Оголошення класів

Лекція 3

доц. кафедри Інформатики
Сінельнікова Т.Ф.

ХНУРЕ, Кафедра Інформатики
informatika@kture.kharkov.ua

- Модифікатори доступу
- Назва класу
- Тіло класу
- Оголошення методу
- Сигнатура методу
- Throws-вираз
- Вираз return
- Оголошення конструкторів
- Ініціалізатори
- Ініціалізації final-полів.
- Метод main

В Java модифікатори доступу вказуються для:
• типів (класів та інтерфейсів) оголошення верхнього рівня;
• елементів посилальних типів (полів, методів, внутрішніх типів);
• конструкторів класів.
масив також може бути недоступний в тому і тільки в тому випадку, якщо не доступний тип, на основі якого він оголошений.

Модифікатори доступу

• public;
• private;
• protected;
• якщо не вказано ні один з цих 3 типів, то рівень доступу визначається за замовчуванням (default).

```
public class Human {
private int age;
/ / Метод, який повертає значення age
public int getAge () {
return age;
}
/ / Метод, який встановлює значення age
public void setAge (int a) {
age = a;
}
}

Human h = getHuman ();
int i = h.getAge (); / / Звернення через метод
```

процес зміни типу поля age:

```
public class Human {
/ / Поле отримує новий тип double
private / * int * / double age;
/ / Старі методи працюють з округленням значення
public int getAge () {
return (int) Math.round (age);
}
public void setAge (int a) {
age = a;
}
/ / Додаються нові методи для роботи з типом double
public double getExactAge () {
return age;
}

public void setExactAge (double a) {
age = a;
}
}
```

Human h = getHuman ();
int i = h.getAge (); / / Коректно

Human h = getHuman ();
double d = h.getExactAge (); / / Точне значення віку

в більшості випадків доступ до полів краще реалізовувати через спеціальні методи (accessors) для читання (getters) і запису (setters).

механізми перевірки вхідних значень:
public void setAge (int a) {
(if a> = 0) {
age = a;
}
}
Підіб'ємо підсумки. Функціональність класу необхідно розділяти на відкритий інтерфейс,
описує дії, які будуть використовувати зовнішні типи, і на внутрішню
реалізацію, яка використовується тільки всередині самого класу ..

• Пакети завжди доступні, тому у них немає одіфікаторов доступу всі вони public, тобто, будь-який існуючий в системі пакет може бути використаний з будь-якої точки програми.
• Типи (класи та інтерфейси) верхнього рівня оголошення. При їх оголошенні є всього дві можливості: вказати модифікатор public або не вказувати його. Якщо доступ до типу є public, то це означає, що він доступний з будь-якої точки коду. Якщо ж він не public, то рівень доступу призначається за умовчанням: тип доступний тільки всередині
того пакету, де він оголошений.
• Масив має той же рівень доступу, що і тип, на основі якого він оголошений.
• Елементи й конструктори об'єктних типів. Мають усіма 4 можливими значеннями рівня доступа.Все елементи інтерфейсів є public.

```
package first;
/ / Певний клас Parent
public class Parent {
}
package first;
/ / Клас Child успадковується від класу Parent,
/ / Але має обмеження доступу за умовчанням
class Child extends Parent {
}
public class Provider {
public Parent getValue () {
return new Child ();
}
}
```

```
package second;
```
Розмежування доступу в Java

```
import first. *;
public class Test {
public static void main (String s []) {
Provider pr = new Provider ();
Parent p = pr.getValue ();
System.out.println (p.getClass (). GetName ());
/ / (Child) p - призведе до помилки компіляції!
}
}
```
Результатом буде:
first.Child

```
public class Point {
private int x, y;
public boolean equals(Object o) {
if (o instanceof Point) {
Point p = (Point)o;
return p.x==x && p.y==y;
}
return false;
}
```

Назва класу
коректний Java-ідентифікатор.

class A {}
Фігурні дужки позначають тіло класу

Область видимості класу, де він може бути доступний за своїм простому імені - його пакет.

Тіло класу може містити оголошення елементів (members) класу:

• полів;
• методів;
• внутрішніх типів (класів та інтерфейсів);

та інших допустимих конструкцій:

• конструкторів;
• ініціалізаторів;
• статичних ініціалізаторів

Оголошення полів починається з перерахування модифікаторів. Можливе застосування будь-якого з 3 модифікаторів доступу, або ніякого зовсім, що означає рівень доступу.
Поле може бути оголошено final, що означає, що воно ініціалізується рівно один раз і більше не буде змінювати свого значення. Найпростіший спосіб роботи з final-змінними - ініціалізація при оголошенні:
final double PI = 3.1415;
допускається ініціалізація final-полів в кінці кожного конструктора класу.

кілька імен полів з можливими ініціалізаторами:

```
int a;
int b = 3, c = b +5, d;
Point p, p1 = null, p2 = new Point ();
```

Повторювані імена полів заборонені.

Забороняється використовувати поле в ініціалізації інших полів до його оголошення.

```
int y = x;
int x = 3;
```

в іншому поля можна оголошувати і нижче їх використання:

```
class Point {
int getX () {return x;}
int y = getX ();
int x = 3;
public static void main (String s []) {
Point p = new Point ();
System.out.println (p.x + "," + p.y);
}
}
```

Результатом буде:

3, 0

Оголошення методу складається з заголовка і тіла методу.
Заголовок складається з:
• модифікаторів (доступу в тому числі);
• типу значення, що повертається або ключового слова void;
• імені методу;
• списку аргументів у круглих дужках (аргументів може не бути);
• спеціального throws-вирази.
Заголовок починається з перерахування модифікаторів. Для методів доступний будь-який з 3 можливих модифікаторів доступу. Також допускається використання доступу по замовчуванням.
Крім цього, існує модифікатор final, який говорить про те, що такий метод не можна перевизначати в спадкоємців.

/ / Void calc (double x, y); - помилка!
void calc (double x, double y);

Важливим поняттям є **сигнатура (signature) методу**. Сигнатура визначається ім'ям методу і його аргументами (кількістю, типом, порядком прямування). Якщо для полів забороняється збіг імен, то для методів в класі заборонено створення двох методів з однаковими сигнатурами.

наприклад,

```
class Point {
void get () {}
void get (int x) {}
void get (int x, double y) {}
void get (double x, int y) {}
}
```

Наступні пари методів несумісні один з одним в одному класі:

```
void get () {}
int get () {}
void get (int x) {}
void get (int y) {}
public int get () {}
private int get () {}
```

```
class Test {
int get() {
return 5;
}
Point get() {
return new Point(3,5);
}
void print(int x) {
System.out.println("it's int! "+x);
}
void print(Point p) {
System.out.println("it's Point! "+p.x+", "+p.y);
}
public static void main (String s[]) {
Test t = new Test();
t.print(t.get()); // Двозначність!
}
}
```

Завершує заголовок методу **throws-вираз**. Він застосовується для коректної роботи з помилками в Java
Приклад оголошення методу:

```
public final java.awt.Point createPositivePoint (int x, int y)
throws IllegalArgumentException
{
return (x> 0 && y> 0)? new Point (x, y): null;
}
```

Далі, після заголовка методу слід тіло методу. Воно може бути порожнім, і тоді записується одним символом "крапка з комою". native-методи завжди мають тільки пусте тіло, оскільки справжня реалізація написана на іншій мові. Звичайні ж методи мають непорожнє тіло, яке описується в фігурних дужках

```
/ / Приклад викличе помилку компіляції
public int get () {
if (condition) {
return 5;
}
}
```

Видно, що хоча тіло методу містить return-вираз, однак не при будь-якому розвитку подій повертається значення буде згенеровано.

```
public int get () {
if (condition) {
return 5;
} Else {
return 3;
}
}
```

Звичайно, значення, вказане після слова return, має бути сумісний по типу з
оголошеним повертається значенням

У методі без повертається значення (зазначено void) також можна використовувати **вираз return** без будь-яких аргументів. Його можна вказати в будь-якому місці методу, у цій точці виконання методу буде завершено:

```
public void calculate (int x, int y) {
if (x <= 0 | | y <= 0) {
return; / / некоректні вхідні значення, вихід з методу
}
... / / Основні обчислення
}
```

Виразів return (з параметром або без для методів з / без значення, що повертається) в тілі одного методу може бути скільки завгодно.

Оголошення конструкторів

Формат оголошення конструкторів схожий на спрощене оголошення методів. Також виділяють заголовок і тіло конструктора. Заголовок складається, по-перше, з модифікаторів доступу (ніякі інші модифікатори не припустимі). Потім вказується ім'я класу,

```
public class Human {
private int age;
protected Human (int a) {
age = a;
}
public Human (String name, Human mother, Human father) {
age = 0;
}
}
```

```
public class Parent {
private int x, y;
public Parent() {
x=y=0;
}
public Parent(int newx, int newy) {
x=newx;
y=newy;
}
}
```

```
public class Vector {
private int vx, vy;
protected double length;
public Vector(int x, int y) {
vx=x;
vy=y;
length=Math.sqrt(vx*vx+vy*vy);
}
public Vector(int x1, int y1, int x2, int y2) {
vx=x2-x1;
vy=y2-y1;
length=Math.sqrt(vx*vx+vy*vy);
}
}
```

```
public class Vector {
private int vx, vy;
protected double length;
public Vector(int x, int y) {
super();
vx=x;
vy=y;
length=Math.sqrt(vx*vx+vy*vy);
}
public Vector(int x1, int y1, int x2, int y2) {
this(x2-x1, y2-y1);
}
}
```

У ряді випадків модифікатор private може бути корисний. наприклад:

• private-конструктор може містити ініціалізували дії, а інші конструктори будуть використовувати його за допомогою this, причому пряме звернення до цього конструктору з якихось причин небажано;

• заборона на створення об'єктів цього класу, наприклад, неможливо створити екземпляр класу Math;

• реалізація спеціального шаблона проектування з ООП Singleton, для роботи якого потрібно контролювати створення об'єктів, що неможливо у випадку наявності не-private конструкторів.

Ініціалізатори

Записуються об'єктні ініціалізатори дуже просто - всередині фігурних дужок.

```
public class Test {
private int x, y, z;
/ / Ініціалізатор об'єкта
{
x = 3;
if (x> 0)
y = 4;
z = Math.max (x, y);
}
}
```

Ініціалізатори не мають імен, виконуються при створення об'єктів і не можуть бути викликані явно, не передаються у спадок

При створенні екземпляра класу викликаний конструктор виконується наступним чином:

• якщо першим рядком йде звернення до конструктора батьківського класу (явне або доданий компілятором за замовчуванням), то цей конструктор виповнюється;

• в разі успішного виконання викликаються все ініціалізатор полів і об'єкта в тому порядку, в якому вони оголошені в тілі класу;

• якщо першим рядком йде звернення до іншого конструктору цього ж класу, то він викликається.

Повторне виконання ініціалізаторів не проводиться.

Ініціалізації final-полів.

Головна вимога - щоб такі поля були проініціалізувати рівно один раз. Це можна забезпечити в наступних випадках:

• ініціалізувати поле при оголошенні;

• ініціалізувати поле рівно один раз в ініціалізатор об'єкта (він повинен бути
записаний після оголошення поля);

• ініціалізувати поле рівно один раз в кожному конструкторі, в першому рядку которогостоіт явне або неявне звернення до конструктора батька. Конструктор, в першому рядку якого стоїть this, не може і не повинен ініціалізувати final-поле, так як ланцюжок this-дзвінків призведе до конструктора з super, в якому ця ініціалізація обов'язково присутній.

```
public class Test {
{
System.out.println ("initializer");
}
int x, y = getY ();
final int z;
{
System.out.println ("initializer2");
}
private int getY () {
System.out.println ("getY ()" + z);
return z;
}
public Test () {
System.out.println ("Test ()");
z = 3;
}
public Test (int x) {
this ();
System.out.println ("Test (int)");
/ / Z = 4; - не можна! final-поле вже було ініціалізовані
}
}
```

Після виконання вираження new Test () на консолі з'явиться:

```
initializer
getY () 0
initializer2
Test ()
```

Метод main

Програма, написана на Java, є набором класів. Вхідною точкою є метод main ().

:

```
public static void main (String [] args) {
}
```

Метод не повертає ніякого значення. Аргументом методу main () є масив рядків. Він заповнюється додатковими параметрами, які були вказані при виклику методу.

```
package test.first;
public class Test {
public static void main (String [] args) {
for (int i = 0; i <args.length; i + +) {
System.out.print (args [i] + "");
}
System.out.println ();
}
}
```

```
javac test \ first \ Test.java
java test.first.Test
java test.first.Test Hello, World!
```

Результатом роботи програми буде:

```
Hello, World!
```