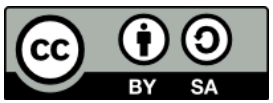
Интерфейси, абстрактни класове, изключения и вътрешни класове

Ненко Табаков, Пламен Танов, Любомир Чорбаджиев

Технологично училище "Електронни системи"
Технически университет, София

24 март 2009 г.



Забележка: Тази лекция е адаптация на:

- Lucy Mendel: *Interfaces, Abstract classes, Exceptions, Inner classes* from *6.092: Java for 6.170* (MIT OpenCourseWare: Massachusetts Institute of Technology)

Допълнителна литература


David Flanagan.

Java In A Nutshell, 5th Edition.

O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2005.


Pat Niemeyer and Jonathan Knudsen.

Learning Java, 2nd edition.

O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

Съдържание

Абстракни класове

- Дефинират се с ключовата дума abstract
- Не могат да се създават обекти от абстрактен клас
- Използват се, когато част от кода на два класа съвпада

```
1  public abstract class Person {
2    private String name = "";
3
4    public String getName() {
5      return name;
6    }
7    public void setName(String n) {
8      name = n;
9    }
10   abstract public String sayGreeting();
11 }
```

Абстракни класове

```
1  class EnglishPerson extends Person {
2    public String sayGreeting() {
3      return "Hello!";
4    }
5  }
6
7  class SpanishPerson extends Person {
8    public String sayGreeting() {
9      return "Hola!";
10   }
11 }
```

Интрефейси

- Интерфейсът дава списък на достъпните методи
- Дефинират се с ключовата дума **interface**
- Реализират се с ключовата дума **implements**
- Използват се за описание на начина на взаимодействие между различни или неизвестни компоненти

```
1 public interface Dragable {
2   public void drag();
3 }
4 public class Icon implements Dragable {
5   public void drag() { ... }
6 }
7 public class Chair implements Dragable {
8   public void drag() { ... }
9 }
```

Реализация на множество интерфейси

- Всеки един клас може да реализира произволен брой (множество) интерфейси

```
1  interface Drawable {
2    public void draw();
3  }
4
5  interface Clickable {
6    public void click();
7  }
8
9  interface Draggable {
10   public void drag();
11 }
```

Реализация на множество интерфейси

```
1  class Icon implements Drawable, Clickable, Draggable {
2    public void draw() {
3      System.out.println("drawing...");
4    }
5    public void click() {
6      System.out.println("clicking...");
7    }
8    public void drag() {
9      System.out.println("dragging...");
10   }
11 }
```

Единично наследяване

- Всеки един клас може да наследява само един клас
- Ако той не е указан непосредствено, то родител е `java.lang.Object`
- Възможно е предефиниране на методи

```
1  class Parent {
2    public String doSomething() {
3      return "Hello (Parent)";
4    }
5  }
6  class Child extends Parent {
7    public String doSomething() {
8      return "Hello (Child)";
9    }
10 }
```

Подтипове

```
1  class Square{
2    public int width;
3  }
4  class Rectangle{
5    public int width, height;
6  }
7
8  int calculateArea (Square x) {
9      return (x.width)*(x.width);
10 }
11 int calculateCircumference (Rectangle x) {
12     return 2*(x.width+x.height);
13 }
```

- Дали Square трябва да наследи Rectangle, или обратното Rectangle трябва да наследи Square?

Подтипове: Rectangle наследява Square

- Дали Square трябва да наследи Rectangle, или обратното Rectangle трябва да наследи Square?

```
1  class Square {
2     public int width;
3     Square( int x ) { width = x; }
4  }
5
6  class Rectangle extends Square {
7     public int height;
8     Rectangle( int width, int height ) {
9           super(width);
10          this.height = height; }
11 }
12 ...
13 Rectangle rect = new Rectangle( 2, 3 );
14 calculateArea( rect ) ;       // returns 4, not 6 !
```

Подтипове: Square наследява Rectangle

- Дали Square трябва да наследи Rectangle, или обратното Rectangle трябва да наследи Square?

```
1 class Rectangle {
2     public int width, height;
3 }
4 class Square extends Rectangle {
5     public int side;
6 }
7 ...
8 Square square = new Square( 3 );
9 calculateCircumference( sq ) ; // w.t.f. no height!
```

Подтипове: отново Square наследява Rectangle

- Дали Square трябва да наследи Rectangle, или обратното Rectangle трябва да наследи Square?

```
1  class Rectangle {
2     public int width, height;
3     Rectangle( int width, int height ) {
4        this.width = width; this.height = height; }
5  }
6  class Square extends Rectangle {
7      Square( int x ) { super( x, x ); }
8  }
9  ...
10 Square square = new Square( 3 );
11 calculateCircumference( sq ) ; // 12, ok
```

Подтипове

- Наследяването позволява повторно изплозване на кода на родителския клас в класа-наследник
- Класът наследник се превръща в подтип на базовия клас. За да бъде истински подтип наследникът трябва да се държи коректно, когато се използва от методи, които очакват екземпляр на базовия клас.
- При наследяване винаги трябва да се осигури правилно функциониране на наследника, когато той замества екземпляр на базовия клас

Наследяване срещу композиция

```
1 public class ListSet extends ArrayList {
2 ...
3 }
```

```
1 public class ListSet { // might want to implement Set
2   private List myList = new ArrayList();
3   public void add(Object o) {
4       if (!myList.contains(o)) myList.add(o);
5   }
6 ...
7 }
```

Изключения

- Помагат за обработката на възникнали изключителни ситуации
- Изключението не може просто да бъде изпуснато
- При възникване на изключения се прекъсва нормалното изпълнение на програмата и се търси код, който е предназначен за обработка на възникналото изключение

```
1  try {
2     // statement(s) that might throw exception
3  } catch (ExceptiontypeA name) {
4     // handle or report exceptiontypeA
5  } catch (ExceptiontypeB name) {
6     // handle or report exceptiontypeB
7  } finally {
8     // clean-up statement(s)
9  }
```

Пример: Обработка на изключения

```
1  class Editor {
2    boolean fileOpen = false;
3    public boolean openFile(String filename) {
4      try {
5        fileOpen = true;
6        File f = new File(filename);
7        // действия с f
8        ...
9        return true;
10     } catch (FileNotFoundException e) {
11       // изпълнява се само при генериране на изключение
12       e.printStackTrace();
13       return false;
14     } finally { // изпълнява се винаги:
15       fileOpen = false;
16     }
17   }
18 }
```

Генериране на изключения

- Използва се ключовата дума **throw**, следвана от обект, който да съхранява информация за причината за възникване
- При дефиницията на конструктор или метод се описват изключенията, които могат да възникнат при изпълнение посредством ключовата дума **throws**

```
1  public class File {
2    public File(String filename)
3          throws FileNotFoundException {
4      ...
5      // файлът не съществува
6      if ( /*  file not found */ ) {
7        throw new FileNotFoundException();
8      }
9      ...
10   }
11 }
```

Вложени класове

```
1 public class EnclosingClass {
2   ...
3   public class ANestedClass {
4     ...
5   }
6   ...
7 }
```

- Кога и защо се използват вложени класове?

Вложени класове

- Имат достъп до всички полета на външния клас (дори **private** полетата)
- Могат да бъдат статични (също така **final**, **abstract**). Когато вътрешният клас е статичен, той има достъп само до статичните полета на външният клас
- Не-статичните вложени класове се наричат още и вътрешни класове (inner classes)

```
1 class EnclosingClass {
2   static class StaticNestedClass {
3     // ...
4   }
5   class InnerClass {
6     // ...
7   }
8 }
```

Вътрешни класове

- Има достъп до всичките полета на съдържащия го клас
- Не могат да имат статични полета
- Не може да съществува без екземпляр на съдържащия го клас

Локални вътрешни класове

```
1  import  java.util.ArrayList;
2  import  java.util.Iterator;
3  public class Stack {
4    private ArrayList items;
5    public Iterator iterator() {
6      // клас дефиниран в тялото на метода
7      class StackIterator implements Iterator {
8        int currentItem = items.size() - 1;
9        public boolean hasNext() { /* ... */ }
10       public ArrayList<Object> next() { /* ... */ }
11       public void remove() { /* ... */ }
12     }
13     // видим е само в границите на самата функция
14     return new StackIterator();
15   }
16 }
```

Анонимни вътрешни класове

```
1  import  java.util.ArrayList;
2  import  java.util.Iterator;
3  public class Stack {
4    private ArrayList items;
5
6    public Iterator iterator() {
7      return new Iterator() {//реализира интерфейса Iterator
8        int currentItem = items.size() - 1;
9        public boolean hasNext() { /* ... */ }
10       public ArrayList<Object> next() { /* ... */ }
11       public void remove() { /* ... */ }
12     };
13   }
14 }
```

Вътрешни класове

```
1 public class Animal {
2   class Brain {
3     // ...
4   }
5   void performBehavior() {
6     Brain brain=new Brain();
7     // ...
8   }
9 }
```

```
1 Animal monkey = new Animal( );
2 Animal.Brain monkeyBrain = monkey.new Brain( );
```

Вътрешни класове като адаптери

```
1 public class EmployeeList {
2   private Employee[] employees = new Employee[0];
3   // ...
4   void add(Employee e) {...}
5   void remove(int index) {...}
6   //...
7 }
```

```
1 public interface Iterator {
2   boolean hasNext();
3   Object next();
4   void remove();
5 }
6 class EmployeeListIterator implements Iterator {
7   // Трябва да познава вътрешното устройство на EmployeeList
8   //...
9 }
```

Вътрешни класове като адаптери

```
1  public class EmployeeList {
2      private Employee[] employees = new Employee[0];
3      //...
4      class Iterator implements java.util.Iterator {
5          private int element = 0;
6          public boolean hasNext( ) {
7              return  element < employees.length ;
8          }
9          public Object next( ) {
10             if ( hasNext( ) )
11                 return employees[ element++ ];
12             else
13                 throw new NoSuchElementException( );
14         }
15         //...
```

Вътрешни класове като адаптери

```
1           public void remove( ) {
2                throw new UnsupportedOperationException( );
3           }
4       }
5       //...
6
7       Iterator getIterator() {
8           return new Iterator();
9       }
10  }
```

Локални класове

```
1 public class Animal {
2   public void performBehavior() {
3     class Brain {
4       // ...
5     }
6     //...
7   }
8 }
```

Специфики на локалните класове

```
1  public class Animal {
2    public void performBehavior(final boolean nocturnal){
3      class Brain {
4        ...
5        void sleep(){
6          if(nocturnal) {
7            ...
8          } else {
9            ...
10         }
11       }
12     }
13     ...
14   }
15 }
```

Анонимни класове

```
1  Iterator getIterator() {
2    return new Iterator() {
3      int element = 0;
4      public boolean hasNext() {
5        return element < employees.length;
6      }
7      public Object next() {
8        if (hasNext())
9          return employees[element++];
10       else
11         throw new NoSuchElementException();
12     }
13     public void remove() {
14       throw new UnsupportedOperationException();
15     }
16   };
17 }
```

Област на видимост на this

```
1 class Animal {
2   int size;
3   class Brain {
4     int size;
5   }
6 }
```

```
1 class Brain {
2   Animal ourAnimal = Animal.this;
3   int animalSize = Animal.this.size;
4 }
```