ті ж, що і в інших мовах, але за стандартом IEEE 754 введені поняття нескінченності +infinity і – infinity і значення NAN (Not a Number), яке може бути отримано, наприклад, при знаходженні квадратного кореня з від’ємного числа.

Арифметичні оператори

Основні арифметичні оператори

Оператор	Короткий опис	Приклад
+	Додавання	3+2 або “pre”+”fix”
–	Віднімання	12–3
*	Множення	length*width
/	Ділення	miles/gallons
%	Оператор ділення по модулю (залишок від ділення першого цілочисельного операнда на другий)	10%4
&	Побітове І	Num&musk
|	Побітове АБО	This|that
^	Побітове виключаюче АБО	Tall^short

~	Побітове доповнення	mask=~item
<<	Зрушення вліво	Var<<3
>>	Зрушення вправо	Var>>12
<<<	Зрушення вліво з додаванням нулів	Arg<<<howMuch
>>>	Зрушення вправо з додаванням нулів	Arg>>>howMuch

Булеві операції

Символ	Дія	Приклад		Символ	Дія	Приклад
==	дорівнює	if (a==14)		&	побітове і	if ((a>14)&(a<17))
!=	не дорівнює	if (a!=14)		&&	І	if((a>14)&&(b>17))
<	менше ніж	if (a<14)		|	побітове або	if((a==16)|(b==19))
>	більше ніж	if (a>14)		||	Або	if ((a==25)||(a==8))
<=	менше або дорівнює	if (a<=1)		!	Ні	if (!((a==16)||(a==3)))
>=	більше або дорівнює	if (a>=4)		^	Виключаюче або	if ((a==16)^(a==19))

Оператор присвоювання

Оператор	Еквівалентний оператор		Оператор	Еквівалентний оператор
a+=b	a=a+b		a&=b	a=a&b
a-=b	a=a-b		a|=b	a=a|b
a*=b	a=a*b		a^=b	a=a^b
a/=b	a=a/b		a<<=b	a=a<<b
a%=b	a=a%b		a>>=b	a=a>>b

Пріоритет і асоціативність операцій

Вищий пріоритет
. ( ) [ ]
++ -- ! ~
New
* / %
+ –
<< >> <<< >>>
< > <= >=

== !=
&
^
|
&&
||
? :
= += –= *= /= %= &= ||= <<= >> = ^=
Нижчий пріоритет

Всі операції виконуються в напрямку зліва направо.

Для перетворення базових типів застосовується операція (type), де в якості (type) використовується один з сумісних базових типів. При приведенні типів даних меншої довжини до типів більшої довжини ніяких операцій проводити не потрібно. Слід звернути увагу на те, що операція присвоювання результатів арифметичних операцій для базових типів char, byte, short викликає помилку компіляції, оскільки при обчисленнях проводиться перетворення до типу int, а Java не дозволяє привласнювати змінній значення більш довшого типу, якщо тільки це не константи. Виняток становлять оператори інкремента, декремента та оператори +=, –=, *=, /=.

В іменах змінних не можуть використовуватися символи арифметичних і логічних операторів, а також символ ‘#’. Припустимим є застосування символів ‘ $’ і ‘ _’, в тому числі і в першій позиції імені.

```
public class TypeByte {
    private static int j;
    public static void main(String[] args) {
        int i = 3;
        byte b = 1,
        b1 = 1 + 2;
        //b = b1 + 1; //помилка приведення типів
        b = (byte)(b1 + 1);//0
        show(b);
        //b = –b; // помилка приведення типів
        b = (byte)–b;//1
        show(b);
        //b = +b1; // помилка приведення типів
        b = (byte)+b1; //2
        show(b);
        b1*= 2; //3
        show(b1);
```

```
        b1++; //4
        show(b1);
        //b = i; // помилка приведення типів
        b = (byte)i; //5
        show(b);
        b+= i++; //працює!!!  //6
        show(b);
        float f = 1.1f;
        b /= f; //працює!!!  //7
        show(b);
    }
    static void show(byte b){
        System.out.println(j + " res=" + b);
        j++;
    }
}
```

В результаті буде виведено:

```
0 res=4
1 res=–4
2 res=3
3 res=6
4 res=7
5 res=3
6 res=6
7 res=5
```

Змінні базових типів, оголошені як члени класу, зберігають нульові значення, відповідні своєму типу. Якщо змінні оголошені як локальні змінні в методі, то перед використанням вони обов'язково мають бути проініціалізовані.

До операторів відноситься також оператор визначення приналежності типу instanceof, оператор [ ] і тернарний оператор ?: (if–then–else).

Логічні операції виконуються над значеннями типу boolean (true або false).

```
// приклад бітові оператори : Operators.java
public class Operators {
    Public static void main(String[] args) {
        System.out.println("5%1=" + 5%1 + "  5%2=" + 5%2);
        int b1 = 0xe;//14 или 1110
        int b2 = 0x9;//9  или 1001
        int i = 0;
        System.out.println(b1 + "|" + b2 + " = " + (b1|b2));
        System.out.println(b1 + "&" + b2 + " = " + (b1&b2));
        System.out.println(b1 + "^" + b2 + " = " + (b1^b2));
        System.out.println(     "~" + b2 + " = " + ~b2);
        System.out.println(b1 + ">>" + ++i + " = " + (b1>>i));
```

```
            System.out.println(b1 + "<<" + i + " = " + (b1<<i++));
            System.out.println(b1 + ">>>" + i +" = " + (b1>>>i));
        }
    }
```

Результатом виконання даного коду буде

```
5%1=0  5%2=1
14|9 = 15
14&9 = 8
14^9 = 7
~9 = –10
14>>1 = 7
14<<1 = 28
14>>>2 = 3
```

Тернарний оператор "?" використовується у виразах:

```
booleanexp ? value0 : value1
```

Якщо booleanexp = true, обчислюється значення value0 і воно стає результатом виразу, інакше результатом є значення value1.

Оператор instanceof повертає значення true, якщо об'єкт є екземпляром даного класу, наприклад:

```
Font obj = new Font("Courier", 1, 18);
if (obj instanceof java.awt.Font) {
    /*оператори*/
}
```

Числові параметри при оголошенні об'єкту класу Font вказують на стиль і розмір шрифту. Результатом дії оператора instanceof буде істина, якщо об'єкт є об'єктом одного з підкласів класу, на приналежність до якого перевіряється даний об'єкт, але не навпаки. Перевірка на приналежність об'єкту до класу Object завжди дасть істину як результат. Результат застосування цього оператора по відношенню до null завжди хибність, тому що null не можна зарахувати до якого–небудь типу. А між тим літерал null можна передавати в методи по посиланню на будь–який об'єктний тип і використовувати як значення, що повертається.

Оператори управління. Оператор if дозволяє альтернативний вибір двох операторів, виконуючи один з них або інший.

```
if (boolexp) {
    /*оператори*/
}
else { //може бути відсутнім
    /*оператори*/
}
```

Цикли в мові Java можна організувати з допомогою операторів:

```
while (boolexpr) {
    /*оператори*/
}
```

```
do {
    /*оператори*/
} while (boolexp);

for(exp1; boolexp; exp3){
    /*оператори*/
}
```

Оператори циклу виконуються допоки булевий вираз boolexp знаходиться рівним true.

Оператор switch. Оператор switch передає управління на виконання одного з декількох операторів залежно від значення виразу exp.

```
switch(exp) {
    case exp1:/*оператори, якщо exp==exp1*/
        break;
    case exp2:/*оператори, якщо exp==exp2*/
        break;
    default: /* оператори Java */
}
```

При збігу умов виду exp==expN виконуються підряд всі оператори N–ого блоку до тих пір, поки не зустрінеться оператор break.

Розширення можливостей отримали оператор переривання циклу break і оператор переривання ітерації циклу continue, які можна використовувати з міткою, для забезпечення виходу з вкладених циклів, наприклад:

```
// приклад: вихід за цикл, помічений OUT
public class DemoLabel {
    public static void main(String[] a) {
        int j = –3;
        OUT: while (j < 10) {
        if (j == 0)
            break OUT;
        else {
            j++;
            System.out.println(j);
        }
    }
    System.out.println("end");
    }
}
```

Тут оператор break розриває цикл, помічений міткою OUT. При цьому немає необхідності у використанні оператора goto для виходу з вкладених циклів.

Цикл for. В операторі for передбачені місця для всіх чотирьох частин циклу. Нижче приведена загальна форма оператора запису for.

for(ініціалізація; завершення; ітерація) тіло циклу;

Будь–який цикл, записаний за допомогою оператора for, можна записати у вигляді циклу while і навпаки. Якщо початкові умови такі, що при вході в цикл умова завершення вже виконана, то оператори тіла циклу і ітерації не виконуються жодного разу. У канонічній формі циклу for відбувається збільшення цілого значення лічильника з мінімального значення до певної межі.

Змінні можна оголошувати всередині розділу ініціалізації оператора for. Змінна, оголошена всередині оператора for, діє в межах цього оператора.

При роботі з масивами та колекціями з'явилася можливість отримувати доступ до їх елементів без використання індексів або ітераторів. Наприклад,

```
int[] array = {1, 3, 5, 11};
for(int i : array)
    System.out.print(" " + i);
```

Але змінити значення елементів масиву за допомогою такого циклу не можливо.

4.3. Основні поняття мови Java

Класи та об'єкти. В класи Java входять змінні (члени класу), методи та конструктори. Всі функції визначаються всередині класів і називаються методами. Неможливо створити метод, що не є методом класу або оголосити метод поза класом. Специфікатори доступу public, private, protected впливають тільки на те, перед чим вони стоять. Елементи по замовчуванню доступні лише для класів з даного пакету. Оголошення класу має вигляд:

[специфікатори] class імя_класу [extends суперклас] [implements список_інтерфейсів]

Специфікатор доступу класу може бути public (клас доступний об'єктам даного пакету і поза пакетом), final (клас не може мати підкласів), abstract (клас містить абстрактні методи, об'єкти такого класу можуть створювати тільки підкласи). По замовчуванню специфікатор встановлюється в friendly (клас доступний в даному пакеті). Наведемо простий приклад класу:

```
class Subject {
    public String name;
    private int age;
    public Subject() {//конструктор
        name = "NoName";
        age = 0;
    }
    public Subject(String n) {//конструктор
        name = n;
    }
    public void setAge(int a) {//метод
        age = a;
    }
    void show() { //метод
```

```
            System.out.println("Ім’я: " + name + ", Вік: " + age);
        }
    }
```

Клас Subject містить два поля name і age, помічені як public і private. Значення поля age можна змінювати тільки за допомогою методів, наприклад, setage(). Поле name доступно і безпосередньо через об'єкт класу Subject. Доступ до методів і public полів даного класу здійснюється тільки після створення об'єкту даного класу:

```
public class SubjectDemo {
    public static void main(String[] args) {
        Subject ob = new Subject("Балаганов");
        ob.name = "Шура Балаганов";
        //ob.age = 19;// поле недоступно
        ob.setAge(19);
        ob.show();
    }
}
```

Компіляція та виконання даного коду приведуть до виводу на консоль такої інформації:

Ім'я: Шура Балаганов, Вік: 19

Класи з прикладів 1 і 2 можна зберігати перед компіляцією в одному файлі Subjectdemo.java, причому ім'я цьому файлу дається на ім'я public класу, тобто Subjectdemo.

Об'єкт класу створюється за два кроки. Спочатку оголошується посилання на об'єкт класу. Потім за допомогою оператора new створюється екземпляр об'єкту, наприклад:

```
String str; //оголошення посилання
str = new String(); //створення об'єкту
```

Проте ці дві дії зазвичай об'єднують в одне:

```
String s = new String(); /*оголошення посилання та створення об'єкта*/
```

Оператор new викликає конструктор, тому в круглих дужках можуть стояти аргументи, що передаються конструктору. Операція присвоювання для об'єктів означає, що два посилання вказуватимуть на одну і ту ж ділянку пам'яті.

Класи та відношення. Класи визначають структуру та поведінку деякого набору елементів предметної області, для якої розробляється програмна модель. Клас описує сукупність об'єктів із загальними атрибутами, методами, відношеннями та семантикою.

Кожен клас має своє ім'я, що відрізняє його від інших класів, та відноситься до певного пакету. Ім'я класу в пакеті має бути унікальним. Фізично пакетом є каталог, в якому розміщені програмні файли з реалізацією класів.

Класи дозволяють розбити поведінку складних систем на просту взаємодію взаємозв'язаних об'єктів. При проектуванні системи необхідно не

тільки ідентифікувати сутності, але і вказати, як вони співвідносяться один з одним.

Відношенням називається зв'язок між класами. В об'єктно-орієн–тованому проектуванні особливе значення мають чотири типи відношень: залежності, узагальнення, асоціації та реалізації.

Залежністю називається відношення використання, що визначає, як зміна стану об'єкту одного класу може вплинути на об'єкт іншого класу, який його використовує. При цьому зворотнє твердження в загальному випадку невірно. Залежності застосовуються тоді, коли екземпляр одного класу використовує екземпляр іншого, наприклад, як параметр методу.

Узагальнення означає, що об'єкти підкласу можуть використовуватися всюди, де зустрічаються об'єкти суперкласу, але у жодному випадку не навпаки. Визначення суперкласу є більш загальним, ніж визначення підкласу. Підклас успадковує властивості батька (атрибути і методи). Ідентифікація суперкласів і підкласів здійснюється з використанням моделі предметної області, оскільки за її допомогою можливий аналіз всіх понять в більш загальних і абстрактних термінах. У результаті поліпшується розуміння коду (особливо для систем з великою кількістю класів), зменшується об'єм повторюваної інформації.

Наприклад, поняття Cashpayment, Creditpayment, Checkpayment дуже схожі одне на інше, і в цьому випадку розумно організувати їх в ієрархію узагальнення – спеціалізацію класів. Клас Payment представляє більш загальне поняття, а його підкласи – спеціалізовані властивості.

Підклас створюється у випадках, якщо:

- він має додаткові атрибути, які цікавлять розробника;
- він має додаткові асоціації, що цікавлять розробника;
- йому відповідає поняття, яке керується, оброблюється, реагує або використовується способом, відмінним від способу, що визначається суперкласом або іншими підкласами;
- він представляє об'єкту поведінку відмінну від поведінки, яка визначається суперкласом або іншими підкласами.

Реалізацією називається відношення між класифікаторами (класами, інтерфейсами), при якому один з них описує контракт (інтерфейс суті), а інший гарантує його виконання.

Асоціації показують, що об'єкти одного класу пов'язані з об'єктами іншого класу і відображають деяке відношення між ними. В цьому випадку можна переміщуватися (за допомогою виклику методів) від об'єктів одного класу до об'єктів іншого. Агрегація – асоціація, що моделює взаємозв'язок “частина/ціле” між класами, які в той же час можуть бути рівноправними. Обидва класи при цьому знаходяться на одному концептуальному рівні і жоден не є більш важливим, ніж інший.

Тіло класу в системі Java може містити оголошення полів даних, конструкторів, методів, внутрішніх класів та інтерфейсів, а також логічні блоки, що використовуються як правило для ініціалізації полів.

Змінні класу і константи. Дані – члени класу, які називаються полями або змінними класу, оголошуються в класі таким чином:

специфікатор тип ім'я;

У мові Java можуть використовуватися змінні класу, оголошені один раз для всього класу із специфікатором static і однакові для всіх екземплярів класу, або змінні екземпляра, що створюються для кожного екземпляра класу. Змінні оголошуються із специфікаторами доступу public, private, protected або по замовчуванню без специфікатора. Окрім членів класу в класі використовуються локальні змінні і параметри методів. Змінні із специфікатором final є константами. Специфікатор final можна використовувати для змінної, оголошеної в методі, а також для параметра методу.

Оголосити та проініціалізувати значення змінних класу і локальних змінних методу, а також параметри методу можна таким чином:

```
class Myclass {
    int x; // змінна екземпляра класу
    int y = 2; // змінна екземпляра класу
    final int YEAR = 2009; // константа
    static int bonus; // змінна класу
    static int b = 1; // змінна класу
    void init(int z){// параметр методу
        z = 3;// переініціалізація
        int a;// локальна змінна методу
        a = 4;// ініціалізація
    }
}
```

У приведеному прикладі використані дані базових типів, що не є посиланнями на об'єкти. Дані можуть бути посиланнями, яким можна призначити реальні об'єкти за допомогою оператора new.

Обмеження доступу. Java надає декілька рівнів захисту, які забезпечують можливість налаштування зони видимості даних і методів. Завдяки наявності пакетів Java повинна вміти працювати з чотирма категоріями видимості між елементами класів:

класи і підкласи в тому ж пакеті (по замовчуванню);

незалежні класи (private);

підкласи в поточному та інших пакетах (protected);

класи, які не є підкласами та не входять в той же пакет (public).

Елемент (атрибут або метод), оголошений public, доступний з будь-якого місця поза класом. Все, що оголошене private, доступне тільки всередині класу і ніде більше. Якщо в елемента взагалі не вказаний модифікатор рівня доступу, то такий елемент буде доступний з підкласів і класів того ж пакету. Саме такий рівень доступу використовується по замовчуванню. Якщо ж необхідно, щоб

елемент був доступний з іншого пакету, але тільки підкласам того класу, якому він належить, потрібно оголосити такий елемент із специфікатором protected.

Конструктори. Конструктор – це метод, який автоматично викликається при створенні об'єкту класу і виконує дії з ініціалізації об'єкту. Конструктор має те ж ім'я, що і клас; викликається не по імені, а тільки разом з ключовим словом new при створенні екземпляра класу. Конструктор не повертає значення, але може мати параметри і бути перезавантажуваним.

Деструкції в мові Java не використовуються, об'єкти знищуються складальником сміття після припинення їх використання. Аналогом деструкцій є метод finalize(). Виконуюче середовище мови Java викликатиме його кожного разу, коли складальник сміття знищуватиме об'єкти цього класу, яким не відповідає жодне посилання. Продемонструємо на прикладі сутність перезавантадення конструктора:

```
class NewBook {
    private String title, publisher;
    private float price;
    public NewBook() {
        title = “NoTitle”;
    }
    public NewBook(String t, String pub, float p) {
        title = new String(t);
        publisher = pub;
        price = p;
    }
}
```

Об'єкт класу Newbook може бути створений двома способами, які використовують один з таких конструкторів:

```
Newbook tips1; // оголошення
tips1 = new Newbook();// ініціалізація
Newbook tips2 = new Newbook("Java2", "Ноутон", 9.f);
```

Оператор new викликає конструктор, тому в круглих дужках можуть стояти аргументи, що передаються конструктору.

Якщо конструктор в класі не визначений, Java надає конструктор по замовчуванню, який ініціалізував об'єкт значеннями теж по замовчуванню. Якщо ж конструктор з параметрами визначений, то конструктор по замовчуванню стає недоступним і для його виклику необхідне явне оголошення такого конструктора.

У наступному прикладі оголошений клас Locate з двома полями (атрибутами), конструктором і методами для ініціалізації та вибору значень атрибутів.

```
class Locate {
    private double x, y; /*по замовчуванню x=0 і y=0 */
    public Locate(){
```

```
            x = 1;
            y = 1;
        }
        public void setx(double a){
            x = a;
        }
        void sety(double b){ /*видимість по замовчуванню*/
            y = b;
        }
        public double getx(){
            return x;
        }
        public double gety(){
        return y;
        }
    }

    public class Distance {
        public static void main(String[] args){
            //локальні змінні не є членами класу
            Locate t1 = new Locate();
            Locate t2 = new Locate();
            double dx, dy, distance;
            t1.setX(5);
            t1.setY(10);
            t2.setX(2);
            t2.setY(6);
            dx = t1.getX() – t2.getX();
            dy = t1.getY() – t2.getY();
            /* обчислення відстані */
            distance = Math.sqrt(dx*dx + dy*dy);
            //distance = Math.hypot(dx, dy);//java 5.0
            System.out.print("Відстань рівна: " + distance);
        }
    }
```

В результаті буде виведено:

відстань рівна: 5.0

Тут використані статичні методи sqrt() або hypot()) з класу Math, які викликаються без оголошення об'єкту вказаного класу. Клас Math містить тільки статичні методи для фізичних і технічних розрахунків, а також константи E і PI.

Простий аплет. Одна з цілей розробки Java: створення аплетів – невеликих програм, що запускаються web-браузером. Оскільки аплети мають

бути безпечними, вони обмежені в своїх можливостях, хоч і залишаються могутнім інструментом підтримки Web-програмування на стороні клієнта.

```
// приклад: простий аплет
import java.applet.Applet;
import java.awt.*;
public class FirstApplet extends Applet {
    private String date;
    public void init() {
        date = new java.util.Date().toString();
    }
    public void paint(Graphics g) {
        g.drawString("Аплет стартував:", 50, 15);
        g.drawString(date, 50, 35);
    }
}
```

Для виведення поточного часу та дати в даному прикладі був використаний об'єкт Date з пакету java.util. Метод tostring() використовується для перетворення інформації, що міститься в об'єкті, в рядок для подальшого виводу в аплет за допомогою методу drawstring(). Цифрові параметри цього методу позначають горизонтальну та вертикальну координати початку рядка, причому відлік ведеться від лівого верхнього кута аплета.

Аплету не потрібний метод main() – код його запуску розміщується в методі init() або paint(). Для запуску аплета потрібно розмістити посилання на його клас в HTML–документ і відкрити цей документ web-браузером, що підтримує Java. При цьому можна обійтися дуже простим фрагментом (тегом) <applet> всередині HTML документа view.html:

```
<html><body>
    <applet code= FirstApplet.class width=300 height=300>
    </applet>
</body></html>
```

Сам файл FirstApplet.class при такому зверненні повинен знаходитися в тому ж каталозі, що і HTML–документ.

4.4. Масиви

Масиви елементів базових типів складаються зі значень, проіндексованих починаючи з нуля. При роботі з масивами масив перш за все необхідно оголосити.

Синтаксис оголошення масиву:

Тип ім'я_масива[ ]

Тип – тип елементів, що складають масив. Тип і число елементів масиву визначають об'єм пам'яті, необхідний для розміщення масиву.

Іменем масива може виступати будь-який ідентифікатор.

Ідентифікатори мови Java повинні починатися з букви будь-якого регістра або символів "_" та "$". Далі можуть слідувати і цифри. Наприклад, _java -

правильний ідентифікатор, а 1_$ - ні. Ще одне обмеження Java виникає з його властивості використовувати для зберігання символи кодування Unicode, тобто можна застосовувати тільки символи, які мають порядковий номер більше 0xC0 в розкладці символів Unicode.

При оголошенні масиву пара квадратних дужок, що визначають масив, може розташовуватися як після імені масиву, так і перед його ім'ям. Другий варіант більше гармонує із способом визначення об'єктів в мові Java. Наприклад, наступні оголошення ідентичні:

int []d; //об'ява масиву цілих чисел

int d[]; //об'ява масиву цілих чисел

У наступному прикладі оголошуються відразу декілька масивів одного і того ж типу:

int[] array1,array2, array3;

Всі масиви в мові Java є динамічними, тому для створення масиву потрібне виділення пам'яті за допомогою оператора new або ініціалізації, які створюють в пам'яті новий екземпляр типу даних посилання.

Наприклад:

d=new int[10];

Цей оператор створює масив з десяти цілих чисел і привласнює змінній d значення адреси першого елементу масиву. Кожен елемент масиву ініціалізується за правилами замовчування, передбачених для кожного типу даних. В цьому випадку всі змінні стають рівними нулю. (Масив об'єктів якогось класу за замовчуванням ініціалізується значеннями null).

Приклад одночасного оголошення і створення масиву за допомогою оператора new:

F[] c=new F[10]; // масив покажчиків на об'єкти

String b[]=new String[10];

Для звернення до окремого елементу масиву використовується індекс, взятий в квадратні дужки. Наприклад, за допомогою оператора присвоєння можна присвоїти значення першим двом елементам масиву d:

d[0]=1;

d[1]=2;

Значення елементів неініціалізованих масивів, для яких виділена пам'ять, встановлюється в нуль. Імена масивів є посиланнями. Для оголошення посилання на масив можна записати порожні квадратні дужки після імені типу, наприклад: : int a[]. Аналогічний результат дає запис int[] a.

```
/* приклад: заміна від'ємних елементів масиву на максимальний */
public class FindReplace {
    public static void main(String[] args) {
        int myArray[];       //оголошення без ініціалізації
        int mySecond[] = new int[100]; /*виділення пам'яті з ініціалізацією значень по замовчуванню */
        int a[] = {5,10,0,–5,16,–2}; //оголошення з ініціалізацією
        int max = a[0];
```

```
            for(int i = 0; i < a.length; i++) //пошук max–елемента
                if(max < a[i])
                    max = a[i];
            for(int i = 0; i < a.length; i++)   { //заміна
                if( a[i] < 0 )
                    a[i] = max;
            mySecond[i] = a[i];
            System.out.println("a[" + i + "]= " + a[i]);
        }
        myArray = a; //посилання на масив a
    }
```

В результаті виконання програми буде виведено:

```
a[0]= 5
a[1]= 10
a[2]= 0
a[3]= 16
a[4]= 16
a[5]= 16
```

Присвоєння mysecond[i]=a[i] приведе до того, що частині елементів масиву mysecond, а саме шести, будуть присвоєні значення елементів масиву a. Решта елементів mysecond збережуть значення, отримані при ініціалізації, тобто нулі. Якщо ж присвоєння організувати у вигляді mysecond=a або myarray=a, то обидва масиви, що беруть участь в присвоєнні, отримають посилання на масив a, тобто обидва міститимуть по шість елементів і посилатимуться на одну і ту ж ділянку пам'яті.

Багатовимірних масивів в Java не існує, але можна оголошувати масиви масивів. Для задання початкових значень масивів існує спеціальна форма ініціалізації, наприклад:

```
int arr[][] = {
    { 1 },
    { 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9, 0 }
};
```

Перший індекс вказує на порядковий номер масиву, наприклад, arr[2][0] вказує на перший елемент третього масиву, а саме на значення 4.

У наступній програмі створюються та ініціалізуються масиви масивів рівної довжини (матриці) і виконується множення однієї матриці на іншу.

```
/* приклад : добуток двох матриць*/
public class Matrix {
    private int[][] a;
    Matrix(int n, int m){
    // створення і заповнення випадковими значеннями
        a = new int[n][m];
```

```
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                a[i][j]= (int) (Math.random() * 5);
        show();
    }
    public Matrix(int n, int m, int k){
        a = new int[n][m];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++) {
                a[i][j]= k;
            }
        if(k!=0) show();
    }
    public void show() {
        System.out.println("Матріца : " + a.length+
            " на " + a[0].length);
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[0].length; j++)
                System.out.print(a[i][j]+ " ");
            System.out.println();
        }
    }
    public static void main(String[] args){
        int n = 2, m = 3, l = 4;
        Matrix p = new Matrix(n, m);
        Matrix q = new Matrix(m, l);
        Matrix r = new Matrix(n, l, 0);
        for (int i = 0; i < p.a.length; i++)
            for (int j = 0; j < q.a[0].length; j++)
                for (int k = 0; k < p.a[0].length; k++)
                    r.a[i][j] += p.a[i][k]* q.a[k][j];
        System.out.println("Добуток матриць: ");
        r.show();
    }
}
```

Оскільки значення елементам масивів привласнюються за допомогою випадкових чисел по методу random(), то одним з варіантів виконання коду може бути:

```
Матриця : 2 на 3
1 1 1
2 4 1

Матриця : 3 на 4
0 1 2 3
```

3 1 0 4
3 4 0 4

Добуток матриць:

Матриця : 2 на 4
6 6 2 11
15 10 4 26

Якщо об'єкт створений всередині класу, то він має прямий доступ до полів, оголошених як private.

Клас MATH. Розглянемо приклад обробки значення випадкового числа, отриманого за допомогою методу random() класу Math. У класі Math існує ряд інших корисних методів, таких як floor(), ceil(), rint(), round(), max(параметр, параметр), min(параметр, параметр), які виконують заокруглення, пошук екстремальних значень, знаходження найближчого цілого та інше.

```
/* приклад : використання методів класу Math при роботі з масивом випадкових чисел*/
public class Mathmethod {
    public static void main(String[] args){
        final int Max_val = 10;
        double d, max = 0, min = Max_val;
        d = Math.random() * Max_val;
        System.out.println("d = " + d);
        System.out.println("Заокр. до найближчого цілого =" +
            Math.round(d));
        System.out.println("Найбільше ціле, <= початк. числа =" +
            Math.floor(d));
        System.out.println("Найбільше ціле, >= початк. числа =" +
            Math.ceil(d));
        System.out.println("Найближ. ціле знач. початк. числа =" +
            Math.rint(d));
    }
}
```

Один із варіантів виконання коду представлений нижче:

d = 0.08439575016076173
Заокр. до найближчого цілого =0
Найбільше ціле, <= початк. числа =0.0
Найбільше ціле, >= початк. числа =1.0
Найближ. ціле знач. початк. числа =0.0

Масиви об'єктів насправді є масивами посилань, які по замовчуванню проініціалізовані значенням null.

```
// приклад : копіювання масиву
```

```
public class Arraycopydemo {
    public static void main(String[] args){
        int mas1[] = { 1, 2, 3 }, mas2[] = { 4, 5, 6, 7, 8, 9 };
        System.out.print("mas1[]: ");
        show(mas1);
        System.out.print("\nmas2[]: ");
        show(mas2);
        //копіювання масиву mas1[] у mas2[]
        System.arraycopy(mas1, 0, mas2, 2, 3);
        /*   0 – mas1[] копіюється, починаючи з першого елементу
        2 – елемент, з якого починається заміна
        3– кількості копійованих елементів */
        System.out.println("\n після arraycopy(): ");
        System.out.print("mas1[]: ");
        show(mas1);
        System.out.print("\n mas2[]: ");
        show(mas2);
    }
    private static void show(int[] mas){
        int i;
        for (i = 0; i < mas.length; i++)
            System.out.print(" " + mas[i]);
    }
}
```

Результат:

```
mas1[]:  1 2 3
mas2[]:  4 5 6 7 8 9
після arraycopy():
mas1[]:  1 2 3
mas2[]:  4 5 1 2 3 9
```

Всі масиви зберігаються в одній з підобластей пам'яті (heap), що виділена системою для роботи віртуальної машини. Визначити загальний обсяг пам'яті та обсяг вільної пам'яті можна за допомогою методів totalmemory() та freememory() класу Runtime.

```
/* приклад : інформація про стан оперативної пам'яті */
public class Runtimedemo {
    public static void main(String[] args){
        Runtime rt = Runtime.getRuntime();
        System.out.println("Повний об'єм пам'яті: " + rt.totalMemory());
        System.out.println("Вільна пам'ять: " + rt.freeMemory());
        double d[] = new double[10000];
        System.out.println("Вільна пам'ять після оголошення масиву: " +
rt.freeMemory());
```

```
            try {
                rt.exec("mspaint"); //запуск mspaint.exe
            } catch (java.io.IOException e) {
                System.out.println("Помилка виконання " + e);
            }
            System.out.println("Вільна пам'ять після запуску mspaint.exe: " +
rt.freeMemory());
        }
    }
```

В результаті виконання цієї програми може бути виведена, наприклад, така інформація:

Повний об'єм пам'яті: 2031616

Вільна пам'ять: 1903632

Вільна пам'ять після оголошення масиву: 1823336

Вільна пам'ять після запуску mspaint.exe: 1819680

Об'єкт класу Runtime створюється за допомогою виклику статичного методу getruntime(), що повертає об'єкт Runtime, який асоційований з даним додатком. Запуск зовнішніх додатків здійснюється за допомогою методу exec(), один з параметрів якого може застосовуватися рядок з іменем додатка, що запускається. Зовнішній додаток використовує для своєї роботи пам'ять операційної системи.

4.5. Обробка виключних ситуацій

Якщо ваша програма порушить семантичні правила мови Java, то віртуальна машина Java (JVM) негайно відреагує на це видачею помилки під назвою "виняткова ситуація". Приклад такої ситуації - вихід за межі масиву. Вона може виникнути при спробі звернутися до елементу за межами границь масиву. Деякі мови програмування ніяк не "реагують" на помилки програміста і дозволяють помилковим програмам виконуватися. Але Java не відноситься до таких мов. І тому програма ретельно перевіряє всі можливі місця, де може виникнути потенційна помилка.

У разі виявлення помилки збуджуються (throw) виняткові ситуації. Якщо є обробники таких ситуацій, вони перехоплюють їх (catch). В подальшому відбувається їх обробка відповідним чином.

Програми на мові Java можуть самостійно збуджувати виняткові ситуації. Для цього використовується оператор throw. Якщо у блоці програмного коду зустрічається оператор throw, виконання методу переривається і управління передається в той метод, який викликав помилковий. Якщо виняткова ситуація може бути оброблена методом, то викликається його обробник. Якщо ж це неможливо, то потік управління передається далі, і так відбувається до того моменту, коли виняткова ситуація не буде перехоплена або доки її не перехопить віртуальна машина Java. В останньому випадку виконання програми переривається і виводиться повідомлення про помилку.

У мові Java кожна виняткова ситуація реалізується як екземпляр класу Throwable або його спадкоємців. Коли в програмі потрібно відстежити можливу виняткову ситуацію, потрібно встановити обробник або декілька обробників. На практиці це оформлюється у вигляді блоку try-catch:

```
try{
    // В цьому місці можливе збудження
    // виняткової ситуації
} catch (ТипВинятковоїСитуації){
    // Тут здійснюється обробка
    // перехопленої виключної
    // ситуації
}
```

Але не завжди виняткові ситуації пов'язані з фатальними збоями. Розглянемо, наприклад, ситуацію, коли програма просто не знайшла якийсь файл в каталозі. В цьому випадку можна перехопити таку помилку. В обробник виняткової ситуації потрібно вставити оператор виклику діалогової панелі, де користувач вкаже місцерозташування цього файлу. Після усунення цієї проблеми програма може бути запущена з того місця, де її виконання було перерване.

Методи, в яких може виникнути виняткова ситуація, описуються так:

```
static void SomeMethod () throws FileNotFoundException {-}
```

У цьому описі оператор throws означає, що метод потенційно може створювати/викликати виняткову ситуацію FileNotFoundException. Вона пов'язана з тим, що відповідний файл не знайдений. Тепер будь-який виклик цього методу в програмі має обрамлюватись описом блоку try-catch. В іншому випадку компілятор видасть помилку і не обробить вихідний текст програми. Типове вирішення проблеми може виглядати наступним чином:

```
try{
    ...
    static void SomeMethod ();
    ...
}catch (FileNotFoundException exception){
    // Дії, що вживаються
    // для усунення помилки
}
```

Недоліком цього способу є те, що він дещо громіздкий. Проте перевага полягає в тому, що будь-яка можлива помилкова ситуація гарантовано буде усунена.

4.6. Наслідування

Один клас (підклас) може успадковувати змінні і методи іншого класу (суперкласу), використовуючи ключове слово extends. Підклас має доступ до всіх відкритих змінних і методів батьківського класу, неначе вони знаходяться

в підкласі. Одноасно підклас може мати методи з тим же ім'ям і сигнатурою, що і методи суперкласу. В цьому випадку підклас перевизначає методи батьківського класу.

В наступному прикладі метод show(), що перевизначається, знаходиться в двох класах Bird і Eagle. За принципом поліморфізму викликається метод, найбільш близький до поточному об'єкту.

```
class Bird {
    private float price;
    private String name;
    public Bird(float p, String str) { //конструктор
        price = p;
        name = str;
    }
    public float getPrice(){
        return price;
    }
    public String getName(){
        return name;
    }
    void show(){
        System.out.println("назва: " + name + ", вартість: "+ price);
    }
}

class Eagle extends Bird {
    private boolean fly;
    public Eagle(float p, String str, boolean f) {
        super(p, str); //виклик конструктора суперкласу
        fly = f;
    }
    void show(){
        System.out.println("назва:" + getName() + ", вартість: " +
            getPrice() + ", політ:" + fly);
    }
}

public class BirdSample {
    public static void main(String[] args) {
        Bird b1 = new Bird(0.85F, "Яструб");
        Bird b2 = new Eagle(10.55F, "Білий Орел", true);
        b1.show(); // виклик show() класу Bird
        b2.show(); // виклик show() класу Eagle
    }
}
```

Об'єкт b1 створюється за допомогою виклику конструктора класу Bird. Відповідно, при виклику методу show(), викликається версія методу з класу Bird. При створенні об'єкту b2 посилання типу Bird ініціалізується об'єктом типа Eagle. При такому способі ініціалізації посилання на суперклас отримує доступ до методів, перевизначених в підкласі.

При оголошенні полів, що збігаються в суперкласі та підкласах, їх значення не перевизначаються і ніяк не перетинаються, тобто існують в одному об'єкті незалежно один від одного. В цьому випадку доступ до необхідного значення певного поля, що належить класу в ланцюжку наслідування, забезпечує програміст.

Наведемо приклад, в якому продемонструємо доступ до полів з однаковими іменами при наслідуванні:

```
class A {
    int x = 1, y = 2;
    public A() {
        y = getX();
        System.out.println("у класі A після виклику getX()  x=" + x + "  y="
+ y);
    }

    public int getX(){
        System.out.println("у класі A");
        return x;
    }
}

class B extends A {
    int x = 3, y = 4;
    public B() {
        System.out.println("у класі B  x="+x+"  y="+y);
    }

    public int getX(){
        System.out.println("у класі B");
        return x;
    }
}

public class DemoAB {
    public static void main (String[] args) {
        A objA = new B();
        B objB = new B();
```

```
            System.out.println(objA.x);
            System.out.println(objB.x);
        }
    }
```

В результаті виконання даного коду послідовно буде виведено:

```
у класі B
у класі A після виклику getX()  x=1  y=0
у класі B  x=3  y=4
у класі B
у класі A після виклику getX()  x=1  y=0
у класі B  x=3  y=4

x=1
x=3
```

В разі створення об'єкту objA проініціалізував посилання на клас A об'єктом класу B. При цьому отриманий доступ до поля x класу A. В другому випадку при створенні об'єкту objB класу B дістав доступ до поля x класу B. Проте скориставшись перетворенням типів вигляду: ((A)objB).x або((B)objA).x можна легко дістати доступ до поля x з відповідного класу.

Проілюструємо на прикладі конструктора класу A одну із сторін поліморфізму :

```
public A() {
y = getX();
}
```

Метод getX() міститься і в класі A, і в класі B. При створенні об'єкту класу B одним із способів:

```
A objA = new B();
B objB = new B();
```

в будь-якому разі спочатку викликається конструктор класу A. Але, оскільки створюється об'єкт класу B, то і метод getX() відповідно викликається той, що належить класу B. Цей метод в свою чергу оперує полем x, що ще не було проініціалізованим для класу B. В результаті у набуде значення x по замовчуванню, тобто нуль.

Не можна створити підклас для класу, оголошеного із специфікатором final:

```
// клас First не може бути суперкласом
final class First {/*код*/}
// наступний клас неможливий
class Second extends First{/*код*/}
```

Використання super і this. Ключове слово super використовується для виклику конструктора суперкласу і для доступу до члена суперкласу.
Наприклад:

```
/* виклик конструктора суперкласу з передачею параметрів */
super(список_параметрів);

super.i = n; /* присвоєння значення атрибуту суперкласу */
super.method(); // виклик методу суперкласу
```

Друга форма super подібна до посилання this на екземпляр класу. Третя форма специфічна для Java і забезпечує виклик перевизначеного методу. Причому, якщо в суперкласі цей метод не визначений, то здійснюватиметься пошук по ланцюжку наслідування до тих пір, поки метод не буде знайдений. Кожен екземпляр класу має неявне посилання this на себе, яке передається також і методам. Після цього можна писати this.price, хоча і необов'язково.

Наступний код показує використання this, що дає змогу побудувати одні конструктори на основі інших.

```
class Locate3D {
    private int x, y, z;
    public Locate3D(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public Locate3D() {
        this(-1, -1, -1);
    }
}
```

У цьому класі другий конструктор для завершення ініціалізації об'єкту звертається до першого конструктора. Така конструкція застосовується в разі, коли в класі є декілька конструкторів і потрібно додати конструктор по замовчуванню.

Посилання **this** використовується в методі для уточнення того, про які саме змінні x та y йде мова в кожному окремому випадку. Це потрібно для організації доступу до змінної класу у разі, якщо в методі є локальна змінна з тим же ім'ям. Інструкція this має бути єдиною в визиваючому конструкторі і бути першою виконуваною операцією.

Перевизначення методів. Здатність Java робити вибір методу, виходячи з ситуації, називається динамічним поліморфізмом. Пошук методу відбувається спочатку в даному класі, потім в суперкласі, поки метод не буде знайдений або не досягнутий Object □ суперклас для всіх класів.

Оператор instanceof діє в цій ситуації аналогічним чином. Результатом дії буде істина, якщо об'єкт є об'єктом одного з підкласів класу, на приналежність

до якого перевіряється даний об'єкт. Перевірка на приналежність об'єкту до класу Object завжди дасть істину як результат. Результат застосування цього оператора по відношенню до null завжди хибний, тому що null не можна вважати яким-небудь відповідним типом. Одночасно літерал null можна передавати в методи за посиланням на будь-який об'єктний тип і використовувати як значення, що повертається.

Статичні методи можуть бути перевизначені в підкласі, але не можуть бути поліморфними, оскільки їх виклик не зачіпає об'єкти.

Повне ім'я методу включає його ім'я, значення, що повертається, і параметри. Якщо два методи з однаковими іменами знаходяться в одному класі, списки параметрів повинні відрізнятися. Такі методи є перевантажуваними (overload). Якщо метод підкласу збігається з методом суперкласу (класу, що породжує), то метод підкласу перевизначає (overriden) метод суперкласу. Всі методи Java є віртуальними (ключове слово virtual як в C++ не використовується). Перевизначення методів є основою концепції динамічного зв’язування, що реалізовує поліморфізм. Коли перевизначений метод викликається через посилання суперкласу, Java визначає, яку версію методу викликати, беручи до уваги тип об'єкту, на який є посилання. Таким чином, тип об'єкту визначає версію методу на етапі виконання.

У наступному прикладі розглядається реалізація поліморфізму на основі динамічного зв’язування. Оскільки суперклас містить методи, перевизначені підкласами, то об'єкт суперкласу викликатиме методи різних підкласів, в залежності від того, на об'єкт якого підкласу у нього є посилання.

```
class A {
    int i, j;
    public A(int a, int b) {
        i = a;
        j = b;
    }

    void show() { // виведення i та j
        System.out.println("i та j: " + i + " " + j);
    }
}

class B extends A {
    int k;
    public B(int a, int b, int c) {
        super(a, b);
        k = c;
    }

    void show() {
    /* виведення k: перевизначений метод show() из A */
```

```
            super.show(); // виведення значень з A
            System.out.println("k: " + k);
        }
    }

    class C extends B {
        int m;
        public C(int a, int b, int c, int d) {
            super(a, b, c);
            m = d;
        }

        void show() {
        /* виведення m: перевизначений метод show() из B */
            super.show(); //виведення значень з B
            // show();/*не працює!!! метод викликатиме сам себе, що приведе до помилки під час виконання */
            System.out.println("m: " + m);
        }
    }

    public class DynDispatch {
        public static void main(String[] args) {
            A Aob;
            B Bob = new B(1, 2, 3);
            C Cob = new C(5, 6, 7, 8);
            Aob = Bob; // установка посилання на Bob
            Aob.show(); // виклик show() з B
            System.out.println();
            Aob = Cob; // установка посилання на Cob
            Aob.show(); // виклик show() з C
        }
    }
```

Результат:

```
i та j: 1 2
k: 3
i та j : 5 6
k:7
m: 8
```

При виклику show() звернення super завжди відбувається до найближчого суперкласу.

Перевантаження методів. Метод називається перевантаженим, якщо існує декілька його версій з одним і тим же ім'ям, але з різним набором

параметрів. Перевантаження може обмежуватися одним класом або декількома класами. При цьому обов'язковим є знаходження в одному ланцюжку наслідування. Слід зазначити, що статичні методи можуть перевантажуватися нестатичними і навпаки.

При виклику перевантажених методів слід уникати ситуацій, коли компілятор буде не в змозі вибрати той або інший метод. Продемонструємо це на прикладі:

```
class ClassC {}
class ClassD extends ClassC{}
public class DemoCD {
    static void show(ClassC obj1, ClassD obj2){
        System.out.println("перший метод show(ClassC, ClassD)");
    }
    static void show(ClassD obj1, ClassC obj2){
        System.out.println("другий метод show(ClassD, ClassC)");
    }
    static void show(Object obj1, Object obj2){
        System.out.println("третій метод show(Object, Object)");
    }

    public static void main(String[] args) {
        ClassC c = new ClassC();
        ClassD d = new ClassD();
        Object ob= new Object();
        show(c,d);//1_перший метод
        show(d,c);//2_другий метод
        show(c,c);//3_третій метод
        //show(d,d);// 4_помилка компіляції
        show(ob, ob);//5_третій метод
        show(c,ob);//6_третій метод
        show(ob,d);//7_третій метод
    }
}
```

У першому, другому і п'ятому випадках параметри, що передаються в метод show(), повністю збігаються з параметрами при оголошенні методу. У третьому випадку перший і другий методи не придатні для використання. Це обумовлено тим, що один з параметрів цих методів об'єкт класу ClassD. Визначення ж методу, що викликається, піднімається ланцюжком наслідування для параметрів, тому в даному випадку викликається метод з параметрами типу Object. Аналогічна ситуація виникає в шостому і сьомому випадках. У четвертому випадку обидва перших метода show() однаково придатні для виклику, як і третій. В результаті виникне помилка компіляції. Для уникнення невизначеності, слід використовувати явне перетворення типів, наприклад:

```
show(d,(ClassC)d);
```

show(d,(Object)d);

Кожний варіант викликає в результаті відповідний йому метод show().

У наступному прикладі екземпляр підкласу створюється за допомогою new. Посилання на нього передається об'єкту суперкласу. При виклику з суперкласу відповідно викликається метод підкласу.

```
class A {
    void myMethod() {
    /* private та protected використовувати не можна, оскільки метод при наслідуванні стає недоступним*/
        System.out.println("метод класу A");
    }
    void myMethod(int i) {
        System.out.println("метод класу A з аргументом");
    }
}

class B extends A {
    void myMethod(int i) {
        System.out.println("метод класу B з аргументом");
    }
}

public class C extends B {
    {
        System.out.println("клас C");
    }
    void myMethod() {
        System.out.println("метод класу C");
    }
}

public class Dispatch {
    public static void main(String[] args) {
        A obj1 = new B();
        obj1.myMethod();
        A obj2 = new C();
        obj2.myMethod();
    obj2.myMethod(10);
    }
}
```

Результати:

```
метод класу A
клас C
метод класу C
```

метод класу B з аргументом

При першому зверненні викликається метод myMethod() з класу A, як успадкований. При другому зверненні викликається метод myMethod() з класу C, як перевизначений. В останньому випадку викликається метод myMethod(int i) з класу B. Пояснюється це тим, що оскільки викликати метод без аргументів не можна, то здійснюється пошук відповідного методу по дереву наслідування.

4.7. Полімофізм і розширюваність

У наступному прикладі приведення до базового типу відбувається у виразі:

```
Stone s1 = new White();
Stone s2 = new Black();
```

Базовий клас Stone надає загальний інтерфейс для всіх спадкоємців. Породжені класи перекривають ці визначення для забезпечення унікальної поведінки. Продемонструємо поняття поліморфізму на прикладі:

```
class Stone {
    public void add() {/*порожня реалізація*/}
}
class White extends Stone {
    public void add() {
        System.out.println("додано білий камінь");
    }
}

class Black extends Stone {
    public void add() {
        System.out.println("додано чорний камінь ");
    }
}

public class StoneRandom {
    public static Stone randStone() {
        //if((int)(Math.random() * 2)==0) return new Black();
        //else return new White();// альтернативний варіант
        switch((int)(Math.random() * 2)){
        case 0:  return new Black();
        case 1:  return new White();

        default: return null;
        }
    }
    public static void main(String[] args) {
        Stone[] s = new Stone[10];
        for(int i = 0; i < s.length; i++)
```

```
            /* заповнення масиву камінням */
                s[i] = randStone();
            for(int i = 0; i < s.length; i++)
                s[i].add();// виклик поліморфного методу
        }
    }
```

Головний клас StoneRandom містить static метод randStone(). Він повертає посилання на випадково обраний об'єкт підкласу класу Stone кожного разу, коли він викликається. Приведення до базового типу вздійснюється оператором return. Він повертає посилання на Black або White. Метод main() містить масив із посилань Stone, заповнений викликами randStone(). Відомо, що є деяка множина посилань на об'єкти базового типу. Коли відбувається переміщення по цьому масиву, метод add() викликається для кожного об'єкту, обраного випадковим чином.

Якщо знадобиться додати систему, наприклад клас Green, то це приведе лише до перевизначення відповідних методів і додавання одного рядка до коду методу randStone(). Це сприяє легкому розширенню системи.

Статичні методи і поліморфізм. До статичних методів принципи поліморфізму непридатні. При використанні посилання для доступу до статичного члена, компілятор при виборі методу або поля враховує тип посилання, а не тип відповідного йому об'єкту. Наведемо приклад, що демонструє поведінку статичного методу:

```
class StaticA {
    public static void show(){
        System.out.println("метод show() з StaticA");
    }
}

class StaticB extends StaticA {}

class StaticC extends StaticB {
    public static void show(){
        System.out.println("метод show() з StaticC");
    }
}

public class StaticDemo {
    public static void main(String[] args) {
        StaticA s1 = new StaticC();
        StaticB s2 = new StaticC();
        StaticC s3 = new StaticC();
        s1.show();
        s2.show();
```

```
        s3.show();
    }
}
```

В результаті виконання даного коду буде виведено:

```
метод show() з StaticA
метод show() з StaticA
метод show() з StaticC
```

При такому способі ініціалізації об'єктів s1 і s2 метод show() буде викликаний з суперкласів StaticA і StaticB відповідно. Поліморфізм проявляється у наслідуванні методів. Для об'єкту s3 буде викликаний власний метод show(). Це обумовлено способом оголошення об'єкту. Якщо ж специфікатор static вилучити з оголошення методів, то виклики методів здійснюватимуться відповідно до принципів поліморфізму.

Клас Object. Ієрархія класів починається з класу Object. Змінна-посилання типу Object може звертатися до об'єкту будь-якого іншого класу. Крім того змінна типу Object може вказувати на будь-який масив. Це обумовлено тим, що масиви реалізуються як класи. У класі Object визначений набір методів, який успадковується всіма класами. Слід зазначити два методи: equals() і toString(). Метод equals() при порівнянні двох об'єктів повертає істину, якщо об'єкти еквівалентні, і хибу □ в іншому випадку. Якщо потрібно порівнювати об'єкти класу, що був створений програмістом, цей метод необхідно перевизначати в цьому класі. Метод toString() повертає рядок з описом об'єкту у вигляді:

```
getClass().getName() + '@' +Integer.toHexString(hashCode())
```

Метод викликається автоматично, коли об'єкт виводиться методами println(), print() і деякими іншими. При створенні класів рекомендується перевизначати метод toString(), щоб пристосувати його для створюваного типу об'єкту. Наведемо приклад, в якому перевизначимо методи equals() і toString:

```
class Point {
    protected byte b;
    protected String str;
    public Point(byte n, String s) {
        b = n;
        str = s;
    }

    public Point() {
        this((byte)0, "NoName");
    }

    public boolean equals(Object obj) {
        if (obj instanceof Point)
        return (this.b == ((Point) obj).b) && (str.equals(((Point) obj).str));
```

```
            return false;
        }

        public String toString() {
            return getClass().getName() + "@"+ " name=" + str + "  b=" + b;
        }
    }

    class PointZ extends Point{
        short s = 100;
    }
```

Метод equals() перевизначається для класу Point так, щоб отриманий об'єкт був об'єктом типу Point або одним з його спадкоємців, а також для порівняння вмісту полів b і str визиваючого та об'єкту, що передається відповідно. Метод toString() крім стандартної інформації про пакет, в якому знаходиться клас Point і самого імені класу, виводить значення полів об'єкту, що викликав цей метод, замість хеш-коду, як це робиться в класі Object.

Слід звернути увагу на виклик одного конструктора з іншого з передачею йому параметрів. Одночасно особливим є перетворення значення типу int до типу byte, оскільки дане перетворення не виконується по замовчуванню через можливу втрату інформації. Наведемо приклад, що демонструє роботу методів equals() та toString():

```
    package com.mypack;
    public class PointDemo {
        public static void main(String[] args) {
            Point p1 = new Point((byte) 1, "Петров");
            Point p2 = new Point((byte) 1, "Петров");
            PointZ p3 = new PointZ();
            Point p4 = new Point();
            System.out.println(p1.equals(p2));
            System.out.println(p1.equals(p3));
            System.out.println(p4.equals(p3));
            System.out.println(p3.equals(p4));
            System.out.println(p1.toString());
        }
    }
```

В результаті виконання даного коду буде виведено наступне

```
true
false
true
true
com.mypack.Point@ name=Петров  b=1
```

Перевизначений метод equals() дозволяє порівнювати об'єкти суперкласу з об'єктами підкласів, але лише по тих полях, які є загальними.

Збірка «сміття». Об'єкти в Java створюються динамічно за допомогою операції new. Звільнення пам'яті виконується автоматично за допомогою механізму «збірки сміття». Коли жодних посилань на об'єкт не існує, передбачається, що об'єкт більше не потрібний. В цьому випадку пам'ять, зайнята об'єктом, може бути звільнена. «Збірка сміття» відбувається під час виконання програми нерегулярно. Для її здійснення потрібно викликати метод gc(), але віртуальна машина виконує це по мірі необхідності.

Іноді перед звільненням пам'яті отрібно виконувати деякі дії, наприклад, звільнити зовнішні ресурси. Для обробки таких ситуацій використовується механізм finalization. В цьому випадку необхідно визначити метод finalize(). Віртуальна машина викликає цей метод завжди при намаганні знищити об'єкт даного класу. Всередині методу finalize() потрібно визначити дії, які мають бути виконані до знищення об'єкту. Безпосередньо перед звільненням пам'яті для об'єкту викликається метод finalize().

Метод finalize() має наступний вигляд:

```
protected void finalize(){
    // код завершення
}
```

Ключове слово protected забороняє доступ до finalize() кодам, визначеним поза цим класом. Метод finalize() викликається лише перед самою збіркою «сміття».

```
class Demo {
    private int a;
    public Demo(int a) {
        this.a = a;
    }

    protected void finalize() {
        System.out.println("об'єкт видалений, a=" + a);
    }
}

public class FinalizeDemo {
    public static void main(String[] args) {
        Demo d1 = new Demo(1);
        d1 = null;
        Demo d2 = new Demo(2);
        Object d3 = d2;         //1
        //Object d3 = new Demo(3); //2
        d2 = d1;
        System.gc();//прохання виконати «збірку сміття»
    }
}
```

В результаті виконання цього коду перед викликом методу gc() без посилання залишиться лише один об'єкт.

об'єкт видалений, a=1

Якщо закоментувати рядок 1 і зняти коментар з рядка 2, то перед виконанням gc() посилання втратять вже два об'єкти.

об'єкт видалений, a=1

об'єкт видалений, a=2

4.8. Засоби мережевого програмування Java

Java робить мережеве програмування простим завдяки наявності спеціальних засобів і класів. Розглянемо деякі види мережевих додатків. Internet-додатки включають Web-браузер, e-mail, мережеві новини, передачу файлів і telnet. Основний протокол, що використовується TCP/IP.

Додатки клієнт/сервер використовують комп'ютер, що виконує спеціальну програму, сервер. Вона надає послуги іншим програмам клієнтам. Клієнт це програма, що одержує послуги від сервера. Клієнт-серверні додатки засновані на використанні верхнього рівня протоколів. На TCP/IP базуються наступні протоколи:

- HTTP - Hypertext Transfer Protocol (WWW);
- NNTP - Network News Transfer Protocol (групи новин);
- SMTP - Simple Mail Transfer Protocol (відправка пошти);
- POP3 - Post Office Protocol (отримання пошти з сервера);
- FTP - File Transfer Protocol (протокол передачі файлів);
- TELNET - Віддалене управління комп'ютерами;

Кожен комп'ютер за протоколом TCP/IP має унікальну IP-адресу. Це 32-бітове число, що представляється як чотири числа від 0 до 255, розділених крапками. IP-адреса може бути тимчасовою і виділятися динамічно для кожного підключення або бути постійною, як для сервера. В більшості випадків при підключенні до комп'ютера замість числової адреси IP використовуються символьні імена, так звані доменні імена. Спеціальна програма DNS (Domain Name Sever) перетворює ім'я домену в числову IP-адресу. Отримати IP-адресу в програмі можна за допомогою об'єкту класу InetAddress пакета java.net.

Наведемо приклад, що виводить IP-адресу локального комп'ютера, підключеного до Internet

```
import java.net.*;
public class MyLocal {
    public static void main(String[] args){
        InetAddress myIP = null;
        try {
            myIP = InetAddress.getLocalHost();}
        catch (UnknownHostException e) {
            System.out.println("помилка доступу ->" + e);
        }
```

```
            System.out.println("Мій IP ->" + myIP);
        }
    }
```

Метод getLocalHost() класу InetAddress створює об'єкт myIP і повертає IP-адресу.

Наступний приклад демонструє, як отримати IP-адресу з імені домена за допомогою сервера імен доменів (DNS), до якого звертається метод getByName().

```
    import java.net.*;
    public class IPfromDNS {
        public static void main(String[] args){
            InetAddress bsu_iba = null;
            try {
                chiti_uch = InetAddress.getByName("www.chiti.uch.net");
            } catch (UnknownHostException e) {
                System.out.println("помилка доступу ->" + e);
            }
            System.out.println("IP-адрес ->" + chiti_uch);
        }
    }
```

Буде виведено: **www.chiti.uch.net/193.108.250.6**

Для явної ідентифікації послуг до IP-адреси приєднується номер порту через двокрапку, наприклад 217.21.43.2:3128. Номери портів від 1 до 1024 використовуються, наприклад, для запуску двох програм серверів на одному комп'ютері. Якщо порт не вказаний явним чином, броузер скористається значенням по замовчуванню: 20 - FTP-дані, 21 - FTP-управління, 23 - TELNET, 53 - DNS, 80 - HTTP, 110 - POP3, 119 - NNTP.

Адреса URL (Universal Resourse Lacator) складається з двох частин – префікса протоколу (http, ftp.) і URI (Universal Resource Identifier). URI містить Internet-адресу, необов'язковий номер порту і шлях до каталогу, що містить файл, наприклад: http://chiti.uch.net/cgi-bin/news.pl.

URI не може містити такі спеціальні символи, як пропуски, табуляції, повернення каретки. Їх можна задавати у вигляді шістнадцятиричних кодів. Наприклад, %20 позначає пропуск. Інші зарезервовані символи: & □ роздільник аргументів, ? □ передує аргументам запитів, + □ пропуск, # □ посилання всередині сторінки (ім’я_сторінки#ім’я_посилання).

Можна створити об'єкт класу URL, що вказує на ресурси в Internet. У наступному прикладі об'єкт URL використовується для доступу до HTML-файлу. Файл відображається у вікні браузера за допомогою методу showDocument().

```
    import java.applet.*;
    import java.net.*;
    import java.awt.*;
```

```
public class MyShowDocument extends Applet {
    URL chiti_uch = null;
    public void init() {
        try {
            chiti_uch =
                new URL("http://chiti.uch.net/cgi-bin/news.pl");
        } catch (MalformedURLException e) {
            System.out.println("помилка: " + e.getMessage());
        }
    }
    public boolean mouseDown(Event evt, int x, int y) {
        /* при клацанні відбувається перехід до сторінки
www.chiti.uch.net */
        getAppletContext().showDocument(chiti_uch, "_blank");
        return true;
    }
}
```

Метод showDocument() може містити параметри для відображення сторінки різними способами: "_self" виводить документ в поточний фрейм, "_blank" в нове вікно, "_top" на все вікно, "_parent" в батьківському вікні, "ім'я вікна" у вікні з вказаним ім'ям.

Продемонструємо на прикладі методи getDocumentBase() та getCodeBase(), що використовуються для отримання URL сторінки аплета та URL аплета.

```
import java.applet.*;
import java.net.*;
import java.awt.*;
public class MyDocumentBase extends Applet {
    public void init() {
        URL html = getDocumentBase();
        URL codebase = getCodeBase();
        System.out.println("URL сторінки : " + html);
        System.out.println("URL аплета : " + codebase);
    }
}
```

У наступній програмі читається вміст HTML-файла з сервера і виводиться у вікно консолі.

```
import java.net.*;
import java.io.*;
public class MyURLTest {
    public static void main(String[] args) {
        try {
            URL chiti_uch = new URL("http://www.chiti.uch.net");
            InputStreamReader isr =
            new InputStreamReader(chiti_uch.openStream());
```

```
                BufferedReader d = new BufferedReader(isr);
                String line = d.readLine();
                while (line != null) {
                    System.out.println(line);
                    line = d.readLine();
                }
            }
            catch (IOException e) {
                System.out.println("помилка: " + e.getMessage());
            }
        }
    }
```

Сокети та сокетні з'єднання. Сокети – це мережеві роз'єми, через які здійснюються двонаправлені потокові з'єднання між комп'ютерами. Сокет визначається номером порту та IP-адресою. При цьому IP-адреса використовується для ідентифікації комп'ютера, номер порту для ідентифікації процесу, що працює на комп'ютері. Коли один додаток знає сокет іншого, створюється сокетне з'єднання. Для з'єднання клієнта з сервером, він ініціалізує сокетне з'єднання. Сервер чекає, поки клієнт не зв'яжеться з ним. Перше повідомлення, що посилається клієнтом на сервер, містить сокет клієнта. Сервер у свою чергу створює сокет, який буде використовуватись для зв'язку з клієнтом, і посилає його клієнтові з першим повідомленням. Після цього встановлюється комунікаційне з'єднання.

Сокетне з'єднання з сервером створюється за допомогою об'єкту класу Socket. При цьому вказується IP-адреса сервера і номер порту (80 для HTTP). Якщо вказано ім'я домена, то Java перетворить його за допомогою DNS-сервера в IP-адресу:

```
try {
    Socket socket = new Socket("localhost", 80);
} catch (IOException e) {
    System.out.println("помилка: " + e);
}
```

Сервер чекає повідомлення клієнта і має бути запущений з вказівкою певного порту. Об'єкт класу ServerSocket створюється з вказівкою конструктору номера порту і чекає повідомлення клієнта за допомогою методу accept(), який повертає сокет клієнта:

```
Socket socket = null;
try {
    ServerSocket server = new ServerSocket(80);
    socket = server.accept();
} catch (IOException e) {
    System.out.println("помилка: " + e);
```

```
    }
```

Клієнт і сервер після встановлення сокетного з'єднання можуть отримувати дані з потоку введення і записувати дані в потік виведення за допомогою методів getInputStream() і getOutputStream() або до PrintStream для того, щоб програма могла використовувати потік як вихідні файли.

Наведемо приклад, в якому для відправлення клієнтові рядка "привіт!", сервер викликає метод getOutputStream() класу Socket. Клієнт отримує дані від сервера за допомогою методу getInputStream(). Після завершення роботи для роз'єднання клієнта і сервера сокет закривається за допомогою методу close() класу Socket.

```
import java.io.*;
import java.net.*;

public class MyServerSocket{
    public static void main(String[] args) throws Exception{
        Socket s = null;
        try {//відправка рядка клієнтові
            ServerSocket server = new ServerSocket(80);
            s = server.accept();
            PrintStream ps = new PrintStream(s.getOutputStream());
            ps.println("привіт!");
            ps.flush();
            s.close(); // розрив з'єднання
        } catch (IOException e) {
            System.out.println("помилка: " + e);
        }
    }
}
```

Наведемо приклад, в якому демонструється отримання клієнтом рядка

```
import java.io.*;
import java.net.*;
public class MyClientSocket {
    public static void main(String[] args) {
        Socket socket = null;
        try {//отримання рядка клієнтом
            socket = new Socket("ім’я_комп’ютера", 80);
            BufferedReader dis = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String msg = dis.readLine();
            System.out.println(msg);
        } catch (IOException e) {
            System.out.println("помилка: " + e);
        }
    }
```

}

Аналогічно клієнт може послати дані серверу через потік виведення за допомогою методу getOutputStream(), а сервер може отримувати дані за допомогою методу getInputStream().

Багатопоточність. Сервер повинен підтримувати багатопоточність, інакше він буде не в змозі обробляти декілька з'єднань одночасно. Сервер містить цикл, що очікує на нове клієнтське з'єднання. Кожного разу, коли клієнт просить про з'єднання, сервер створює новий потік.

Наведемо приклад, в якому створюється клас NetServerThread, що розширює клас Thread.

```
import java.net.*;
import java.io.*;
public class NetServerThread extends Thread {
    Socket socket;
    int i;
    PrintStream ps;
    public NetServerThread(Socket s) {
        socket = s;
        try {
            ps = new PrintStream(s.getOutputStream());
        } catch (IOException e) {
            System.out.println("помилка: " + e);
        }
    }
    public static void main(String[] args) {
        Socket s = null;
        try {
            ServerSocket  server = new ServerSocket(80);
            s = server.accept();
            NetServerThread nst = new NetServerThread(s);
            nst.start();
        } catch(Exception e) {
            System.out.println("помилка: " + e);
        }
    }
    public void run() {
        while (true) {
            String  msg = "повідомлення: " + i++;
            send(msg);
        }
    }
    public void send(String msg) {
        ps.println(msg);
        System.out.println(msg + "<передача>");
```

```
            ps.flush();
        }
    }
```

Сервер передає повідомлення, що посилається клієнтові. Для клієнтських додатків підтримка багатопоточності також необхідна. Наприклад, один потік чекає виконання операції введення/виведення, а інші потоки виконують свої функції.

Продемонструємо, як клієнт отримує повідомлення в потоці:

```
import java.net.*;
import java.io.*;
public class NetClientThread extends Thread {
    BufferedReader br = null;
    Socket s = null;
    public NetClientThread() {
        try {//з'єднання з кільцевою адресою
            s = new Socket("127.0.0.1", 80);
            InputStreamReader isr =
                new InputStreamReader (s.getInputStream());
            br = new BufferedReader(isr);
        } catch (IOException e) {
            System.out.println("помилка: " + e);
        }
    }
    public static void main(String[] args) {
        NetClientThread nct = new NetClientThread();
        nct.start();
    }
    public void run() {
        while (true) {
            try {
                String msg = br.readLine();
                if (msg == null) break;
                else System.out.println(msg);
            } catch (IOException e) {
                System.out.println("помилка: " + e);
            }
        }
    }
}
```

Сервер має бути ініціалізованим до того, як клієнт спробує здійснити сокетне з'єднання. При цьому може бути використана IP-адреса локального комп'ютера.

4.9. Контрольні питання

1.Що являє собою мова Java.
2.Історія виникнення мови Java.
3.Які види програм можна створювати з допомогою мови Java.
4.Порівняти між собою можливості мови Java та C++.
5.Як розміщуються об'єкти в Java.
6.Як реалізована концепція динамічного розподілу пам'яті в Java.
7.Які істотні можливості з'явились в мові Java порівняно з C++.
8.Які основні поняття мови Java існують.
9.Що таке клас в мові Java.
10. Основне призначення класу в Java.
11. Оголошення класу в мові Java.
12. Які існують специфікатори доступу до класу в мові Java.
13. Що таке аплет в мові Java.
14. Які характерні риси притамані аплету в мові Java.
15. Базові типи даних в мові Java.
16. Оператори мови Java.
17. Арифметичні оператори мови Java.
18. Булеві операції в мові Java.
19. Різновиди оператору присвоювання в мові Java.
20. Пріорітет і асоціативність операцій в мові Java.
21. Перетворення базових типів в мові Java.
22. Ідентифікатори змінних в мові Java.
23. Що відноситься до операторів управління в мові Java.
24. Умовний оператор в мові Java.
25. Циклічні оператори в мові Java.
26. Масиви в мові Java.
27. Способи оголошення масивів в мові Java.
28. Створення масивів в мові Java.
29. Яких значень набувають неініціалізовані елементи масивів.
30. Методи класу Math.
31. Що таке відношення в контексті поняття клас.
32. Чотири типи стосунків в об'єктно-орієнтованому програмуванні.
33. Що таке залежність в контексті відношень між класами.
34. Розкрийте поняття узагальнення, асоціації та реалізації в контексті відношень між класами.
35. Змінні класу.
36. Константи класу.
37. Які існують модифікатори рівня доступу змінних.
38. Що таке конструктор. Основне призначення.
39. В яких випадках застосовують конструктор по замовчуванню.

Розділ 3. Лабораторний практикум.

Лабораторна робота №1.
Створення простої програми на мові Java

Опис програми на мові Java. Початковий текст програми в Java розміщується в файлі з ім'ям створюваного класу і з розширенням .java.

Створимо клас Myfirstprogram, отже, ім'я файлу з початковим текстом буде Myfirstprogram.java. Імена програм мають бути такими, щоб ім'я відповідало призначенню і по імені можна було зрозуміти призначення програми (класу, методу, даних).

Проста програма, яка виводить привітання, виглядає таким чином:

```
/* Моя перша програма MyFirstProgram.class */
    class MyFirstProgram {
        public static void main(String args[ ]) {
            System.out.println("Привіт! Я студент групи ..... А це моя перша програма"); }}
```

Перший рядок оточений символами /* та */ – це коментар.

Другий рядок оголошує новий клас з ім'ям Myfirstprogram. Повне визначення класу міститься фігурними дужками.

Третій рядок відкриває метод main(). Всі Java-додатки починаються з цього методу. Ключове слово public – специфікація доступу. Він має бути саме public, а не private, оскільки на початку програми викликається зовнішнім методом. Ключове слово static дозволяє викликати метод main() без обов'язкового створення конкретного екземпляра класу. Ключове слово void повідомляє компілятор, що функція main() не повертає значень.

У даного методу є один параметр – args, який є масивом екземплярів рядкового класу string. Змінна args приймає в себе будь-які параметри командного рядка. В наведеній вище програмі ця інформація ігнорується. Складні програми можуть мати багато класів, але тільки один з них повинен володіти методом main(), з якого починається виконання програми.

Створення аплетів (програм для Інтернет) не передбачає використання методу main(), оскільки Web-браузер застосовує інші засоби для їх запуску.

Опис методу main() міститься внутрішніми фігурними дужками.

Четвертий рядок. System – це клас, що представляє доступ до системи, out – це вихідний потік, println() – вбудований метод, який виводить на екран текст, заданий всередині як параметр.

Для компіляції цієї програми в JDK слід виконати в командному рядку команду:

javac Myfirstprogram.java

В процесі компіляції створюється файл з тим же ім'ям і з розширенням .class, тобто Myfirstprogram.class.

Для виконання отриманої програми потрібно задати в командному рядку:

java Myfirstprogram

і отримати текст:

Привіт! Я студент групи ..... А це моя перша програма

У цій лабораторній роботі активне введення даних не використовується. Замість нього необхідно використовувати значення параметрів, що передаються класу в командному рядку при запуску, і при компільюванні значення.

При передачі методу main аргументів командного рядка використовується параметр args. Для доступу до аргументів, що передаються класу (файлу Program.class) при запуску, можна використати такий фрагмент:

```
public static void main( String[] args ) {
    String str=new String();
    for( int i=0; i<args.length; i++ ) {
        str=args[i];
        System.out.print( "args[" + i + "]:" + str + "\n\t" );
    }
}
```

Завдання на виконання. Враховуючи імена математичних функцій, приведених в лабораторній роботі №2 , знайти значення виразу:

№ варіанта	**Математичний запис**
1	$\frac{3\cos^2(x-\frac{\pi}{6})}{\frac{1}{2}+\sin y^2}$
2	$\frac{\lvert x-y\rvert}{(1+2x)^a}-e^{\sqrt{1+\omega}}$
3	$\frac{5a^{nx}}{b+c}-\sqrt{\lvert\cos x^3\rvert}$
4	$\ln\lvert a^7\rvert+\operatorname{arctg} x^2+\frac{\pi}{\sqrt{\lvert a+x\rvert}}$
5	$\sqrt[5]{\frac{(a+b)^2}{c+d}+e^{\sqrt{x+1}}}$
6	$\frac{1}{2\pi}\sqrt{\frac{2mg}{m(a\sin\alpha+b\cos\alpha)}}$
7	$\frac{1}{4}\left(\frac{1+x^2}{1-x}+\frac{1}{2}\operatorname{tg} x\right)$
8	$a+\sqrt{b}+\ln\lvert x^3\rvert+e^{\sqrt[3]{c}}$
9	$\lvert\sin\alpha\rvert\cos\frac{\alpha}{2}+\sqrt{a^2+b^2}$
10	$\sqrt[5]{x}(\operatorname{arctg} z+\cos^2 y)$

Лабораторна робота №2.

Основні конструкції мови Java. Типи даних

Java є суто типізованою мовою. Це означає, що перш ніж використовувати які-небудь змінні, потрібно їх означити відповідним типом. До основних конструкцій мови Java відносяться оператори умови та циклів. Детально інформація про них, а також типи даних наведена в розділі 3. Там же розглянуті і питання по роботі з масивами.

Розглянемо простий приклад роботи з двовимірним масивом, що демонструє введення його елементів за допомогою генератора випадкових чисел, а також їх виведення.

```
public class mas1 {
static int n[][]=new int[5][5];
public void main(String args[]){
    for(int i=0;i<5;i++)
    for(int j=0;j<5;j++)
    n[i][j]=(int) (Math.random()*6);
System.out.println("Masiv \n");
    for(int i=0;i<5;i++){
        System.out.print("\n");
    for(int j=0;j<5;j++)
        System.out.print(n[i][j]);}
    }
}
```

Розглянемо інший приклад, в якому знаходимо суму елементів одновимірного масиву.

```
public class mas2{
    int a[];
    a=new int[100];
    void print_(){
        System.out.println("Massiv \n");
        for(int i=0;i<100;i++){
            System.out.println("a["+i+"]"+a[i]);
    }
    void sum(){
        int sum;
        for(int i=0;i<100;i++)
            s+=a[i];
        System.out.println("");
        System.out.println("Sum raven"+sum);
    }
    public static void main(String args[]){
        for(int i=0;i<100;i++)
```

```
                a[i]=(int) (Math.random()*6);
            print_();
            sum();
        }
    }
```

Ці приклади, по-перше, ілюструють основну структуру коду мови Java (про це йшлося вище).

По-друге, демонструється робота з масивом в Java. Як зазначалося раніше, масив створюється динамічно з допомогою оператора new. Оскільки нумерація елементів в Java розпочинається з нуля, це знайшло відображення в циклі for, що має вигляд:

```
for(int i=0;i<100;i++){
…
}
```

Завдання: Створити програму на Java із застосуванням базових конструкцій мови, яка виконує перетворення даних відповідно до варіанту завдання.

1. Обчисліть f= 10! трьома різними способами (з використанням операторів циклу while/for/do-while)
2. Напишіть програму, що сортує масив цілих чисел за збільшенням.
3. Напишіть програму, що сортує масив цілих чисел за зменшенням.
4. Знайдіть найменший елемент в двовимірному масиві та номер рядка і стовпця, в якому вони розташовані.
5. Знайдіть суму всіх непарних чисел масиву цілих чисел.
6. Знайдіть суму найбільшого та найменшого елементів двовимірного масиву.
7. Знайдіть індекс елемента, що має найменше відхилення від найбільшого елемента двовимірного масива.
8. Знайдіть сумарне відхилення по рядках кожного елементу від найбільшого його елементу.
9. Елементи одновимірного масиву циклічно змістити на дві позиції вліво.
10. Елементи одновимірного масиву циклічно змістити на п'ять позиції вправо.

Лабораторна робота №3.
Введення в класи Java

Класи. Мова Java є об'єктно-орієнтованою. В її основі лежить поняття класу. Він поєднує дані та функції (методи), які обробляють ці дані. Якщо в традиційному процедурному програмуванні дані і функції відокремлені один від одного, то в об'єктно-орієнтованому вони об'єднуються під загальним "дахом" з назвою клас. Клас можна визначити як тип об'єкту. Кожен об'єкт належить деякому класу, що визначає сукупність даних і методів, характерних для цього об'єкту.

Клас оголошується за допомогою ключового слова class.

Синтаксис оголошення класу:

```
class ім'я_класа{
    функції і змінні класу
}
```

Конструктори об'єктів. Для кожного створюваного об'єкту потрібна ініціалізація. Зазвичай код ініціалізації розміщують в тілі функції-конструктора класу. Функція-конструктор є методом класу, який має те ж ім'я, що і сам клас (ім'я функції-конструктора збігається з ім'ям класу). Наприклад, якщо використовуємо клас Car, функція-конструктор називатиметься теж Car. Конструктор класу викликається кожного разу при створенні об'єкту цього класу. Функція-конструктор не може повертати значень і при цьому в її визначенні не пишеться ключове слово void. Тип значення, що повертається цією функцією, просто не потрібно вказувати.

Абстрактні класи. Абстрактним класом в Java називається клас, який містить абстрактні методи – методи, перед оголошенням яких указується ключове слово abstract і визначення яких дається тільки в підкласах класу .

```
public abstract class Road{
    abstract void Drawroadt();
    abstract int Findmedian(int []x);
    //...Other methods.
}
public class Superhighway extends Road{
    void Drawroad(){
    //...
    }
    int Findmedian(int []x){
    //...
    }
}
```

В мові Java передбачений єдиний спосіб розподілу пам'яті – оператором new. Відносно виділення блоків пам'яті багато в чому діють ті ж правила, що і в C++. Але є і виключення: у Java є можливість динамічного задання імені створюваного класу, наприклад:

Classvar = new ("Class" + "Name");

Класи та їх окремі члени можуть бути статичними. В цьому випадку вони позначаються ключовим словом static. Перевага статичних членів полягає в тому, що вони стають такими, що розділяються між всіма класами-нащадками і екземплярами класу. Це означає, що, посилаючись на декілька успадкованих класів або декілька екземплярів, насправді ви посилаєтеся на один і той же член класу, розташований в одній і тій же ділянці пам'яті. На додаток до стандартних статичних визначень в Java є ініціалізації – блоки коду, помічені ключовим словом static. Їх завдання – ініціалізація статичних змінних. При завантаженні класу спочатку виконуються блоки ініціалізації, а вже потім починається привласнення значень простим змінним, які ініціалізувалися в порядку їх опису. Це ж відноситься і до блоків ініціалізації. У прикладі, показаному нижче, змінні ініціалізуються в такому порядку: xxx, yyy.

```
class StaticClass{
    short zzz = 10;
    static int xxx;
    static float yyy;
    static{
        xxx = 12345;
        yyy = 3.1415;
    }
}
```

Далі виконується блок ініціалізації, і вже потім проводиться ініціалізація змінної zzz.

Завдання.

1. Створити на Java ієрархію класів Графік – точка, коло, чотирикутник, еліпс, задній фон. Клас графік повинен містити абстрактний метод draw(). Решта класів повинна його реалізовувати, відображаючи атрибути об'єктів у вигляді рядка, наприклад:
Точка: x=10, y=20
Чотирикутник: x=2, y=5, w=3, h=4 (w та h – ширина та висота відповідно). Описати в дочірніх класах всі необхідні атрибути, конструктори і методи (об'єкти повинні мати координати і колір; у класу Background є колір і назва текстури). Обов'язкове застосування інкапсуляції для приховування атрибутів.
2. Створити ієрархію об’єктів за наступною схемою: транспортні засоби – автомобіль, потяг, літак.
3. Створити ієрархію класів чотирикутник – прямокутник, квадрат та трапеція.
4. Створити ієрархію класів за наступною схемою: Преса – газета, журнал та електронне видання.
5. Продемонструвати ієрархію класів. Вивести на екран рядок з поточним часом і датою в наступному форматі: рік, місяць, день тижня, число, час. Для цього створіть два класи: Clock (для отримання поточного часу і виведення його на екран) і Timeformatter (для формування рядка з часом і датою наведеного вище

формату за допомогою методів класу Date). У класі Clock організуйте отримання поточної дати за допомогою конструктора класу Date без параметрів.

6. Оголосити клас «Вектор» для зберігання посилань на об'єкти. Клас повинен мати такі поля: масив посилань, який може рости; кількість посилань в масиві. Клас повинен мати такі методи: 1)очистити весь масив; 2) додати посилання в масив; 3) отримати і-й елемент; 4) видалити і-й елемент; 5) вивести значення масиву на екран.

7. Продемонструвати ієрархію класів. Вивести на екран рядок з поточним часом і датою, в наступному форматі: рік, місяць, день тижня, число, час. Для цього створіть два класи: Clock (для отримання поточного часу і виведення його на екран) і Timeformatter (для формування рядка з часом і датою наведеного вище формату за допомогою методів класу Date). За допомогою методу Thread.sleep() задайте проміжок часу, через який ви будете читати і виводити поточну дату. Не забудьте, що метод Thread.sleep породжує виключення Ioexception, яке обов'язково хоч якось повинно бути оброблено.

8. Розробіть програму, яка виводить на екран таймер на три хвилини з посекундним зворотним відліком часу. Скористайтеся методами класу java.util.Date.

9. Напишіть програму, яка реалізує додавання, віднімання, множення, ділення комплексних чисел. Для цього створіть клас Complex, в якому реалізуйте відповідні методи.

10. Оголосити клас «Матриця» з полем – двомірним масивом дійсних чисел і методами 1) складання з іншою матрицею; 2) множення на число; 3) множення на іншу матрицю; 4) транспонування; 5) вивід на друк.

Лабораторна робота №4.
Класи в Java. Інкапсуляція. Наслідування. Поліморфізм.

Об'єктно-орієнтоване програмування засноване на трьох концепціях:

ікапсуляція – об'єднання в об'єкті даних і функцій для обробки цих даних;

наслідування – механізм, за допомогою якого один об'єкт може наслідувати властивості (дані і функції) іншого об'єкту і додавати до них риси, характерні для нього;

поліморфізм – це властивість, яка дозволяє одне і те ж ім'я використовувати для вирішення різних завдань.

Поєднання наслідування і поліморфізму дозволяє легко створити серію подібних, але разом з тим унікальних об'єктів.

Завдяки наслідуванню такі об'єкти мають багато схожих характеристик, а завдяки поліморфізму, кожен з них може володіти власною поведінкою.

Інкапсуляція. Інкапсуляція полягає в об'єднанні даних і функцій, які обробляють ці дані, в єдине ціле, тобто в клас.

Є можливість задати обмеження для об'єктів інших класів на доступ до функцій (методів) і змінних (даних) цього класу. Для цього існують так звані модифікатори доступу public (глобальний), protected (захищений), private (локальний), final(кінцевий). Модифікатори доступу забезпечують захист даних і методів класу від зовнішнього втручання і неправильного використання.

Змінні та методи мають бути "видимі" іншим об'єктам тільки у тому випадку, коли правила доступу дозволяють це. Гарний стиль конструювання вимагає виконання правила, за яким має бути видимим тільки те, що необхідне, і не більше.

Якщо модифікатор перед оголошенням властивості (змінна/метод класу) не вказується, то властивість буде "видима" тільки класам в тому ж пакеті.

Модифікатор **public в**казує, що властивості (методи/змінні) даного класу будуть видимі об'єктам інших класів. Це так звана широко відкрита видимість.

Модифікатор **private в**казує, що доступ до властивостей і методів класу може здійснюватися тільки за допомогою методів об'єктів даного класу.

Модифікатор **protected** указує, що дані і методи даного класу можуть бути доступні як з об'єктів даного класу, так з об'єктів підкласів даного класу і класів з того ж пакету.

Якщо функція класу немає public, то аплет не може викликати дану функцію використовуючи оператор ".". Єдиний спосіб вашого аплета отримати доступ до членів з міткою protected полягає у використанні інтерфейсних функцій.

Якщо перед визначенням змінної помістити модифікатор final, то це фактично перетворює її на константу. Метод, помічений як final, в підкласах не можна перевизначити.

Використовувати метод final можна з різною метою. По-перше, для підвищення безпеки. Можна дозволити створювати підклас і наслідувати в

ньому всі властивості суперкласу, забороняючи, проте, застосування власних версій методів при наслідуванні. По-друге, для зменшення часу виконання програми. Якщо метод помічений як final, інших його версій не існує. Отже, в процесі виконання програми немає необхідності звертатися до процедур динамічного зв'язування і чекати на результати їх роботи. Компілятор може оптимізувати код програми, тому модифікатор final потрібно застосовувати якомога частіше.

Наслідування. Наслідування є однією з наймогутніших концепцій об'єктно-орієнтованого програмування.

Властивості класу передаються "за спадком". Скажімо, можна створити клас "автомобілі", що містить методи і дані, характерні для всіх автомобілів, і на його основі інший клас – "легкові автомобілі", що успадковує всі властивості батьківського класу "автомобілі" і що володіє деякими іншими додатковими власними властивостями.

Наслідування полегшує роботу програміста, дозволяючи записувати загальні частини програми тільки один раз (у батьківському класі).

Така стратегія, наприклад, спрощує передачу коду програми по мережі. У Java аплет – це об'єкт деякого класу, що успадковує властивості загального класу Applet. Велика частина програмного забезпечення зберігається локально в браузері, що істотно скорочує обсяг коду, що пересилається по мережі.

Ієрархія класів. Щоб скористатися наслідуванням мови Java, треба оголосити новий клас, який є розширенням іншого класу. Новий клас називається підкласом, а початковий – суперкласом.

Підклас успадковує всі властивості батьківського класу (суперкласу). Підклас набуває всіх властивостей суперкласу, але, крім того, може володіти додатковими властивостями. У класа може бути тільки один суперклас. Така підсхема носить назву одиничного наслідування.

Для того, щоб оголосити клас A підкласом суперкласу B використовується ключове слово extends.

```
class A extends B {    ...    }
```

Якщо суперклас не вказується явно при оголошенні класу за допомогою ключового слова extends, роль суперкласу виконує клас Object. Тобто фактично всі класи в Java є підкласами класу Object.

Оголошення класу може містити модифікатор public.Цей модифікатор робить клас доступним решті всіх класів.

Якщо ви не вказуєте модифікатор доступу public,оголошення вважається дружнім. Всі оголошення такого типу є загальнодоступними в межах свого модуля компіляції (файл початкового коду) або пакету. Класи не можуть бути оголошені як private або protected.

Поліморфізм. Термін поліморфізм походить від двох грецьких слів: полі, що означає багато, і морф, що означає форма. Отже, поліморфізм має відношення до багатьох форм чогось.

Поліморфізм надає можливість похідним класам (підкласам) міняти те, що роблять методи, успадковані ними від базових класів (суперкласів). Іншим видом

поліморфізму є перезавантаження методів – використання одного і того ж імені для завдання загальних для класу дій. Виконання кожної конкретної дії при цьому визначається типом даних. Тобто клас може містити декілька версій методу, які відрізняються кількістю та/або типом аргументів.

Наприклад, для мови С, в якій поліморфізм підтримується недостатньо, знаходження абсолютної величини числа вимагає різних функцій: abs(), labs() і fabs(). Ці функції підраховують і повертають абсолютну величину цілих, довгих цілих і чисел з плаваючою крапкою відповідно. У Java це все може бути виконано за допомогою однієї функції abs().

Приклад визначення двох методів з однаковими іменами:

```
class Car{
    Tire leftFront=new Tire();
    Tire rightFront=new Tire();
    Tire leftRear=new Tire();
    Tire rightRear=new Tire();
        ...
    void SwapTires (Tire a, Tire b){
        Tire temp;
        temp=a;
        a=b;
        b=temp;
    }
    public void RotateTires( ){
        SwapTires(this.leftFront, this.rightRear);
        SwapTires(this.leftRear, this.rightFront);
    }
    public void RotateTires( Tire t1, Tire t2, Tire t3, Tire t4,){
        SwapTires(t1, t2);
        SwapTires(t2, t3);
        SwapTores(t3,t4);
    }
}
```

У прикладі створено дві версії методу Rotatetires. Якщо відбувається виклик Rotatetires() без параметрів, використовується перша версія, яка переставляє шини по діагоналі. Для виклику другої версії методу слід написати:

Rotatetires(this.leftFront,this.rightRear, this.leftRear, this.rightFront);

При розробці програм може знадобитися додавання в підклас додаткових властивостей, яких не було в суперкласі. Наприклад, в базовому класі визначена складна підпрограма, яку потрібно перенести в підклас з невеликими доповненнями. Перевизначення методу в підкласі вимагає дублювання значного об'єму коду. Для того, щоб не переписувати код методу суперкласу можна використовувати ключове слово super. Ключове слово super вказує компілятору, що необхідно викликати метод суперкласу.

Приклад:

```
        class Transportation{
            int MaxLoad;
            int PeopleCapacity;
            void Horn () {
                System.out.println("Honk.");
            }
}
        class Sedan extends Transportation {
            int StereoWattage;
            SparkPlug p1,p2,p3,p4,p5,p6;
            OilFilter o1;
            AirFilter a1;
            Sedan(int StereoSize,Int doors){
            E
            }
            void TuneUp(){
                p1=new SharkPlug();
                p2=new SharkPlug();
                p3=new SharkPlug();
                p4=new SharkPlug();
                p5=new SharkPlug();
                p6=new SharkPlug();
                o1=new OilFilter();
                a1=new AirFilter();
            }
}
        class LowRider extends Sedan{
            AirShocks s1,s2,s3,s4;
            void TuneUp(){
                super.TuneUp();
                s1=new AirShocks();
                s2=new AirShocks();
                s3=new AirShocks();
                s4=new AirShocks();
            }
}
```

Оператор new служить для створення нового об'єкту. Оператор складається з трьох частин: ключового слова new імені класу об'єкту і набору аргументів, що передаються в конструктор.

Завдання. Розробити ієрархію класів Java, відповідну варіанту завдання. Обов'язково використовувати абстракцію, наслідування, інкапсуляцію, public-, private- і protected-методи. Розробити тестову програму, що маніпулює об'єктами цих класів (створення, видалення, пошук та інше). Поля класів визначити

самостійно (мінімум 4-5 полів на клас). Тестовий додаток має бути консольного типу.
Окрім вказаних у варіанті завдання класів реалізувати клас Manager, що реалізує всю логіку по роботі з об'єктами даних класів.

N п/п	**Опис**	**Класи**	**Пояснення**
1	«Файлова система»	Диск, Каталог, Файл	Імітація ієрархічної файлової системи для роботи з бінарними та текстовими файлами. Стандартні операції роботи з каталогами і файлами: створення, видалення, копіювання. Доступ послідовний і випадковий. Пошук файлу по імені і по вмісту.
2	«Комп’ютерна мережа»	Пристрій, Порт, Кабель	Спрощена модель телекомунікаційної мережі. Операції: створення та видалення пристроїв, створення і видалення портів на пристроях, прокладка кабелів між портами пристроїв. Пошук найкоротшого шляху між двома заданими пристроями.
3	«Словник»	Слово, Стаття, Посилан-ня	На одне слово може припадати декілька статей, і на одну статтю – декілька слів. Реалізувати додавання, редагування, видалення статей, прив'язку статті до слова. Реалізувати пошук по словнику з урахуванням помилок введення слова (пропущена/зайва/спотворена літера).
4	«Фотоальбом»	Сторінка, Фото, Автор	Система для впорядкованого зберігання робіт групи фотографів. Реалізувати маніпуляції з фото і їх авторами. Пошук фото по імені автора, категорії, розміру, даті створення/модифікації. Передбачити видачу звітів частоті поповнення альбому тим або іншим автором за вказаний часовий період.

5	«Форум»	Тема, Повідом-лення Учасник	Система для спілкування групи користувачів. Форум містить групу Тем, в яких Учасники можуть розіщувати свої повідомлення. На будь-яке повідомлення можна написати відповідь. Виділяється привілейований учасник Адміністратор, який може створювати нові теми, редагувати або видаляти будь-які повідомлення та учасників. Рядовим учасникам дозволяється редагувати та видаляти тільки власні повідомлення.
6	«Авіаагенство»	Місто, Рейс, Білет	Ведення списку населених пунктів, сполучених авіалініями, списку рейсів між містами з інформацією про дату вильоту/прильоту, наявність місць в трьох класах. Видача квитків на певний рейс і місця в заданому класі. Підбір маршруту з пересадками по заданих відправній та кінцевій точках маршруту.
7	«Бібліотека»	Книга, Читач, Обліковий запис	Ведення списків книг і читачів, "видача" книг. Обліковий запис відображає стан книги (знаходиться у читача, коли взята, коли має бути повернена). Після повернення книги обліковий запис не знищується, змінюється лише її статус. Пошук книги по жанру, авторові, назві. Реалізувати побудову звітів за вказаний часовий період: читачі заданої книги; книги заданого читача; рейтинг популярності заданої книги.
8	«Магазин»	Категорія, Товар, Замовлен-ня	Торгівля різними категоріями товарів. Пошук товару по категорії, виробнику, вартості. Оформлення замовлення на товар. Відстежування стану замовлення. Надання звітів по продажах в заданий період часу, рейтинг популярності того або іншого товару.

9	«Вокзал»	Місто, Рейс, Квиток	Ведення списку населених пунктів, сполучених залізничними коліями, списку рейсів з інформацією про дату виїзду, наявність місць в трьох класах. Видача квитків на певний рейс в заданому класі. Підбір маршруту з пересадками по заданих відправній і кінцевій точці маршруту.
10	«Деканат»	Студент, Дисципліна, Оцінка	Ведення успішності студентів, списку студентів, дисциплін, що вивчаються ними.

Лабораторна робота №5.
Графіка в Java

Клас java.awt.Graphics. У класі Graphics з пакету java.awt існує багато різноманітних методів для роботи з графічними примітивами. Засоби мови Java дозволяють змінювати колір і створювати зображення ліній, прямокутників, еліпсів і многокутників. При роботі з графікою в Java слід враховувати особливості системи координат. Верхній лівий кут вікна має координати (0,0), вісь x направлена вправо, а вісь y – вниз. Така система координат використовується в багатьох комп'ютерних системах, але не відповідає математичній системі координат, де вісь y направлена вгору.

В більшості випадків функції **paint** аплета передається об'єкт класу **Graphics**. Для побудови зображень необхідно звертатися до його методів. Можна також створити власний об'єкт цього класу, наприклад, для прискорення виведення зображень при відтворенні складної анімації. Розглянемо основні методи класу Graphics.

drawLine(). Для зображення лінії в метод drawline необхідно передати чотири параметри: координати (x1, y1) початку і (x2, y2) кінця лінії. В наступному прикладі зображені десять паралельних ліній.

Приклад:

```
import java.awt.Graphics;
import Java.applet.*;
public class DrawLines extends Applet{
    public void paint (Graphics g){
        for(int i=0; i<100; i+=10)
            g.drawLine (i, i, i+10, i) ;
    }
}
```

drawRect() і fillRect(). Можна намалювати і зафарбувати прямокутник. Кути прямокутника можуть бути закруглені, а зображення може бути псевдотривимірним.

Приклади:

```
// Малювання прямокутника з верхнім лівим кутом в
// (left, top), і нижнім правим кутом в
// (left+width, top+height).
g.drawRect(left, top, width, height);

//Малювання такого ж прямокутника, як в drawrect,
//але зафарбованого кольором лінії контура.
g.fillRect(left, top, width, height);
```

```
// Малювання звичайного і зафарбованого прямокутників з
// закругленими кутами.
// Радіуси закруглення по горизонталі і вертикалі
// визначаються значеннями horizontal-radius
// і vertical-radius.
g.drawRoundRect(left, top, width, height,horizontal-radius,vertical-radius);
g.fillRoundRect(left, top, width, height,horizontal-radius, vertical-radius);

// Малювання прямокутника із затінюванням по краях для
// створення ефекту тривимірного зображення.
// Якщо значення polarity встановлене в true, то
// прямокутник "виступає" з екрану,
// а при false – "втиснуте" в екран.
g.draw3DRect (left, top, width, height, polarity) ;
```

Функції малювання прямокутників містять функції малювання ліній і дуг, а також функції закрашення (заповнення) об'єктів. Кути закругленого прямокутника формуються з дуг еліпса. Спочатку еліпс розділяється на чотири частини, частини "розтягуються" і далі з'єднуються прямими лініями. Кожен кут закругленого прямокутника є чвертю еліпса, пропорції якого визначаються шириною і висотою прямокутника.

Приклад аплета для малювання множини прямокутників:

```
import java.awt.Graphics;
import java.applet.*;
public class drawRects extends Applet {
public void paint (Graphics g){
    for (int i=0; i<140; i+=20)
        g.drawRect(i, 5, 15, 45);
    for (int i=0; i<140; i+=20)
        g.fillRect(i, 55, 15, 45);
    for (int i=0; i<140; i+=20)
        g .drawRoundRect(i, 100, 20, 45, 10, 15);
    for (int i=0; i<100; i+=20)
        g.draw3DRect(i, 150, 15, 35, true);
    }
}
```

drawPolygon() та fillPolygon(). Многокутники будуються по двомірному масиву координат їх вершин. Якщо перший елемент масиву не збігається з останнім, багатокутник буде розімкнений. Метод fillpolygon будує закрашений багатокутник. Якщо багатокутник розімкнений, то проводиться автоматичне з'єднання першої і останньої крапок.

У класі Polygon є конструктор Polygon() без параметрів, за допомогою якого створюється порожній багатокутник. Координати вершин можна додати пізніше методом addpoint(x,y). Інший конструктор, з двома масивами як

параметрами, створює готовий многокутник. Потім об'єкт класу Polygon можна намалювати на екрані.

Примітка. Клас Polygon не міститься в класі Graphics і має бути завантажений окремо. З цієї причини в прикладі імпортується весь пакет java.awt.

Приклад створення многокутників:

```
import java.awt.*;
import java.applet.*;
public class DrawPoligons extends Applet {
    int []X=new int[100];
    int []Y=new int[100];
    int count; Polygon p;
    void makePoly(int offX, int offY){
        X[0]=offX+14;
        X[1]=offX+38;
        X[2]=offX+22;
        X[3]=offX+19;
        X[4]=offX+2;
        Y[0]=offY+2;
        Y[1]=offY+22;
        Y[2]=offY+23;
        Y[3]=offY+39;
        Y[4]=offY+12;
        count=4;
    }
    public void paint(Graphics g){
        makePoly (10,10);
        g.drawPolygon(X, Y, count);
        makePoly(10,100);
        g.fillPolygon(X, Y, count);
        p=new Polygon( );
        p.addPoint(100,120);
        p.addPoint(140,142);
        p.addPoint(130,162);
        p.addPoint(142,122);
        p.addPoint(90,92);
        g. fillPolygon(p);
    }
}
```

drawOval() та drawArc(). Метод drawoval малює еліпс, а метод drawarc – дугу еліпса. Зафарбовування проводиться відповідно методами filloval і flllarc. Параметрами виступають координати найменшого описаного прямокутника. Еліпс визначається чотирма параметрами: двома координатами вершини описаного прямокутника, його шириною і висотою.

Для **drawarc** існують два додаткові параметри, що визначають початок і довжину дуги в кутових градусах. Довжина дуги відлічується по напряму годинникової стрілки від полудня, так що 90 градусів відповідають цифрі 3 на циферблаті.

Приклад:

```
import java.awt.Graphics;
import java.applet.*;
public class DrawOvals extends Applet{
        public void paint(Graphics g){
            for(int i=0; i<140; i+=20)
                g.drawOval(i,5,15,45);
            for(int i=0; i<140; i+=20)
                g.fillOval(i,55,15,45);
            for(int i=0; i<140; i+=20)
                g.drawArc(i,100,20,45, i+10, i+100);
            for(int i=0; i<140; i+=20)
                g.fillArc(i, 150, 20, 45, i+10, i+100);
        }
}
```

drawString(). Перед виведенням на екран рядка тексту доцільно створити об'єкт Font, що визначає ім'я, стиль і розмір шрифту для цього рядка. Наприклад:

```
Font k =new Font("Timesroman", Font.PLAIN, 24);
g.setFont(k);
g.drawString("Рядок тексту!",10,20);
```

У першому рядку застосований конструктор класу Font для створення об'єкту k, який потім передається в об'єкт класу Graphics. Першим параметром конструктора є ім'я шрифту. Існують п'ять "універсальних" шрифтів, доступних в будь-якому браузері: Courier, Dialog, Helvetica, Symbol і Timesroman.

Список шрифтів конкретного браузера можна отримати методом getfontlist(), що міститься в класі java.awt.Toolkit.

З кожним шрифтом можна використовувати такі стилі: Font.BOLD (напівжирний), Font.ITALIC (курсив) або обидва стилі одночасно з допомогою команди k = new Font ("Courier", Font.ITALIC+Font.BOLD, 12). Константи стилів визначені в класі Font, фактично є цілими числами.

Метод drawstring мови Java не має багатьох необхідних можливостей. При виведенні рядка на екран ігноруються символи перекладу рядка і повернення каретки, отже, необхідні додаткові розрахунки при розміщенні рядків на екрані. Тому існують спеціальні методи для виведення декількох рядків. Об'єкти класу Fontmetrics (що повертаються функцією getfontmetrics класу Graphics) мають методи, за допомогою яких можна дізнатися розміри символів, міжрядковий інтервал і висоту рядка. Верхня лінія рядка визначається з урахуванням висоти прописних (великих) букв. Нижня лінія рядка проходить по нижній точці символів типу "g". Міжрядковий інтервал задає відстань між базовими лініями

двох сусідніх рядків. А загальна висота рядка вимірюється від нижньої точки символів типу "y" до верхньої точки прописних букв (наприклад, "M").

Декілька методів з класу FontMetrics:

public int getleading() – повертає значення міжрядкового інтервалу;

public int getascent() – повертає відстань між базовою і верхньою лініями рядка;

public int getdescent() – повертає відстань між базовою і нижньою лініями рядка. (Нижня лінія визначається нижньою точкою символів типу "g");

public int getheight() – повертає суму загальної висоти рядка і міжрядкового інтервалу;

public int charwidth(int ch) – повертає ширину символу;

public int stringwidth(String str) – повертає ширину рядка (сума ширини всіх символів);

public int charswidth(char data [], int off, int len) – повертає ширину масиву символів (суму ширини len символів масиву, починаючи з off);

public int[] getwidths() – повертає ширину кожного з 256 базових символів.

Функції обчислення ширини особливо корисні при розробці текстових редакторів або для ефектного форматування рядка на екрані.

Приклад форматування тексту:

```
import java.awt.*;
import java.applet.*;

public class g5 extends Applet{
    public void paint(Graphics g){
        int w = 200;
        int h = 150;
        g.drawRect(10,10,w,h);
        Font f=new Font("Courier", Font.PLAIN, 18);
        g.setFont(f);
        FontMetrics fm = g.getFontMetrics();
        g.drawString("Courier", 11, 10+fm.getHeight());
        f=new Font("TimesRoman", Font.ITALIC,12);
        g.setFont(f);
        fm=g.getFontMetrics();
        g.drawString("TimesRoman", w-fm.stringWidth("TimesRoman"),
10+fm.getHeight());
        f=new Font("Symbol", Font.PLAIN,18);
        g.setFont(f);
        fm=g.getFontMetrics();
        g.drawString("Symbol",10,h);
        f=new Font("Dialog", Font.PLAIN,18);
        g.setFont(f);
        fm=g.getFontMetrics();
```

```
            g.drawString("Dialog",w-fm.stringWidth("Dialog"),h);
        }
}
```

Задання кольору. В мові Java для роботи з кольором застосовується базова модель RGB, в якій кожен компонент кольору (червоний, зелений і синій) задається цілим числом в діапазоні від 0 до 255. Будь-який колір визначається трьома числами, наприклад: білий – (255,255,255), червоний – (255,0,0), зелений – (0,255,0), чорний, – (0,0,0).

Клас java.awt.Color є головним класом для роботи з кольором. Проглянувши початкові тексти цього класу можна побачити, що в мові Java підтримується ряд стандартних кольорів. Їх коди приведені в розділі 2.

У класі Color існують три конструктори. Аргументами першого з них є три цілі числа в діапазоні від 0 до 255, що визначають інтенсивність червоного, зеленого та синього кольорів.

```
Color lineColor=new Color(140,23,190);
```

Аргументи другого конструктора – три числа з плаваючою крапкою в діапазоні від 0.0 до 1.0, які перетворюються до цілих чисел з діапазону 0–255.

Аргумент третього конструктора – 32-розрядне число, в якому синьому кольору відповідають біти з 0 по 7, зеленому – з 8 по 15, а червоному з 16 по 23.

У класі **Color** присутні різноманітні методи роботи з кольором. Наприклад, для екстракції окремих складових кольору використовується методи **getred(), getblue()** і **getgreen()**, а метод **getrgb()** комбінує кольори в одне ціле число.

Клас **Graphics** використовує об'єкти класу **Color** для визначення кольору об'єкту, що відображається. Для виводу в кольорі об'єктів, що створюються такими методами, як **drawoval()** або **drawrect()**, слід використовувати метод **setcolor()**. Наприклад, для того, щоб встановити поточний колір для малювання червоним:

```
g.setColor(Color.red);
```

Поточний колір можна отримати, скориставшись функцією **getcolor()**.

Фон вікна аплета за замовчуванням сірий. Для його зміни використовується метод **setbackground**(), визначений в класі **Component**(підкласом якого є **Applet**). Існує парний йому метод **getbackground**(), який отримує поточний фон вікна аплета. Метод **setforeground**() діє на всі об'єкти аплета, встановлюючи для них заданий колір.

Завдання. Використовуючи графічні примітиви, вивести на web-сторінку анімоване зображення:

1.Тріод, використовуючи графічні примітиви.

2.Транзистор, використовуючи графічні примітиви.

3.Будиночок в лісі вночі, використовуючи графічні примітиви.

4.Резистор, використовуючи графічні примітиви.

5.Анімованого сніговика, використовуючи графічні примітиви.

6.Лампочку, яка спалахує та потухає, використовуючи графічні примітиви.

7.Діод, використовуючи графічні примітиви.

8.Послідовність написів "Hello, World" на графічному примітиві, з кольором заливки відмінним від кольору контура. Кожен напис повинен відрізнятися від попереднього шрифтом, кольором і розміром символів. Встановіть затримку між виводом написів в 1 секунду.

9.Годинник, що йде, з циферблатом і двома стрілками, використовуючи графічні примітиви.

10. Чашку з димлячою кавою, використовуючи графічні примітиви.

Лабораторна робота №6.
Розробка і застосування аплетів

Java-програми, які можуть бути вбудовані в Web-сторінки, називаються аплетами Java.

Розробимо програму, розглянуту в лабораторній роботі №1, у вигляді аплета.

Аплети наслідують властивості базового класу Applet, що дозволяє їм без збільшення обсягу забезпечити функціональність повномасштабних застосувань. Інформація з java.applet.* розміщена локально разом з браузером, що дозволяє прискорити процес завантаження аплета.

```
import java.applet.*;
public class Hello extends Applet{system.out.println("Привіт, студент");
```

При запуску цього аплета ви не побачите повідомлення "Hello world" в області, що відведена для аплета на HTML-сторінці. Це обумовлено тим, що функція System.out.println() виводить повідомлення у вікно командного сеансу, а не у вікно браузера.

Для того, щоб побачити те, що виводить функція System.out.println() у MS Internet Explorer, потрібно запустити Java Console (Вікно мови Java) в секції меню View (Вид). Якщо там немає такого меню, то потрібно встановити прапорець Java Console Enable в Tools| Internet Options|Advanced та перезавантажити машину.

Для виведення тексту в аплеті використовується метод drawstring() класу java.awt.Graphics і метод paint() класу java.applet.Applet:

```
import java.applet.*;
import java.awt.*;
public class Hello extends Applet{
    public void paint(Graphics g){
        g.drawString("Привіт, студент!",20,20);
    }
}
```

Якщо Java-програма є аплетом, то для її запуску створюється файл HTML, в тезі <APPLET> якого міститься посилання на файл з розширенням .class, наприклад,

```
<APPLET Code="hello.class">.
```

Теги <APPLET> і </APPLET> є контейнером для визначення аплета.

Атрибути тега <APPLET>:

Атрибут	**Опис**
CODE	Назва аплета, що включається в сторінку
WIDTH	Ширина прямокутної області (у пікселах) у вікні браузера, резервована для роботи аплета. Обов'язковий параметр

HEIGHT	Висота прямокутної області (у пікселах) у вікні броузера, зарезервована для роботи аплета
ALT	Задає альтернативний текст, який відображатиметься в тому випадку, якщо тег <APPLET> розпізнається браузером, але завантаження аплетів відключене або не підтримується
CODEBASE	Визначає шлях до каталога класів, в якому зберігаються аплети. Якщо цей атрибут опущений, то використовуватиметься каталог, в якому розташовується сам HTML документ
NAME	Задає ім'я аплета. Цей параметр може використовуватися для адресації одного аплета до іншого на цій же сторінці
ALIGN	Створює вирівнювання аплета на сторінці (Як значення цього атрибуту можуть виступати CENTER, LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM)
VSPACE	Вказує кількість пікселів вільного простору вище і нижче за область, займану аплетом
HSPACE	Задає кількість пікселів вільного простору зліва і праворуч від області, займаної аплетом

Приклад включення вище наведеного аплета в Web-сторінку:

```
<HTML>
    <HEAD>
        <TITLE> Вас вітає Applet </TITLE>
    </HEAD>
    <BODY>
        <APPLET CODE="Hello.class" HEIGHT=100 WIDTH=150>
        </APPLET>
    </BODY>
</HTML>
```

В даному прикладі завантажуваний аплет називається Hello, для нього резервується прямокутна область висотою 100 пікселів і шириною 150 пікселів.

Область, зарезервована для аплета, заповнена кольором, встановленим броузером за замовчуванням (сірим).

Продемонструємо аплет, що містить роботу з графічними елементами та обробку подій. В основу його роботи покладено визначення відстані між двома точками на екрані. Безпосередньо точки фіксуються за клацанням миші таким чином: перше клацання миші фіксує першу точку на екрані з одночасною видачею координат цієї точки, а друге клацання мишею другу точку з видачею координат відповідно другої точки, а також відстань між двома точками (у

пікселях). При наступному клацанні мишею процес повторюється спочатку: фіксується перша крапка і так далі.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class meline1 extends Applet implements MouseMotionListener,
    MouseListener {
    // повідомлення
    String msg1 ="";
    String msg2 ="";
    // поточні координати миші
    int curX = 0, curY = 0;
    // перемикач для точок
    int k=0;
    //координати двох точок
    int x1,y1,x2,y2;
    //відстань
    double r;
    //блок прослухування подій від миші - аплет
    public void init() {
        addMouseMotionListener(this);
        ddMouseListener(this);
    }
    // Обробка події клацання миші
    public void mouseClicked(MouseEvent e){
        urX = e.getX();
        urY = e.getY();
        if (k==0) k=1;
        if (k==1){
            x1= curX; y1= curY;
            msg1 = "точка: ("+x1+","+ y1+")";
        }else
            if (k==2){
                x2 = curX;
                y2 = curY;
                msg2 = "точка 2: ("+x2+","+ y2+")";
            }
        repaint();
    }
    // інші події від миші
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e){}
```

```
        public void mouseDragged(MouseEvent e){}
        // обробка подій переміщення
        public void mouseMoved(MouseEvent e){
            //відобразити поточні координати в рядку статусу
            showStatus ("Координати: " +  e.getX() + ", " +  e.getYO);
        }
        public void paint (Graphics g){
            if (k==1){
                //виведення повідомлень про координати точок та відстані між ними
                g.drawString (msg1, x1, y1);
                k=2;
            }else
                if (k==2){
                    g.drawString(msg1, x1, y1);
                    drawString(msg2, x2, y2);
                    g.drawLine(x1,  y1,  x2,  y2);
                    r=Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
                    g.drawString("Відстань між точками: "+r, x2, y2+30); k=1;
                }
        }//  paint
    }//  meline1
```

Як зазначалося вище, для розробки аплетів з графічними елементами, потрібно застосування пакету AWT, що підтримує великий набір графічних методів. Вся графіка створюється відносно вікна. Це може бути головне, дочірнє вікно аплета або вікно автономного застосування. У прикладах, що наводяться нижче, графіка виводиться у головному вікні аплета, проте та ж техніка придатна і до будь-якого типу вікна.

Для обробки подій від миші реалізуються інтерфейси MouseListener і MouseMotionListener. Аплет є як джерелом, так і слухачем подій від миші. Змінна k фіксує стан, якій точці відповідає клацання миші. Залежно від цього формується або повідомлення msg1 або msg2 з вказівкою координат поточної точки. При обчисленні відстані між двома точками використовується метод sqrt класу Math, що містить набір методів математичних функцій.

Наведемо приклад, в якому створюється аплет, що дозволяє керувати переміщенням диску з допомогою клавіш. Диск може переміщуватись вгору, вниз, вліво та право. Програма відстежуватиме ситуацію, коли диск наблизиться до меж області. Створений клас реалізує два інтерфейси: методи роботи з мишею та методи роботи з клавіатурою.

Відразу після завантаження диск знаходиться в центрі області. Далі користувач може змінити положення, клацнувши кнопкою.

```
import java.awt.*;
import  java.awt.event.*;
import java.applet.*;
```

```
public class medisk1 extends Applet implements   MouseListener {
    //координати центру диску та радіус
    Int xC, yC, r;
    Public void init()  {
        addMouseListener(this);
        xC=this.getWidth()/2;
        yC=this.getHeigth()/2;
        r=40;
    }// init
    // обробка подій від миші
    Public void mouseClicked(MouseEvent e) {
        xC=e.getX();
        yC=e.getY();
        repaint();
    }//mouseClicked
    // інші методи інтерфейсу MouseListener
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e){}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    // малювання
    public void paint (Graphics g) {
        g.setColor(Color.blue);
        g.fillOval(xC-r,yC-r,2*r,2*r);
        g.setColor(Color.red);
        g.drawOval(xC-r,yC-r,2*r,2*r);
    }// paint
}//medisk1
```

Додаткові методи для аплетів

У класі Applet існує ряд додаткових методів, які можуть виявитися корисними при програмуванні аплетів.

URL getcodebase() повертає URL-адресу, з якої був завантажений аплет. Можна використовувати для завантаження додаткових даних (малюнків, текстів або іншої інформації). Метод дозволяє не змінювати код аплета при його переміщенні на інший сервер.

URL getdocumentbase() повертає URL-адресу HTML-документа, що запустив аплет. Використовується аналогічно getcodebase().

void resize(int width, int height) – змінює розміри вікна аплета відповідно до нових значень width і height. Якщо вікно аплета є частиною складної сторінки, то зміна його розмірів може досить сильно вплинути на конфігурацію сторінки.

void showstatus(String message) – виводить повідомлення в рядок стану (зазвичай розташовану в нижній частині вікна браузера). Можна

використовувати для виведення довідкової інформації при попаданні покажчика на певний об'єкт.

Image getimage(URL s) – метод, що в переважній кількості випадків завантажує зображення. Метод getimage сам по собі не завантажує зображення, а повертає управління відразу після створення спеціального потоку, який буде цим займатися. Згодом, при спробі відмалювати зображення, буде доступна та його частина, яка прийнята з мережі на даний момент.

Image getimage(URL s, String name) подібний до **Image getimage(URL s).** Пошук зображення здійснюється за іменем в каталозі URL.

Augioclip getaudioclip(URL s) – метод завантаження аудіокліпу для подальшого відтворення. Подібний до getimage().

AudioClip getAudioClip(URL s, String name) – метод для завантаження аудіокліпу за ім'ям вказаного URL.

void play(URL s) – завантаження і відтворення аудіокліпу. При невдалому пошуку або при виникненні помилки завантаження відтворення не виконується.

void play (URL s, String name) – завантаження і відтворення аудіокліпу за ім'ям в каталозі URL.

Appletcontext getappletcontext() – містить список всіх аплетів, запущених локально. При розміщенні на сторінці декількох аплетів список міститиме покажчики на кожен з них. Метод призначений для організації зв'язку між аплетами.

Завдання. Розробити аплет, що малює плоску криву, рівняння якої задано в таблиці відповідно до варіанту.

№ варіанта	**Рівняння**
1	$\frac{3\cos^2(x-\frac{\pi}{6})}{\frac{1}{2}+\sin y^2}$
2	$\frac{\lvert x-y\rvert}{(1+2x)^a}-e^{\sqrt{1+\omega}}$
3	$\frac{5a^{nx}}{b+c}-\sqrt{\lvert\cos x^3\rvert}$
4	$\ln\lvert a^7\rvert+\operatorname{arctg} x^2+\frac{\pi}{\sqrt{\lvert a+x\rvert}}$
5	$\sqrt[5]{\frac{(a+b)^2}{c+d}+e^{\sqrt{x+1}}}$
6	$\frac{1}{2\pi}\sqrt{\frac{2mg}{m(a\sin\alpha+b\cos\alpha)}}$
7	$\frac{1}{4}\left(\frac{1+x^2}{1-x}+\frac{1}{2}\operatorname{tg} x\right)$

8	$a+\sqrt{b}+\ln\left\|x^3\right\|+e^{\sqrt[3]{c}}$
9	$\|\sin\alpha\|\cos\frac{\alpha}{2}+\sqrt{a^2+b^2}$
10	$\sqrt[5]{x}(\operatorname{arctg}\ z+\cos^2 y)$

Лабораторна робота №7.
Обробка подій

Додатки Java не мали б користі, якби не могли обробляти події, пов'язані з діями користувача: переміщення миші, введення з клавіатури та інше.

Розглянемо обробку таких подій на прикладі стандартної бібліотеки java.awt.

Модель обробки подій побудована на основі стандартного шаблону об’єктно-орієнтованого проектування Observer/Observable. Для розгляду візьмемо будь-який компонент AWT. Для нього можна задати один або декілька клас-спостерігачів. У AWT вони називаються слухачами (listener) і описуються спеціальними інтерфейсами, назва яких закінчується як Listener. Коли з відповідним об'єктом щось відбувається, створюється об'єкт "подія" (event). Він "посилається" всім слухачам, що дає змогу знати про дію користувача. В результаті на нього можна відреагувати.

Кожна подія є підкласом класу java.util.EventObject, методи і властивості якого описані нижче. Події пакету AWT є підкласами java.awt.AWTEvent. Для зручності класи різних подій і інтерфейси слухачів розміщені в окремому пакеті java.awt.event.

Розглянемо застосування подій на прикладі простої події ActionEvent.

Нехай, в нашій програмі створюється кнопка збереження файлу:

```
Button save = new Button("Save");
add(save);
```

Коли вікно додатка з цією кнопкою з'явиться на екрані, користувач зможе натиснути її. В результаті AWT згенерує ActionEvent. Для того, щоб отримати і обробити подію, необхідно зареєструвати слухача. Назва потрібного інтерфейсу відповідає назві події ActionListener. Даний інтерфейс містить лише один метод, який має один аргумент ActionEvent. У інших слухачів методів може бути декілька.

Оголосимо клас, який реалізує цей інтерфейс:

```
class SaveButtonListener implements ActionListener {
    private Frame parent;
    public SaveButtonListener(Frame parentFrame){
        parent = parentFrame;
    }
    public void actionPerformed(ActionEvent e){
        FileDialog fd = new FileDialog(parent,
            "Save file", FileDialog.SAVE);
        fd.setVisible(true);
        System.out.println(fd.getDirectory()+"/"+
            fd.getFile());
    }
}
```

Конструктор класу вимагає в якості параметру посилання на батьківський фрейм. Без нього не можна реалізувати діалог вибору файлу FileDialog. У методі actionPerformed класу ActionListener описуються дії, які необхідно зробити після натиснення користувачем на кнопку. Потрібно відкрити діалог вибору файлу, за допомогою якого визначається шлях його збереження. Для нашого прикладу можна обмежитись виведенням на консоль.

Наступний крок полягає в реєстрації слухача. Назва методу знову відповідає назві інтерфейсу addActionListener.

```
save.addActionListener(new SaveButtonListener(frame));
```

Наведемо повний лістинг програми:

```
import java.awt.*;
import java.awt.event.*;
public class Test {
    public static void main(String args[]) {
        Frame frame = new Frame("Test Action");
        frame.setSize(400, 300);
        Panel p = new Panel();
        frame.add(p);
        Button save = new Button("Save");
        save.addActionListener(new SaveButtonListener(frame));
        p.add(save);

        frame.setVisible(true);
    }
}

class SaveButtonListener implements ActionListener {
    private Frame parent;
    public SaveButtonListener(Frame parentFrame){
        parent = parentFrame;
    }
    public void actionPerformed(ActionEvent e){
        FileDialog fd = new FileDialog(parent, "Save file", FileDialog.SAVE);
        fd.setVisible(true);
        System.out.println(fd.getDirectory()+fd.getFile());
    }
}
```

Після запуску програми з'явиться вікно з кнопкою "Save". Якщо натискувати на неї, відкриється файловий діалог. Після вибору файлу на консолі відображається повний шлях до нього.

Для кожної події AWT визначений клас XXEvent та інтерфейс XXListener. З компонентом-джерелом подій пов’язаний метод реєстрації слухача addXXListener.

Зовсім не обов'язково, щоб одна подія могла породжуватися лише одним компонентом в результаті дій користувача. Наприклад, розглянутий ActionEvent генерується при натисненні на кнопку (Button) після натиснення клавіші Enter в полі введення тексту (TextField). При подвійному клацанні миші на елементі списку (List) та інших подіях можна визначити, які події генерує той або інший компонент, по наявності методів addXXListener.

Багато слухачів, на відміну від ActionListener, мають більше ніж один метод для різних видів подій. Наприклад, MouseMotionListener спостерігає за рухом миші і має два методи:

mouseMoved (звичайний рух);

mouseDragged (переміщення з натиснутою кнопкою миші).

Інколи необхідно працювати лише з одним методом, а інші доводиться оголошувати і залишати порожніми. Для того, щоб оптимізувати роботу, в пакеті java.awt.event використовуються допоміжні класи-адаптери, наприклад, MouseMotionAdapter (назва цього класу прямо виходить з назви слухача). Ці класи наслідуються від Object і реалізують відповідний інтерфейс. Адаптер абстрактний клас, але абстрактних методів у нього немає. Всі вони оголошені порожніми. Від такого класу можна створити відповідний клас і перевизначити лише ті методи, які потрібні для програми.

Класи повідомлень (event) містять допоміжну інформацію для обробки подій. Метод getSource() повертає об'єкт-джерело події. Конкретні спадкоємці AWTEvent можуть мати додаткові методи. Наприклад, MouseEvent повідомляє про натиснення кнопки миші, а його методи getX і getY повертають координати точки, де сталася ця подія.

Разом з методом addXXListener важливу роль відіграє removeXXListener. В Java непотрібні об'єкти вилучаються з пам'яті автоматично. Це здійснює менеджер сміття, який слідкує за посиланнями на об'єкти і відстежує, щоб не залишалося посилань на непотрібні об'єкти. Якщо слухач вже виконав своє призначення і не потрібний, то з програми можна вилучити посилання на нього. Проте компонент зберігатиме його в своєму списку слухачів. Для виклику менеджеру сміття garbage collector застосовують метод removeXXListener.

Розглянемо всі події AWT і відповідних їм слухачів, визначених в Java.

Події MouseMotionListener та MouseEvent розглядалися вище в прикладі. Вони відповідають за переміщення курсора миші. Відповідний слухач має два методи - mouseMoved для звичайного переміщення і mouseDragged для переміщення з натиснутою кнопкою. Цей слухач працює не з подією MouseMotionEvent (оскільки немає такого класу), а з MouseEvent як і MouseListener.

MouseListener та MouseEvent мають методи mouseEntered і mouseExited. Перший викликається, коли курсор миші з'являється над компонентом, а другий коли виходить за межі компонента.

Для обробки натиснення кнопки миші існують три методи: mousePressed, mouseReleased і mouseClicked. Якщо користувач натиснув, а потім відпустив кнопку, то слухач отримає всі три події у вказаному порядку. Якщо клацань було декілька, то метод getClickCount класу MouseEvent поверне їх кількість. Методи getX і getY повертають координати точки, де відбулася подія. Для того, щоб визначити, яка кнопка миші натискалася, потрібно скористатися методом getModifiers і порівняти результат з константами:

(event.getModifiers() & MouseEvent.BUTTON1_MASK)!=0

Найчастіше перша кнопка відповідає лівій кнопці миші.

KeyListener та KeyEvent відстежує натиснення клавіш клавіатури і має три методи: keyTyped, keyPressed, keyReleased. Перший відповідає за введення чергового Unicode-символа з клавіатури. Метод keyPressed сигналізує про натиснення, а keyReleased - про відпуск деякої клавіші. Взаємозв'язок між цими подіями може бути непростим. Наприклад, якщо користувач натискуватиме і утримуватиме клавішу Shift і в цей час натискуватиме клавішу "A", станеться одна подія типа keyTyped і декілька keyPressed/Released. Якщо користувач натискуватиме і утримуватиме, наприклад, пропуск, то після першого keyPressed буде багато разів викликаний метод keyTyped, а після відпуску keyReleased.

У класі KeyEvent визначено велику кількість констант, які дозволяють точно ідентифікувати натискання тієї чи іншої клавіші. Поряд з цим визначається стан службових клавіш (Ctrl, Alt, Shift та інших).

У кожній програмі один із компонентів володіє фокусом і може отримувати події від клавіатури, відповідно це FocusListener та FocusEvent. Фокус можна перемістити, якщо клацнути мишкою по іншому компоненту або натиснути клавішу Tab.

Інтерфейс FocusListener містить два методи focusGained і focusLost (отриманий/втрачений).

Компоненти-спадкоємці TextComponent відповідають за введення тексту і породжують TextEvent. Слухач має один метод textValueChanged. З його допомогою можна відстежувати кожну зміну тексту. Наприклад, можна видавати користувачеві підказку за першими введеними символами.

Подію ItemListener та ItemEvent можуть генерувати такі класи, як Checkbox, Choice, List. У слухача є метод itemStateChanged, який сигналізує про зміну стану елементів.

Подія AdjustmentListener та AdjustmentEvent генерується компонентом ScrollBar. Слухач має метод adjustmentValueChanged, що сигналізує про зміну стану смуги прокрутки.

Подія WindowListener та WindowEvent сигналізує про зміну стану вікна (клас Window і його спадкоємці).

Розглянемо детально один з методів слухача windowClosing. Цей метод викликається при спробі закрити вікно. Наприклад, користувач натискає на відповідну кнопку в заголовку вікна. В Java вікна при цьому не закриваються,

оскільки AWT лише посилає WindowEvent у відповідь на таку дію. При цьому ініціювати закриття вікна повинен програміст:

```
public class WindowClosingAdapter extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        ((Window)e.getSource()).dispose();
    }
}
```

Оголошений адаптер в методі windowClosing отримує посилання на вікно, від якого прийшла подія. Щоб зробити компонент невидимим можна скористатися методом setVisible(false). Але оскільки Window автоматично породжує вікно операційної системи, існує спеціальний метод dispose, який звільняє всі системні ресурси, пов'язані з цим вікном.

Коли вікно буде закрито, у слухача викликається ще один метод windowClosed.

Подія ComponentListener та ComponentEvent відображає зміну основних параметрів компонента положення, розмір, властивість visible.

Подія ContainerListener та ContainerEvent дозволяє відстежувати зміну списку компонент, що містяться в цьому контейнері.

З розвитком Java в AWT з'являються нові події, що дозволяють підтримувати коліщатко миші. Проте всі вони працюють за тією ж схемою.

Розглянемо сутність обробки подій за допомогою внутрішніх класів.

У тілі класу можна оголошувати внутрішні класи. В прикладах, що наведені вище, така можливість не використовувалась. Однак іноді існує необхідність у використанні анонімних класів.

Наведемо приклад, в якому в програму додається кнопка, щоб додати слухача. Найчастіше доцільно описати логіку дій в окремому методі того ж класу. У випадках введення слухача в окремому класі виникає ряд незручностей. Це, насамперед, пов’язано з тим, що з'являється новий клас, якому до того ж необхідно передати посилання на вихідний клас.

Набагато зручніше реалізувати це наступним чином:

```
Button b = new Button();
b.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent e){
            processButton();
        }
    }
);
```

Розглянемо детально, що відбувається в даному прикладі. Спочатку створюється кнопка, для якої викликається метод addActionListener. Аргумент даного методу нагадує спробу створити екземпляр інтерфейсу (new ActionListener()). Фігурна дужка вказує, що породжується екземпляр нового класу, оголошення якого послідує за нею. Вказаний клас успадкується від Object

і реалізує інтерфейс ActionListener. Йому необхідно реалізувати метод actionPerformed. В цьому методі викликається processButton. Це метод, який заплановано розмістити в зовнішньому класі. Завдяки цьому внутрішній клас може безпосередньо звертатися до методів зовнішнього класу.

Такий клас називається анонімним, оскільки він не має власного імені. Проте компілятор завжди створює .class-файл для кожного класу Java. Якщо зовнішній клас називається Test, то після компіляції з'явиться файл Test$1.class.

Розглянемо приклад програмного коду, який використовує події.

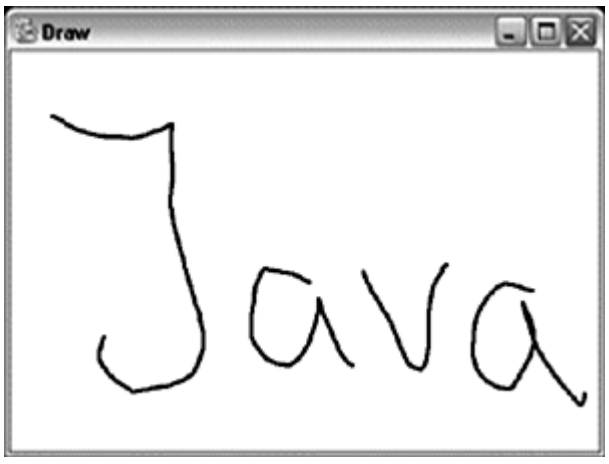
Створимо примітивний графічний редактор, який дозволяє малювати за допомогою курсора при переміщенні його з натисненою кнопкою миші. Натиснення пропуску пов’язане з очищенням області для малювання.

```
import java.awt.*;
import java.awt.event.*;

public class DrawCanvas extends Canvas {
    private int lastX, lastY;
    private int ex, ey;
    private boolean clear=false;

    public DrawCanvas () {
        super();
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                lastX = e.getX();
                lastY = e.getY();
            }
        });

        addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseDragged(MouseEvent e) {
                ex=e.getX();
                ey=e.getY();
                repaint();
            }
        });

        addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                if (e.getKeyChar()==' ') {
                    clear = true;
                    repaint();
                }
            }
        });
```

```
    }

    public void update(Graphics g) {
        if (clear){
            g.clearRect(0, 0, getWidth(), getHeight());
            clear = false;
        }else{
            g.drawLine(lastX, lastY, ex, ey);
            lastX=ex;
            lastY=ey;
        }
    }

    public static void main(String s[]) {
        final Frame f = new Frame("Draw");
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                f.dispose();
            }
        });
        f.setSize(400, 300);

        final Canvas c = new DrawCanvas();
        f.add(c);

        f.setVisible(true);
    }
}
```

Клас DrawCanvas є тією областю, на якій можна малювати. Його конструктор ініціалізував всіх необхідних слухачів. У разі настання події відбувається перемальовування (метод repaint), логіка якого описана в update. Метод main ініціалізує frame з урахуванням windowClosing.

В результаті можна що-небудь намалювати, наприклад:



Додати обробку подій в аплет дуже легко. Для цього в головному класі аплета досить перевизначити метод handleevent. Коли виникає яка-небудь подія (наприклад, користувач розташував курсор над вікном аплета або зробив клацання мишею в області цього вікна), метод handleevent отримує управління. Йому передається об'єкт класу Event, властивості якого описують подію, що виникла:

```
public Object target;  // ініціатор події
public long when;      // час, коли подія відбулася
public int  id;         // тип події(Key_press, Mouse_down...)
public int  x;          // координати
public int  y;          // курсора
public int  key;        // код клавіші
public int  modifiers;// код модифікатора (control, shift, Alt ...)
public Object arg;     // допоміжні дані
public Event evt;      // поле для з'єднання подій в списки
```

Для обробки подій, пов'язаних з мишею, зручніше перевизначати інші, менш універсальні методи. Ці методи передбачені в базовому класі Component, від якого "відбувся" клас Applet.

Якщо вам потрібно відстежувати натиснення мишею на кнопки, перевизначите метод mousedown, якщо відпуск цих клавіш – метод mouseup, переміщення – mousemove і так далі.

Аплети мають справу тільки з лівою клавішею миші.

Прототип методу handleevent має структуру:

```
public boolean handleevent(Event evt);
```

Наведемо типовий приклад обробки подій, коли всі вони обробляються на рівні аплета. В даному випадку, якщо ініціатор події має тип Button з ім'ям "OK", то виконуються відповідні дії і повертається значення true, тобто подія далі не передається.

```
class Myapplet extends Applet {
    ...
    public boolean action (Event evt, Object arg) {
    ...
        if ((ev.target instanceof Button) && arg.equals ("OK")) {
            // Виконати відповідні дії
            ...
            return true;
        } else {
            // Інші випадки
            ...

            ...
            return false; }
        }
```

…
}

Як параметр методу handleevent передається об'єкт класу Event, який містить всю інформацію про подію. По вмісту полів класу Event можна визначити координати курсора миші в момент, коли користувач натиснув клавішу, відрізнити одинарне клацання від подвійного і так далі.

Таблиця 13.1

Список полів класу Event:

public Object arg;	Довільний аргумент події, значення якої залежить від типу події
public int clickCount;	Це поле має значення тільки для події з типом Mouse_downmouse_down і містить кількість натиснень на клавішу миші. Якщо користувач зробив подвійне клацання мишею, в це поле буде записано значення 2
public Event evt;	Наступна подія в зв'язаному списку
public int id;	Тип події. Нижче ми перерахуємо можливі значення для цього поля
public int key;	Код натиснутої клавіші (тільки для події, створеної при виконанні користувачем операції з клавіатурою)
public int modifiers;	Стан клавіш модифікації <Alt>, <Ctrl>, <Shift>
public Object target;	Компонент, в якому відбулася подія
public long when;	Час, коли відбулася подія
public int x;	Координата по вісі X
public int y;	Координата по вісі Y

Таблиця 13.2

Поле id (тип події) може містити такі значення:

Значення	**Тип події**
ACTION_EVENT	Користувач хоче, щоб відбулася деяка подія
GOT_FOCUS	Компонент (у нашому випадку вікно аплета) отримав фокус введення. Про фокус введення ви дізнаєтеся з розділу, присвяченого роботі з клавіатурою
KEY_ACTION	Користувач натиснув клавішу типу "Action"

Значення	Тип події
KEY_ACTION_RELEASE	Користувач відпустив клавішу типу "Action"
KEY_PRESS	Користувач натиснув звичайну клавішу
KEY_RELEASE	Користувач відпустив звичайну клавішу
LIST_DESELECT	Відміна виділення елементу в списку
LIST_SELECT	Виділення елементу в списку
LOAD_FILE	Завантаження файлу
LOST_FOCUS	Компонент втратив фокус введення
MOUSE_DOWN	Користувач натиснув клавішу миші
MOUSE_DRAG	Користувач натиснув клавішу миші і почав виконувати переміщення курсора миші
MOUSE_ENTER	Курсор миші увійшов до зони вікна аплета
MOUSE_EXIT	Курсор миші покинув зону вікна аплета
MOUSE_MOVE	Користувач почав виконувати переміщення курсора миші, не натискаючи клавішу миші
MOUSE_UP	Користувач відпустив клавішу миші
SAVE_FILE	Збереження файлу
SCROLL_ABSOLUTE	Користувач перемістив движок смуги перегляду в нову позицію
SCROLL_LINE_DOWN	Користувач виконав над смугою перегляду операцію зрушення на один рядок вниз
SCROLL_LINE_UP	Користувач виконав над смугою перегляду операцію зрушення на один рядок вгору
SCROLL_PAGE_DOWN	Користувач виконав над смугою перегляду операцію зрушення на одну сторінку вниз
SCROLL_PAGE_UP	Користувач виконав над смугою перегляду операцію зрушення на одну сторінку вверх
WINDOW_DEICONIFY	Користувач запитав операцію відновлення нормального розміру вікна після його мінімізації
WINDOW_DESTROY	Користувач збирається видалити вікно
WINDOW_EXPOSE	Вікно буде відображено
WINDOW_ICONIFY	Вікно буде мінімізовано

Значення	Тип події
WINDOW_MOVED	Вікно буде переміщено

Якщо подія пов'язана з клавіатурою (тип події KEY_ACTION або KEY_ACTION_RELEASE), в полі key може знаходитися код натиснутої клавіші (табл. 13.3.).

Таблиця 13.3

Перелік значень поля key (код натиснутої клавіші):

Значення	Клавіша
DOWN	Клавіша переміщення курсора вниз
END	<End>
F1	<F1>
F2	<F2>
F3	<F3>
F4	<F4>
F5	<F5>
F6	<F6>
F7	<F7>
F8	<F8>
F9	<F9>
F10	<F10>
F11	<F11>
F12	<F12>
HOME	<Home>
LEFT	Клавіша переміщення курсора вліво
PGDN	<Page Down>
PGUP	<Page Up>
RIGHT	Клавіша переміщення курсора вправо
UP	Клавіша переміщення курсора вверх

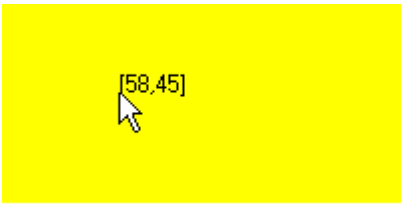
Стан клавіш модифікації <Alt>, <Ctrl>, <Shift>, вказаний в бітовому полі modifiers. Для доступу до стану окремої клавіші модифікації застосовані такі маски (табл.13.4.)

Таблиця 13.4

Перелік констант масок для доступу до стану клавіш модифікації

Значення маски	Опис
ALT_MASK	Була натиснута клавіша <Alt>
CTRL_MASK	Була натиснута клавіша <Ctrl>
SHIFT_MASK	Була натиснута клавіша <Shift>

Продемонструємо обробку подій, пов'язаних з мишею, на прикладі аплета, що у своєму вікні відображає координати курсора у момент клацання лівої клавіші миші.



Вікно аплета

Кожного разу, коли користувач клацає мишею всередині вікна аплета, воно перемальовується наново.

```
import java.applet.Applet;
import java.awt.*;
public class EventProc extends Applet{
    Event ev = null;
    ...
    public void init(){
        setBackground(Color.yellow);
        setForeground(Color.black);
    }

    public String getAppletInfo(){
        return "Name: EventProc";
    }
    public void paint(Graphics g){
        if(ev != null){
            g.drawString("[" + ev.x + "," +
                ev.y + "]", ev.x, ev.y);
        }
    }

    public boolean mouseDown(Event evt, int x, int y){
        ev = evt;
        repaint();
        return true;
    }
}
```

У головному класі аплета визначено поле ev класу Event, методи init, getappletinfo, paint і mousedown. Серед них найбільший інтерес представляють два останні.

Метод paint. У головному класі аплета визначено поле ev, призначене для тимчасового зберігання об'єктів-подій. При ініціалізації в це поле записуємо значення null:

```
Event ev = null;
```

Коли відбувається перемальовування вікна, метод paint перевіряє вміст поля ev. Якщо воно не рівне null, метод малює рядок вигляду [x, y] в точці, де знаходився курсор миші у момент виникнення події:

```
public void paint(Graphics g){
    if(ev != null){
        g.drawString("[" + ev.x + "," + ev.y + "]", ev.x, ev.y); }}
```

Координати цієї точки зберігаються в полях x і y об'єкту ev.

Метод mousedown. При виникненні події, пов'язаної з натисненням клавіші миші, метод handleevent викликає метод mousedown. У наведеному аплеті метод mousedown перевизначено таким чином:

```
public boolean mousedown(Event evt, int x, int y){
    ev = evt;
    repaint();
    return true;
}
```

Отримавши посилання на об'єкт-подію evt, реалізація методу mousedown зберігає її в полі ev, а потім перемальовує вікно аплета за допомогою методу repaint. Виконавши обробку події, метод повертає значення true. В результаті для нього не виконуватиметься обробка, прийнята по замовчуванню в методі mousedown базового класу Component.

Завдання. Написати аплет:

1.Що виводить повідомлення про натиснену клавішу клавіатури.

2.З допомогою якого можна переміщувати маленький чорний прямокутник всередині вікна аплету з допомогою клавіш управління курсором.

3.З допомогою якого можна переміщувати зображення дискети всередині вікна аплету з допомогою миші.

4.Який містить прямокутник та елементи управління ним (кнопки, текстові поля і т. і.). Аплет повинен обробляти події, що поступають від елементів управління та перемальовувати зображення.

5.Який містить стовпчикову діаграму та елементи управління нею. Задати ширину стовпчиків, їх висоти та колір. Аплет повинен обробляти події, що поступають від елементів управління та перемальовувати зображення.

6.Який містить стовпчикову діаграму та елементи управління нею, та додає до діаграми рядок-заголовок. В аплеті задати ширину стовпчиків, їх висоти, колір та рядок.

7.Який виводить графік функції та елементи управління його кольором (кнопки, текстові поля і т. і.). Аплет повинен обробляти події, що поступають від користувача та перемальовувати зображення.

8.Що являє собою графічний редактор, який дозволяє малювати за допомогою курсора. Якщо переміщати його з натиснутою кнопкою миші, з’являється лінія. Натиснення клавіші Пропуск приводить до очищення поля.

9.Що дозволяє малювати криву лінію з допомогою курсора миші.

10. Який виводить у своєму вікні координати точки, де відбулось клацання мишею.

Лабораторна робота №8.
Створення мережевих додатків. Робота з сокетами

Передача даних з використанням сокетів. У бібліотеці класів Java є дуже зручний засіб, за допомогою якого можна організувати взаємодію між додатками Java і аплетами, що працюють як на одному і тому ж, так і на різних вузлах мережі TCP/IP. Це засіб, що з'явився на базі операційної системи UNIX, – так звані сокети (sockets). Як приклад, можете уявити собі сокети у вигляді двох розеток, в які включений кабель, призначений для передачі даних через мережу. Переходячи до комп'ютерної термінології, сокети – це програмний інтерфейс, призначений для передачі даних між додатками.

Перш ніж додаток зможе виконувати передачу або прийом даних, він повинен створити сокет, вказавши при цьому адресу вузла IP, номер порту, через який передаватимуться дані, і тип сокета.

Номер порту служить для ідентифікації додатку. Існують так звані "добре відомі" номери портів, зарезервовані для різних застосувань. Наприклад, порт з номером 80 зарезервований для використання серверами Web при обміні даними через протокол HTTP.

Що ж до типів сокетів, то їх два – потокові і датаграмні.

За допомогою потокових сокетів можна створювати канали передачі даних між двома додатками Java у вигляді потоків. Потоки можуть бути вхідними або вихідними, звичайними або форматованими, з використанням або без використання буферизації.

Потокові сокети дозволяють передавати дані тільки між двома застосуваннями, оскільки вони припускають створення каналу між цими застосуваннями. Проте іноді потрібно забезпечити взаємодію декількох клієнтських застосувань з одним серверним або декількох клієнтських застосувань з декількома серверними застосуваннями. В цьому випадку можна або створювати в серверному застосуванні окремі завдання і окремі канали для кожного клієнтського застосування, або скористатися датаграмними сокетами. Останні дозволяють передавати дані відразу всім вузлам мережі, хоча така можливість рідко використовується і часто блокується адміністраторами мережі.

Для передачі даних через датаграммні сокети не потрібно створювати канал – дані посилаються безпосередньо тому застосуванню, для якого вони призначені з використанням адреси цього застосування у вигляді сокета і номера порту. При цьому одне клієнтське застосування може обмінюватися даними з декількома серверними застосуваннями чи навпаки, одне серверне застосування – з декількома клієнтськими.

Проте датаграмні сокети не гарантують доставку пакетів даних, що передаються. Навіть якщо пакети даних, які передаються через такі сокети, дійшли до адресата, не гарантується, що вони будуть отримані в тій же послідовності, в якій були передані. Потокові сокети, навпаки, гарантують доставку пакетів даних, причому в правильній послідовності.

Причина відсутності гарантії доставки даних при використанні датаграмних сокетів полягає у використанні такими сокетами протоколу UDP, який, у свою чергу заснований на протоколі з негарантованою доставкою IP. Потокові сокети працюють через протокол гарантованої доставки TCP.

Робота з потоковими сокетами. Інтерфейс сокетів дозволяє передавати дані між двома додатками, що працюють на одному або різних вузлах мережі. В процесі створення каналу передачі даних один з цих додатків виконує роль сервера, а інший – роль клієнта. Після того, як канал буде створений, додатки стають рівноправними – вони можуть передавати один одному дані симетричним чином. Розглянемо цей процес в деталях.

Ініціалізація сервера. Розглянемо дії програмного засобу, який на момент ініціалізації є сервером. Спочатку потрібно зробити серверний компонент. Для цього створюємо об'єкт класу Serversocket, вказавши конструктору цього класу номер порту , що використовується.

Serversocket ss; ss = new Serversocket(9999);

Об'єкт класу Serversocket зовсім не є сокетом. Він призначений всього лише для установки каналу зв'язку з клієнтським програмним забезпеченням, після чого створюється сокет класу Socket, придатний для передачі даних.

Установка каналу зв'язку з клієнтським додатком виконується за допомогою методу accept, визначеному в класі Serversocket:

Socket s; s = ss.accept();

Метод accept припиняє роботу потоку, який здійснив виклик, до тих пір, поки клієнтський програмний засіб не встановить канал зв'язку з сервером. Якщо програмний засіб однопоточний, його робота буде блокована до моменту установки каналу зв'язку. Уникнути повного блокування програмного засобу можна, якщо використати для створення каналу передачі даних окремий потік.

Як тільки канал буде створений, можна використовувати сокет сервера для створення вхідного і вихідного потоку класу Inputstream і Outputstream, відповідно:

Inputstream is; Outputstream os;

is = s.getInputStream();

os = s.getOutputStream();

Ці потоки можна використовувати так же, як і потоки, пов'язані з файлами.

Потрібно звернути увагу на те, що при створенні серверного сокета не вказана адреса IP і тип сокета, а тільки номер порту.

Що стосується адреси IP, то він відповідає адресі IP вузла, на якому запущений додаток сервера. У класі Serversocket визначений метод getinetaddress, що дозволяє визначити цю адресу:

public Inetaddress getinetaddress();

Тип сокета вказувати не потрібно, оскільки для роботи з датаграмними сокетами призначений клас Datagramsocket.

Ініціалізація клієнта. Процес ініціалізації клієнтського додатка виглядає досить просто. Клієнт повинен просто створити сокет як об'єкт класу Socket,

вказавши адресу IP серверного додатка та номер порту, що використовується сервером:

Socket s; s = new Socket("localhost",9999);

В цьому прикладі як адресу IP вказано спеціальну адресу localhost, призначену для тестування мережевих додатків на локальному комп'ютері, а як номер порту – значення 9999, що використовується сервером.

Тепер можна створювати вхідний і вихідний потоки. На боці клієнта ця операція виконується точно також, як і на стороні сервера:

Inputstream is;

Outputstream os;

is = s.getInputStream();

os = s.getOutputStream();

Передача даних між клієнтом і сервером. Після того, як серверний і клієнтський додатки створили потоки для прийому і передачі даних, обидва вони можуть читати і писати в канал даних, викликаючи методи read і write, визначені в класах Inputstream і Outputstream.

Наведемо фрагмент коду, в якому додаток спочатку читає дані з вхідного потоку в буфер buf, а потім записує прочитані дані у вихідний потік:

byte buf[]= new byte[512];

int lenght;

lenght = is.read(buf);

os.write(buf, 0, lenght);

os.flush();

На базі потоків класу Inputstream і Outputstream можна створити потоки, що буферизують, і потоки для передачі форматованих даних.

Завершення роботи сервера і клієнта. Після завершення передачі даних потрібно закрити потоки, викликавши метод close:

is.close();

os.close();

Коли канал передачі даних більше не потрібний, сервер і клієнт повинні закрити сокет, викликавши метод close, визначений в класі Socket:

s.close();

Серверний додаток, крім того, повинен закрити з'єднання, викликавши метод close для об'єкту класу Serversocket:

ss.close();

Клас Socket. Після короткого введення в сокети наведемо опис найцікавіших конструкторів і методів класу Socket.

Конструктори класу Socket. Найчастіше для створення сокетів в клієнтських додатках застосовується один з двох конструкторів, прототипи яких наведені нижче:

public Socket(String host,int port);

public Socket (Inetaddress address,int port);

Перший з цих конструкторів дозволяє вказувати адресу серверного вузла у вигляді текстового рядка, другий, – у вигляді посилання на об'єкт класу Inetaddress. Другим параметром задається номер порту, з використанням якого передаватимуться дані.

Методи класу Socket. Наведемо найбільш цікаві методи класу Socket.

Методи getinputstream() і getoutputstream(). Ці методи призначені для створення вхідного і вихідного потоку, відповідно:

public Inputstream getinputstream();

public Outputstream getoutputstream();

Такі потоки пов'язані з сокетом і мають бути використані для передачі даних по каналу зв'язку.

Методи getinetaddress() і getport(). Вони дозволяють визначити адресу IP і номер порту, пов'язані з даним сокетом (для віддаленого вузла):

public Inetaddress getinetaddress();

public int getport();

Метод getlocalport повертає для даного сокета номер локального порту:

public int getlocalport();

Після того, як робота з сокетом завершена, його необхідно закрити методом close:

public void close();

Метод toString(). Метод повертає текстовий рядок, що представляє сокет:

public String tostring();

Використання датаграмних сокетів. Датаграмні сокети не гарантують доставку пакетів даних. Проте, вони працюють швидше потокових і забезпечують можливість широкомовної розсилки пакетів даних одночасно всім вузлам мережі. Остання можливість використовується не дуже широко в мережі Internet, проте в корпоративній мережі Intranet нею користуються часто.

Для роботи з датаграмними сокетами додаток повинен створити сокет на базі класу Datagramsocket, а також підготувати об'єкт класу Datagrampacket, в який буде записаний прийнятий від партнера по мережі блок даних.

Канал, а також вхідні і вихідні потоки створювати не потрібно. Дані передаються і приймаються методами send і receive, визначеними в класі Datagramsocket.

Клас Datagramsocket. Розглянемо конструктори і методи класу Datagramsocket, призначеного для створення і використання датаграмних сокетів.

Конструктори класу Datagramsocket. У класі Datagramsocket визначено два конструктори, прототипи яких представлені нижче:

public Datagramsocket(int port);

public Datagramsocket();

Перший з цих конструкторів дозволяє визначити порт для сокета, другий припускає використання будь-якого вільного порту.

Методи класу Datagramsocket

Метод getlocalport(). Зазвичай серверні додатки працюють з використанням якогось заздалегідь визначеного порту, номер якого відомий клієнтським застосуванням. Тому для серверних застосувань більше підходить перший з приведених вище конструкторів.

Клієнтські додатки, навпаки, часто застосовують будь-які вільні на локальному вузлі порти, тому для них годиться конструктор без параметрів.

До речі, за допомогою методу getlocalport додаток завжди може дізнатися номер порту, закріпленого за даним сокетом:

public int getlocalport();

Метод receive(…) і send(…). Прийом і передача даних на датаграмному сокеті виконується за допомогою методів receive і send відповідно:

public void receive(Datagrampacket p);

public void send(Datagrampacket p);

Як параметр цим методам передається посилання на пакет даних (відповідно, що приймається і передається), визначений як об'єкт класу Datagrampacket.

Метод close (…). Даний метод, призначений для закриття сокета:

public void close();

Нагадаємо, що збірка сміття в Java виконується тільки для об'єктів, що знаходяться в оперативній пам'яті. Такі об'єкти, як потоки і сокети, ви повинні закривати після використання самостійно.

Клас Datagrampacket. Перш ніж приймати або передавати дані з використанням методів receive і send потрібно підготувати об'єкти класу Datagrampacket. Метод receive запише в такий об'єкт прийняті дані, а метод send – перешле дані з об'єкту класу Datagrampacket вузлу, адреса якого вказана в пакеті.

Конструктори класу Datagrampacket. Підготовка об'єкту класу Datagrampacket для прийому пакетів виконується за допомогою наступного конструктора:

public Datagrampacket(byte ibuf[], int ilength);

Цьому конструктору передається посилання на масив ibuf, в який потрібно буде записати дані, і розмір цього масиву ilength.

Якщо потрібно підготувати пакет для передачі, треба застосувати конструктор, який додатково дозволяє задати адресу IP iaddr і номер порту iport вузла призначення:

public Datagrampacket(byte ibuf[], int ilength InetAddress iaddr,
    int iport);

Таким чином, інформація про те, в який вузол і на який порт необхідно доставити пакет даних, зберігається не в сокеті, а в пакеті, тобто в об'єкті класу Datagrampacket.

Методи класу Datagrampacket. Окрім тільки що описаних конструкторів, в класі Datagrampacket визначено чотири методи, що дозволяють отримати дані і інформацію про адресу вузла, з якого прийшов пакет, або для якого призначений пакет.

Метод getdata() . Повертається посилання на масив даних пакету:

public byte[] getdata();

Метод getlength(). Визначає розмір пакету, дані з якого зберігаються в цьому масиві:

public int getlength();

Методи getaddress() і getport(). Визначають адресу і номер порту вузла, звідки прийшов пакет, або вузла, для якого призначений пакет:

public Inetaddress getaddress();

public int getport();

Якщо ви створюєте клієнт-серверну систему, в якій сервер має заздалегідь відому адресу і номер порту, а клієнти – довільні адреси і різні номери портів, то після отримання пакету від клієнта сервер може визначити за допомогою методів getaddress і getport адресу клієнта для встановлення з ним зв'язку.

Якщо ж адреса сервера невідома, клієнт може посилати широкомовні пакети, вказавши в об'єкті класу Datagrampacket адресу мережі. Така методика зазвичай використовується в локальних мережах.

Як вказати адресу мережі?

Адреса IP складається з двох частин – адреси мережі і адреси вузла. Для розділення компонент 32-розрядної адреси IP використовується 32-розрядна маска, в якій бітам адреси мережі відповідають одиниці, а бітам адреси вузла – нулі.

Наприклад, адреса вузла може бути вказана як 193.24.111.2. Виходячи із значення старшого байта адреси, це мережа класу C, для якої за замовчуванням використовується маска 255.255.255.0. Отже, адреса мережі буде такою: 193.24.111.0.

Завдання. Написати додаток:

1. Що виконує функції Web-браузера, який виконує обробку наступних тегів із усіма їх параметрами: TITLE, TABLE, TR, TD, TH, CAPTION, IMG, A, FONT, HR, BR.
2. Що виконує функції proxy-сервера для обслуговування декількох клієнтів.
3. Що виконує відправлення електронної пошти по заданих адресах з можливістю написання листа і вкладення файлів.
4. Що виконує прийом електронних повідомлень з поштового сервера та їхнє візуальне відображення.
5. Що виконує функції Web-браузера по обробці наступних тегів із усіма їх параметрами: TITLE, HR, BR, SELECT, OPTION, IMG, A, BASEFONT.
6. Що виконує функції Web-сервера по організації віртуальної кореневої директорії, щодо якої здійснюється відправлення статичних HTML-сторінок.
7. Що виконує функції FTP-сервера по підтримці основних команд: створення і видалення директорії, одержання інформації про файли, відправлення і прийом файлів.

8. Що виконує функції FTP-клієнта по відправленню та прийому файлів в двох режимах – бінарному та текстовому із графічним інтерфейсом.

9. Що реалізує протокол ICQ (прийом та відправлення повідомлень, ведення бази отриманих і відправлених повідомлень, підтримку контакт-листа).

10. Що забезпечує одержання списку підтримуваних груп новин із сервера, перегляд заданих груп новин, сортування отриманих повідомлень.

Список використаної літератури

1. Методологія інформаційних систем та баз даних: теоретичний і практичний підходи : навчальний посібник / укл. Ю.О. Ушенко, М.Л. Ковальчук, М.С. Гавриляк, А.Л. Негрич. – Чернівці : Чернівецький нац. ун-т, 2021. – 244 с.
2. Алгоритми і структура даних: Навчальний посібник / В.М.Ткачук. - ІваноФранківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. 286 с.
3. Алгоритми та структури даних. Навчальний посібник / Т. О. Коротєєва. Львів : Видавництво Львівської політехніки, 2014. - 280 с.
4. Ковалюк Т.В. Основи програмування. / Ковалюк Т.В. Київ: BHV Киев, 2005. 400 с.
5. Глоба Л. С. Розробка інформаційних ресурсів та систем [Електронний ресурс] : конспект лекцій / Л. С. Глоба, Т. М. Кот. - Київ : НТУУ "КПІ",
6. Кей С. Хорстманн (2014). Java SE 8. Вводный курс. «Вільямс». ISBN 978-5-8459-1900-7.
7. Фрэд Лонг та ін. (2014). Руководство для программиста на Java: 75 рекомендаций по написанию надежных и защищенных программ. «Вільямс». ISBN 978-5-8459-1897-0.