# ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ МОВОЮ JAVA

# 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.](http://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 С++ 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 С++, 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).

# 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  –27 до 27–1 | 0 |
| short | коротке ціле | 16 бит | значення в діапазоні від  –216 до 216–1 | 0 |
| Int | Ціле | 32 бит | значення в діапазоні від  –232 до 232–1 | 0 |
| Long | довге ціле | 64 бит | значення в діапазоні від  –264 до 264–1 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Float | плаваюча крапка | 32 бит | значення в діапазоні від1.7\*10–38 до 1.7\*1038 | 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: 0х51.

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 а 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 | if (a!=14) |  | && | І | if((a>14)&&(b>17) |
| < | менше ніж | if (a<14) |  | | | побітове | 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) |  | ^ | Виключаю че 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:

1. res=4
2. res=–4
3. res=3
4. res=6
5. res=7
6. res=3
7. res=6
8. 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.

# 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'екта\*/

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 у = 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 а;// локальна змінна методу а = 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, у; /\*по замовчуванню x=0 і y=0 \*/ public Locate(){

x = 1;

у = 1;

}

public void setx(double а){ x = а;

}

void sety(double b){ /\*видимість по замовчуванню\*/ у = b;

}

public double getx(){ return x;

}

public double gety(){ return у;

}

}

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.

# 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[] а.

/\* 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 а

}

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 а. 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 а, 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[][] а; Matrix(int n, int m){

// створення і заповнення випадковими значеннями а = new int[n][m];

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

а[i][j]= (int) (Math.random() \* 5);

show();

}

public Matrix(int n, int m, int k){

а = new int[n][m];

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++) {

а[i][j]= k;

}

if(k!=0) show();

}

public void show() { System.out.println("Maтріца : " + а.length+

" на " + а[0].length);

for (int i = 0; i < а.length; i++) {

for (int j = 0; j < а[0].length; j++) System.out.print(а[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:

Maтриця : 2 на 3

1 1 1

2 4 1

Maтриця : 3 на 4

0 1 2 3

3 1 0 4

3 4 0 4

Добуток матриць:

Maтриця : 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.

# 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.

# 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 А об'єктом класу В. При цьому отриманий доступ до поля х класу А. В другому випадку при створенні об'єкту objB класу В дістав доступ до поля х класу В. Проте скориставшись перетворенням типів вигляду: ((A)objB).x або((B)objA).x можна легко дістати доступ до поля х з відповідного класу.

Проілюструємо на прикладі конструктора класу А одну із сторін поліморфізму :

public A() { y = getX();

}

Метод getX() міститься і в класі A, і в класі В. При створенні об'єкту класу В одним із способів:

A objA = new B(); B objB = new B();

в будь-якому разі спочатку викликається конструктор класу А. Але, оскільки створюється об'єкт класу В, то і метод getX() відповідно викликається той, що належить класу В. Цей метод в свою чергу оперує полем х, що ще не було проініціалізованим для класу В. В результаті y набуде значення х 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 А");

}

void myMethod(int i) {

System.out.println("метод класу А з аргументом");

}

}

class B extends A {

void myMethod(int i) {

System.out.println("метод класу В з аргументом");

}

}

public class C extends B {

{

System.out.println("клас C");

}

void myMethod() { System.out.println("метод класу С");

}

}

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 А клас C

метод класу С

метод класу В 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.

# 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;

+ b;

}

public String toString() {

return getClass().getName() + "@"+ " name=" + str + " 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

# 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 одержуює 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 {

сhiti\_uch = InetAddress.getByName("[www.chiti.uch.net](http://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**](http://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](http://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](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 getInputStrеam() і getOutputStrеam() 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().

Цей приклад можна застосувати на одному комп'ютері, що буде одночасно виступати в ролі клієнта і сервера. Для цього використовуються статичні методи getLocalHost() класу InetAddress для здобуття динамічної IP-адреси комп'ютера, яка виділяється при вході в Internet.

**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.

# 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 С++.
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 С++.
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.

# 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 мітиться 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() мітиться 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 аrgs. 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 | 3cos2 (x  )  6  1  sin y 2  2 |
| 2 | | x  y |  e 1  (1 2x)a |
| 3 | 5a nx  3  | cosx |  b  c |
| 4 | ln | a 7 | arctg x 2    | a  x | |
| 5 | (a  b)2  5  e x1  c  d |

|  |  |
| --- | --- |
| 6 | 1 2mg  2 m(a sin   b cos) |
| 7 | 1 1  x 2 1     tg x  4  1  x 2  |
| 8 | 3 с  a  b  ln x3  e |
| 9 | | sin  | cos   a 2  b2  2 |
| 10 | 5 x (arctg z  cos2 y) |

# 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.

# 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.

* 1. Створити ієрархію об’єктів за наступною схемою: транспортні засоби – автомобіль, потяг, літак.
  2. Створити ієрархію класів чотирикутник – прямокутник, квадрат та трапеція.
  3. Створити ієрархію класів за наступною схемою: Преса – газета, журнал та електронне видання.
  4. 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 Datе). У класі Clock організуйте отримання поточної дати за допомогою конструктора класу Date без параметрів.
  5. 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 i-й елемент; 4) видалити i-й елемент; 5) вивести значення масиву на екран.
  6. 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 Datе). 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.
  7. Розробіть програму, яка виводить на екран таймер на три хвилини з посекундним зворотним відліком часу. Скористайтеся методами класу java.util.Date.
  8. Напишіть програму, яка реалізує додавання, віднімання, множення, ділення комплексних чисел. Для цього створіть клас Complex, в якому реалізуйте відповідні методи.
  9. 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.

# 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 А 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){

Е

}

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. |

# 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 х направлена вправо, а вісь у – вниз. Така система координат використовується в багатьох комп'ютерних системах, але не відповідає математичній системі координат, де вісь у 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 (х1, у1) початку і (х2, у2) 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 "у" до верхньої точки прописних букв (наприклад, "М").

# 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.

# 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.\* разміщена 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 | 3cos2 (x  )  6  1  sin y 2  2 |
| 2 | | x  y |  e 1  (1 2x)a |
| 3 | 5a nx  3  | cosx |  b  c |
| 4 | ln | a 7 | arctg x 2    | a  x | |
| 5 | (a  b)2  5  e x1  c  d |
| 6 | 1 2mg  2 m(a sin   b cos) |
| 7 | 1 1  x 2 1     tg x  4  1  x 2  |
| 8 | 3 с  a  b  ln x3  e |
| 9 | | sin  | cos   a 2  b2  2 |
| 10 | 5 x (arctg z  cos2 y) |

# 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);

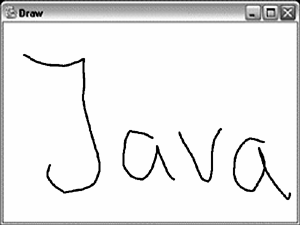
f.setVisible(true);

}

}

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 у; // 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" |
| KEY\_ACTION\_RELEASE | Користувач відпустив клавішу типу "Action" |
| KEY\_PRESS | Користувач натиснув звичайну клавішу |
| KEY\_RELEASE | Користувач відпустив звичайну клавішу |

|  |  |
| --- | --- |
| **Значення** | **Тип події** |
| 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 |
| 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> |

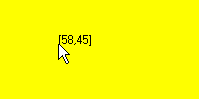
|  |  |
| --- | --- |
| **Значення** | **Клавіша** |
| F1 | <F1> |
| F2 | <F2> |
| F3 | <F3> |
| F4 | <F4> |
| F5 | <F5> |
| F6 | <F6> |
| F7 | <F7> |
| F8 | <F8> |
| F9 | <F9> |
| F10 | <F10> |
| F11 | <F11> |
| F12 | <F12> |
| HOME | <Home> |
| LEFT | Клавіша переміщення курсора вліво |
| PGDN | <Page 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, у] 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 у 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 у){ ev = evt;

repaint(); return true;

}

Отримавши посилання на об'єкт-подію evt, реалізація методу mousedown зберігає її в полі ev, а потім перемальовуває 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.

1. Що дозволяє малювати криву лінію з допомогою курсора миші.
2. Який виводить у своєму вікні координати точки, де відбулось клацання мишею.

# 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 также, 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 С, 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.

# ЛІТЕРАТУРА

1. Ноутон П., Шилдт Г. Java 2 / пер. с англ. – СПб.: БХВ-Петербург, 2001. – 1072 с.
2. Хабибуллин И.Ш. Самоучитель Java. – СПб.: БХВ-Петербург, 2002. – 464 с.
3. Дмитриева М.В. Самоучитель JavaScript. – СПб.: БХВ-Петербург, 2003. – 512 с.
4. Дуванов А.А. Web-конструирование. DHTML. – СПб.: БХВ- Петербург, 2003.
5. Программирование на Java / Вязовик Н.А. – М.: ИНТУИТ.РУ

«Интернет-университет информационных технологий», 2003. – 592 с.

1. Вебер Д. Технология Java в подлиннике: пер с англ. – СПб.: ВНV– Санкт-Петербург, 1997. – 1104 с.
2. Основы Web-технологий/ Храмцов П.Б., Брик С.А., Русак А.М. и др.; под. ред. Храмцова П.Б. – М.: ИНТУИТ.РУ « Интернет- университет-информационных технологий», 2003. – 512 с.