

Класове и интерфейси

Ненко Табаков, Пламен Танов, Любомир Чорбаджиев

Технологично училище “Електронни системи”
Технически университет, София

15 април 2009



Забележка: Тази лекция е адаптация на:

- Justin Mazzola Paluska: *Classes and Interfaces* from *6.092: Java for 6.170* (MIT OpenCourseWare: Massachusetts Institute of Technology)

Лиценз: Creative commons BY-NC-SA



Тези материали са разработени в рамките на проекта “Софтуерна академия “Електронни системи”, съфинансиран от Европейския съюз и Европейския социален фонд.

Съдържание

Терминология

- **class** — класовете могат да се разглеждат като описание, спецификация на обектите от дадения клас.
- **interface** — интерфейсът дефинира списъкът на достъпните методи
- **instance** — физическото представяне на даден клас или интерфейс в паметта
- **метод** — функция, която е дефинирана в рамките на класа
- **поле** — променлива, която е част от класа
- **статично поле** — променлива, която е една и съща за всички инстанции на класа

Типове данни в Java

В Java има два основни класа от типове данни:

- примитивни типове
- обекти

Обекти

- Обектът е инстанция (екземпляр) на даден клас. Всеки обект принадлежи на даден клас.
- За създаване на инстанция на клас се използва операторът **new**
- Операторът **new**:
 - Заделя място в паметта за новия обект
 - Извиква съответния конструктор и инициализира обекта
 - Връща препратка към новосъздадения обект

```
1 Bean bean=new Bean();
```

Използване на обекти

- Обектите позволяват извикване на техните методи:

```
1 public static void main(String[] args) {  
2     Bean bean = new Bean();  
3     bean.plantBean(); // Invoked on instance  
4 }
```

- Обектите позволяват достъп до техните полета:

```
1 public static void main(String[] args) {  
2     Point myPoint = new Point ();  
3     myPoint.x = 10;  
4     myPoint.y = 15;  
5 }
```

- Когато даден обект повече не ни трябва просто спираме да го използваме. Паметта заемана от него ще бъде освободена, когато няма повече препратки към него.

Дефиниране на клас

- Структурата на кода, необходим за дефиниране на клас, е следната:

```
1 [<access>] [abstract/final] class <class_name>
2     [extends <class_name>]
3     [implements <interface_name>, ...]{
4     //полеа
5     //конструктор
6     //методи
7 }
```

- Пример:

```
1 public class Point {
2     ...
3 }
```


Членове на класа

В рамките на класа могат да се дефинират следните видове членове на класа:

- конструктори
- полета (нестатични и статични)
- методи (нестатични и статични)
- вложени класове

Конструктор

- Конструкторът трябва да има същото име, като това на класа
- Един клас може да има няколко конструктора
- В конструктора се извършва инициализация на конструируания обект

```
1 [<access>] class_name ([<argument list>]){  
2     //тяло на конструктора  
3 }  
4  
5 public class Point {  
6     public Point () {  
7         ...  
8     }  
9     ...  
10 }
```

Пример: Конструктори

```
1 public class HelloWorld {  
2  
3     public HelloWorld (String helloMessage) {  
4         myString = helloMessage;  
5     }  
6  
7     public HelloWorld () {  
8         myString = "Hello, \u00a0World";  
9     }  
10  
11     public static void main(String[] args) {  
12         HelloWorld myHelloWorld=new HelloWorld();  
13         HelloWorld myHelloWorld2=new HelloWorld("Hello!!!");  
14     }  
15 }
```

Методи

- Методите описват поведението на класа. Те описват реакцията на обектите от класа на външни въздействия (съобщения). Методите извършват операции
- Методите работят върху състоянието (полетата) на класа
- Методите могат да имат произволен брой аргументи и могат връщат не повече от една стойност
- Ако даден метод не връща стойност, то неговият тип е `void`
- Един клас може да има произволен брой методи

```
1 [<access>] <result type> method_name ([<argument list>])  
2   //тяло на метода  
3 }
```

Пример: Методи

```
1 class Box {  
2     public boolean isEmpty() {  
3         ...  
4     }  
5  
6     public int numberOfBooks() {  
7         ...  
8     }  
9 }
```

Предефениране на методи (overloading)

- В един клас може да има два (или повече) метода с еднакво име стига аргументите да са им различни
- При извикване методът се избира на основата на името и списъка от реално предадените аргументи

```
1 void    foo () {  
2     ...  
3 }  
4 void    foo (int a) {  
5     ...  
6 }  
7 public  static void main (String[] args) {  
8     obj.foo(); //извиква първия метод  
9     obj.foo(7); //извиква втория метод  
10 }
```

Полета

- Полето е част от клас
- Полето е променлива – съдържа данни
- Всяко поле има тип, който определя какъв вид данни ще се записват в него

```
1 public class Bean {  
2     public int beanCounter = 0;  
3     public Date date;  
4  
5 }
```

Пример

```
1 public class BankAccount {  
2     private int balance;  
3  
4     public BankAccount() {  
5         balance = 0;  
6     }  
7  
8     public void withdraw(int amount) {  
9         balance = balance - amount;  
10    }  
11  
12    public void deposit(int amount) {  
13        balance = balance + amount;  
14    }  
15 }
```


Достъп до членовете на класа

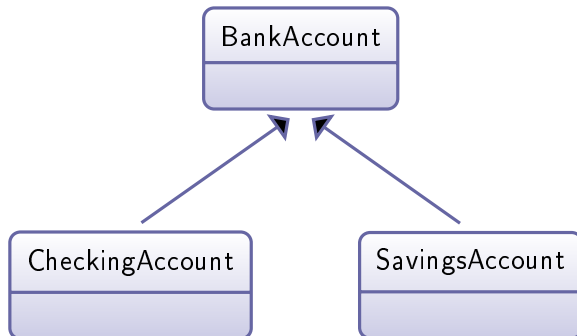
Достъпът до член на класа се дефинира с една от следните ключови думи:

- **public** – всеки клас от всеки пакет има достъп
- **protected** – всеки подклас (наследник) има достъп
- **(default)** – само класове от същия пакет имат достъп
- **private** – само съответния клас има достъп

Наследяване

- Чрез наследяване даден клас може да придобие функционалност от друг клас
- Чрез наследяване може да се постигне абстракция на функционалността и данните
- Наследяването позволява да се намали сложността на големи софтуерни системи

Наследяване



- Две отделни идеи с различно поведение, но имат базова функционалност която е обща

Интерфейси

- Интерфейсът дефинира списък на достъпните методи
- В интерфейс се декларират методи, но не се дефинират
- Интерфейсите нямат конструктори

```
1 interface BankAccount {  
2     public void withdraw(int amount);  
3     public void deposit(int amount);  
4 }
```

Използване на интерфейси

- Един клас може да имплементира един или няколко интерфейса
- Един интерфейс може да разшири друг интерфейс
- Ако един клас имплементира даден интерфейс, то този клас трябва да предостави реализация на всеки метод от интерфейса
- Ако един клас имплементира няколко интерфейса то този клас трябва да предостави реализация на всеки метод от всеки интерфейс

Пример: Употреба на интерфейси

```
1 public class CheckingAccount implements BankAccount {
2     private int balance;
3
4     public CheckingAccount(int initial) {
5         balance = initial;
6     }
7     // implemented methods from BankAccount
8     public void withdraw(int amount) {
9         balance = balance - amount;
10    }
11    public void deposit(int amount) {
12        balance = balance + amount;
13    }
14    public int getBalance() {
15        return balance;
16    }
17 }
```

Абстрактни класове

- Абстрактният клас е нещо средно между интерфейс и клас
 - може да има дефинирани методи
 - може да има полета
 - не могат да се създават обекти
- Помага да се дефинира една идея както като функционалност така и като данни
- В един абстрактен клас може да се разположат методи, които имат обща функционалност за всички подкласове
- Абстрактен клас се дефинира с ключовата дума `abstract`

Пример: Употреба на абстрактни класове

```
1 public abstract class BankAccount {  
2     protected int balance;  
3  
4     public int getBalance() {  
5         return balance;  
6     }  
7  
8     public void deposit(int amount) {  
9         balance = balance + amount;  
10    }  
11  
12    public abstract void withdraw(int amount);  
13 }
```


Пример: Наследяване на абстрактни класове

```
1 public class CheckingAccount extends BankAccount {  
2     public CheckingAccount () {  
3         balance = 0;  
4     }  
5  
6     public void withdraw (int amount) {  
7         balance = balance - amount;  
8     }  
9 }
```

Пример: Наследяване на абстрактни класове

```
1 public class SavingsAccount extends BankAccount {
2     private int numberOfWithdrawals;
3     public SavingsAccount() {
4         balance = 0;
5         numberOfWithdrawals = 0;
6     }
7     public void withdraw(int amount) {
8         if (numberOfWithdrawals > 5) {
9             throw new RuntimeException(
10                 "Cannot make >5 withdrawals a month");
11         } else {
12             balance = balance - amount;
13             numberOfWithdrawals++;
14         }
15     }
16     public void resetNumOfWithdrawals() {
17         numberOfWithdrawals=0;
18     }
19 }
```