JAVA

ПЛАТФОРМЕНО-НЕЗАВИСИМИ ПРОГРАМНИ ЕЗИЦИ

Лекция 5. Класове и обекти

доц. д-р инж. Румен П. Миронов

1. Основни принципи на ООП

Класовете (class) са описание (модел) на реални предмети или явления, наречени същности (entities). Например класът "Студент".

Класовете имат характеристики – в програмирането са наречени свойства (properties). Например съвкупност от оценки.

Класовете имат и поведение – в програмирането са наречени **методи** (methods). Например явяване на изпит.

Методите и **свойствата** могат да бъдат видими и невидими – от това зависи дали всеки може да ги използва или са само за вътрешна употреба в рамките на класа.

Обектите (objects) са екземпляри (инстанции) на класовете. Например Иван е Студент, Петър също е студент.

1. Основни принципи на ООП

Един програмен език е обектно-ориентиран, ако той позволява не само работа с класовете и обектите, но и дава възможност за имплементирането и използването на принципите и концепциите на ООП: наследяване, абстракция, капсулация и полиморфизъм.

- ***Наследяване (Inheritance)*** - йерархиите от класове подобряват четимостта на кода и позволяват преизползване на функционалност;

- ***Абстракция (Abstraction)*** – един обект се вижда само от гледната точка, която ни интересува, катро се игнорират всички останали детайли;

- ***Капсулация (Encapsulation)*** - скриване на ненужните детайли в класовете и предоставяне на прост и ясен интерфейс за работа с тях.

- ***Полиморфизъм (Polymorphism)*** – еднотипна работа с различни обекти, които дефинират специфична имплементация на някакво абстрактно поведение.

2. Декларация и дефиниция на клас

Клас (**class**) наричаме описанието на даден обект от реалността. Класът представлява шаблон, който описва видовете състояния и поведението на обектите (екземплярите), които биват създавани от този клас (шаблон).

Обект (object) наричаме екземпляр създаден по дефиницията (описанието) на даден клас. Когато един обект е създаден по описанието, което един клас дефинира, казваме, че обектът е от тип "името на този клас".

Класът съдържа дефиниция на това какви данни трябва да се съдържат в един обект, за да се опише състоянието му. Обектът (конкретния екземпляр от този клас) съдържа самите данни. Тези данни дефинират състоянието му.

Освен състоянието, в класа също се описва и поведението на обектите. Поведението се изразява в действията, които могат да бъдат извършвани от обектите. Средството на ООП, чрез което можем да описваме поведението на обектите от даден клас, е декларирането на методи в класа.

2. Декларация и дефиниция на клас

Елементи на класа

- ***Декларация*** **на класа (class declaration)** – това е редът, на който се декларира името на класа. Например:

```
public class Dog {  .....   }
```

- ***Тяло*** **на клас** – по подобие на структурите, класовете също имат част, която следва декларацията им, оградена с фигурни скоби – "{" и "}", между които се намира съдържанието на класа. Тя се нарича тяло на класа.

```
public class Dog {
                    // ... Here the class body comes ...
  }
```

2. Декларация и дефиниция на клас

- ***Полета*** (fields) – полетата са променливи (член-променливи), декларирани в класа. В тях се пазят данни, които отразяват състоянието на обекта и са нужни за работата на методите на класа. Стойността, която се пази в полетата, отразява конкретното състояние на дадения обект, но съществуват и такива полета, наречени статични, които са общи за всички обекти.

```
private String name; // Field/Property-storage definition
```

- ***Свойства*** (properties) – наричаме характерните особености на даден клас. Обикновено стойността на тези характеристики се пази в полета. Подобно на полетата, свойствата могат да бъдат притежавани само от конкретен обект, или да са споделени между всички обекти от тип даден клас.

- ***Методи*** (methods) – методите са тези части от класа, където се описва поведението на обектите от даден тип. В методите се изпълняват алгоритмите и се обработват данните за обекта.

2. Декларация и дефиниция на клас

- ***Конструктор*** (constructor) – това е специален метод, който се използва за създаване на нови обекти. Носи името на своя клас и няма тип на връщана стойност (not even void).

Всички класове трябва да имат поне един конструктор.

Ако класът не декларира явно такъв, компилаторът на Java автоматично създава конструктор без параметри, наречен default constructor (конструктор по подразбиране).

- ***Референцията*** е променливата, чрез която достъпваме обекта. В нея, за разликата от променливите от примитивен тип, не се съдържа самата стойност (т.е. самият обект), а адрес към реалния обект в хийпа.

Когато декларираме една променлива от тип, някакъв клас, но не искаме тя да е инициализирана с връзка към конкретен обект, тогава трябва да й присвоим стойност null.

2. Декларация и дефиниция на клас

```
class Dog {                                          // Openening brace of the class body
    private String name;                             // Property-field definition

    public Dog() {                                   // Constructor definition
                this.name = "Sharo";   }
    public Dog( String name ) {                      // Constructor definition
                this.name = name;      }
    public String getName() {                        // Property getter-method definition
                return this.name;      }
    public void setName( String name ) {             // Property setter-method definition
                this.name = name;      }
    public void bark() {                             // Method definition
                System.out.printf("Dog %s said: Wow-wow!%n", name);
        }
}                                                     // Closing brace of the class body
```

2. Декларация и дефиниция на клас

Използване на дефиниран клас:

- Създаване на обект от тип този клас. За целта използваме ключовата дума **new** в комбинация с някой от конструкторите на класа. Това ще създаде обект от дадения тип.

- Манипулиране на новосъздадения обект. Той трябва да бъде присвоен на променлива от типа на обекта. По този начин в тази променлива ще бъде запазена връзката (**референцията**) към него.

Чрез променливата, използвайки точкова нотация, можем да извикваме методите, getter и setter методите на обекта, както и да достъпваме полетата (член-променливите) му.

2. Декларация и дефиниция на клас

```
public static void main(String [] args) {
        Scanner input = new Scanner(System.in);
         System.out.print("Write first dog's name: ");
         String firstDogName = input.nextLine();
       Dog firstDog = new Dog(firstDogName); // Assign dog name with a constructor
         System.out.print("Write second dog's name: ");
       Dog secondDog = new Dog();
         secondDog.setName(input.nextLine());        // Assign dog name with a property
       Dog thirdDog = new Dog();                     // Create a dog with a default name
         Dog[] dogs = new Dog[] { firstDog, secondDog, thirdDog };
         for (Dog dog : dogs) {
                 dog.bark();
         }
}
```

```
Write first dog's name: Bobcho
Write second dog's name: Walcho
Dog Bobcho said: Wow-wow!
Dog Walcho said: Wow-wow!
Dog Sharo said: Wow-wow!
```

3. Полета и методи

Методи - деклариране, имплементация и извикване

Деклариране на метод наричаме регистрирането на метода, за да бъде разпознаван в останалата част на Java.

[public][static] <return_type> <method_name> ([<param_list>])

Задължителните елементи в декларацията на един метод са:

- Тип на връщаната от метода стойност – <return_type>.
- Име на метода – <method_name>.
- Списък с параметри на метода – <param_list> – съществува само ако метода има нужда от тях в процеса на работата си. Списъкът от параметри може да е празен (тогава се пише "()" след името на метода). Дори методът да няма параметри, кръглите скоби трябва да присъстват в декларацията му.

3. Полета и методи

В обектно-ориентираното програмиране, начинът, по който еднозначно се разпознава един метод е чрез двойката елементи от декларацията му – име на метода и списък от неговите параметри. Тези два елемента определят така наречената **спецификация** на метода (в литературата се среща и като **сигнатура** на метода).

Java като език за обектно -ориентирано програмиране, също разпознава еднозначно различните методи, използвайки тяхната спецификация – името на метода <method_name> и списъкът с параметрите на метода – <param_list>.

Типът на връщаната стойност на един метод е част от декларацията му, но не е част от сигнатурата му.

3. Полета и методи

Имплементация (създаване) на метода, е реалното написване на кода, който решава конкретната задача, заради която се създава метода. Този код се съдържа в самия метод.

Извикване е процесът на стартиране на изпълнението, на вече декларирания и създаден метод от друго място на програмата, където трябва да се реши проблемът, който метод решава.

Параметрите на методите могат да бъдат *примитивни типове* и *референтни типове*. Не може да се предават методи като параметри, но може да се предаде обект и да се извикат неговите методи.

Предефинирани методи

3. Полета и методи

Модификатори за достъп

Модификатор се нарича ключова дума, с помощта, на която се дава допълнителна информация на компилатора за кода, за който се отнася модификатора. В Java има три модификатора за достъп - **public**, **protected** и **private**.

❖ ***public*** е специален вид модификатор, наречен модификатор за достъп (access modifier). Той се използва, за да укаже, че извикването на метода може да става от кой да е Java-клас, независимо къде се намира той.

❖ ***private*** - е противоположен на **public**, т.е. ако един метод бъде деклариран с модификатор за достъп **private**, то този метод не може да бъде извикан извън класа, в който е деклариран, дори този клас да се намира в същия пакет.

3. Полета и методи

❖ ***protected*** е специален вид модификатор за скрити полета и методи, които са достъпни само вътрешно от класа. Скриването на тези детайли гарантира, че никой освен самия клас няма да променя директно данните и така няма да има възможност да се сбърка нещо.

❖ ***default*** – прилага се, когато не се използва никакъв модификатор за достъп пред съответния елемент. То е по-ограничително от public-видимостта, тъй като позволява, да достъпваме съответният елемент, само от класове, които се намират в същия пакет, в който се намира класът, на който принадлежи елементът.

3. Полета и методи

Статични полета и методи

В ООП има специална категория полета и методи, които се асоциират с тип данни (клас), а не с конкретна инстанция (обект). Наричаме ги статични членове (static members), защото са независими от конкретните обекти. Те се използват, без да има създадена инстанция на класа, в който са дефинирани. Статичните членове в Java могат да бъдат полета, методи и конструктори.

Статичните полета се инициализират, когато типът данни (класът) се използва за пръв път по време на изпълнението на програмата.

Когато един метод притежава ключовата дума **static**, в декларацията си, наричаме метода статичен. Това означава, че този метод може да бъде извикан от кой да е друг метод, независимо дали другият метод е статичен или не.

```
public class Sequence {
    private static int currentValue = -1;          // Static field
                // Intentionally deny instantiation of this class
    private Sequence() {  }
    public static int nextValue() {                // Static method
                currentValue++;
                return currentValue;
        }
}
public class SequenceManipulating {
        public static void main(String[] args) {
            System.out.printf("Sequence[1..3]: %d, %d, %d%n",
                Sequence.nextValue(), Sequence.nextValue(),
                Sequence.nextValue());
}
```

Изход от програмата:
Sequence[1..3]: 0, 1, 2

OverlappingScopeTest.java

```
class OverlappingScopeTest {
        int myValue = 3;
        void printMyValue() {
                System.out.println("My value is: " + myValue);
        }
        public static void main(String[] args) {
            OverlappingScopeTest instance = new OverlappingScopeTest();
             instance.printMyValue();
        }
}
                void printMyValue() {
                // Defining new local variable with the same name
                        int myValue = 5;
                        System.out.println("My value is: " + myValue);
                }
```

My value is: 3

My value is: 5

4. Обекти

Всеки обект е представител на точно един клас и е създаден по шаблон на този клас. Създаването на обект от дефиниран клас наричаме инстанциране (instantiation). Инстанция (instance) е фактическият обект, който се създава от класа по време на изпълнение на програмата.

Всеки обект е инстанция на конкретен клас. Тази инстанция се характеризира със състояние (state) – множество от стойности, асоциирани с атрибутите на класа.

- Създаване на обекти от предварително дефинирани класове по време на изпълнението на програмата става чрез оператора new. Новосъздаденият обект обикновено се присвоява на променлива от тип, съвпадащ с класа на обекта (при това присвояване същинският обект не се копира). В променливата се записва само *референция* към новосъздадения обект (неговият адрес в паметта).

4. Обекти

```
Dog firstDog = new Dog(firstDogName);      // Assign dog name with a constructor
Dog secondDog = new Dog();
    secondDog.setName(input.nextLine());   // Assign dog name with a property
Dog thirdDog = new Dog();                  // Create a dog with a default name
```

При създаването на обект с оператора new се заделя памет за този обект и се извършва начална инициализация на член-променливите му. Инициализацията се осъществява от специален метод на класа, наречен конструктор.

4. Обекти

Разрушаването на обектите в Java чрез специална програма, наречена **garbage collector** във виртуалната машина, която се грижи за това вместо нас. Обектите, към които в даден момент вече няма референция в програмата автоматично се унищожават и паметта, която заемат се освобождава. По този начин се предотвратяват много потенциални бъгове и проблеми.

Ако искаме ръчно да освободим даден обект, трябва да унищожим референция към него, например:

firstDog= null;

Това не унищожава обекта веднага, но го оставя в състояние, в което той е недостъпен от програмата и при следващото включване на системата за почистване на паметта (garbage collector) той ще бъде освободен

5. Абстрактни класове

Ключовата дума abstract пред името на класа означава, че класът не е готов и не може да бъде инстанциран. Такъв клас се нарича **абстрактен** клас. А как да укажем коя точно част от класа не е пълна? Това отново става с ключовата дума **abstract** пред името на метода, който трябва да бъде имплементиран. Този метод се нарича абстрактен метод и не може да притежава имплементация, а само декларация.

Всеки клас, който има поне един абстрактен метод, се нарича абстрактен. Обратното, обаче не е в сила. Възможно е да дефинирам клас като абстрактен дори когато в него няма нито един абстрактен метод.

Абстрактните класове са нещо средно между клас и интерфейс. Те могат да дефинират обикновени методи и абстрактни методи. Обикновените методи имат тяло (имплементация), докато абстрактните методи са празни (без имплементация) и са оставени да бъдат реализирани от класовете-наследници.

Използвана литература:

1. Светлин Наков и колектив. Въведение в програмирането с Java. Софтуерен Университет, София, април 2017. ISBN 978-954-400-055-4
http://www.introprogramming.info

2. Bruce Eckel. Thinking in Java. 4th Edition, Pearson, 2006. ISBN-10: 0131872486, ISBN-13: 978-0131872486.

Въпроси ?