КЛАСОВЕ И ИНТЕРФЕЙСИ

Ненко Табаков
Пламен Танов
Технологическо училище “Електронни системи”
Технически университет – София

24 септември 2008

КЛАСОВЕ И ИНТЕРФЕЙСИ

Забележка: Тази лекция е адаптация на лекция от курса:

•6.092 Java Preparation for 6.170, Януари 2006
- Lucy Mendel
- Corey McCaffrey
- Rob Toscano
- Justin Mazolla Paluska
- Scott Osler
- Ray He

Интернет адрес:
http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-092January--IAP--2006/CourseHome/index.htm

ОСНОВНИ ТЕРМИНИ

- ***class*** – класовете описват обекти
- ***interface*** **–** интерфейсът дава списък на достъпните методи
- ***instance*** – физическото представяне на даден клас или интерфейс в паметта

- **метод** – функция, която е дефинирана в класа
- **поле –** променлива, която е част от класа
- **статично поле** – променлива, която е една и съща за всички инстанции на класа

ТИПОВЕ ДАННИ

Типове данни:

- примитивни типове
- обекти

ОБЕКТИ

- Обектът е инстанция на клас
- За създаване на инстанция на клас се използва операторът ***new***
- Операторът ***new:***
  - Заделя място в паметта за новия обект
  - Извиква съответния конструктор
  - Връща препратка към новия обект

ПРИМЕР
инстанциране на клас

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

име на класа име на обекта конструктор

УПОТРЕБА НА ОБЕКТИ

Чрез обект могат да се извикват методи

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

УПОТРЕБА НА ОБЕКТИ

Чрез обект могат да се достигат член-променливи (полета)

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

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

ДЕФИНИЦИЯ НА КЛАС

Шаблонът за дефиниране на клас изглежда по следния начин:

```
[достъп][abstract/final] class име_на_класа
        extends име_на_клас
        implements име_на_интерфейс, ...{
    //конструктор
    //методи
    //полета
}



public class Point {
    ...
}
```

ЧЛЕНОВЕ НА КЛАСА

Един клас може да има следните членове:

- конструктор
- член-променливи (и статични)
- методи (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 класа

```
[достъп]име_на_класа ([аргументи]){
    //тяло на конструктора
}

public class Point {
    public Point () {
        ...
    }
    ...
}
```

ПРИМЕР конструктори

```
public  class HelloWorld {

   public String myString;

   public HelloWorld (String helloMessage) {
      myString = helloMessage;
   }

   public HelloWorld () {
      myString = "Hello, World";
   }

   public static void main(String[] args) {
      HelloWorld myHelloWorld = new HelloWorld();
      HelloWorld myHelloWorld2 = new HelloWorld("Hello!!!");
   }
}
```

МЕТОДИ

- Методите извършват операции
- Методите работят върху състоянието на класа
- Методите могат да имат произволен брой аргументи и връщат максимум една стойност
- Ако даден метод не връща стойност, то неговият тип е ***void***
- Един клас може да има произволен брой методи

```
[достъп]тип име_на_метода ([аргументи]){
   //тяло на метода
}
```

ПРИМЕР
методи

```
class Box {
    public boolean isEmpty() {
        ...
    }

    public int numberOfBooks() {
        ...
    }
}
```

ПРЕДЕФИНИРАНЕ НА МЕТОДИ *OVERLOADING*

- В един клас може да има два метода с еднакво име стига аргументите да са им различни
- Методът се извиква на базата на името му плюс аргументите му

```
void  foo () {
}

void  foo (int a) {
}

public  static void main (String[] args) {
   obj.foo(); //извиква първия метод
   obj.foo(7); //извиква втория метод
}
```

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

```
[достъп]тип име_на_променлив [= стойност];

public  class Bean {
   public int beanCounter = 0;
   public Date date;

}
```

ПРИМЕР

```
public class BankAccount {
   private int balance;

   public BankAccount() {
      balance = 0;
   }

   public void withdraw(int amount) {
      balance = balance - amount;
   }

   public void deposit(int amount) {
      balance = balance + amount;
   }
}
```

ДОСТЪП

- Достъпът се определя с една от четирите ключови думи
  - ***public*** – всеки клас от всеки пакет има достъп
  - ***protected*** – всеки подклас има достъп
  - ***(default)*** – само класове от същия пакет имат достъп
  - ***private*** – само съответния клас има достъп

НАСЛЕДЯВАНЕ

- Чрез наследяване даден клас може да наследи функционалност от друг клас
- Чрез наследяване може да се постигне по-добра абстракция на функционалността и данните
- Наследяването намалява сложността на големи софтуерни системи

НАСЛЕДЯВАНЕ

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

ИНТЕРФЕЙСИ

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

Шаблонът за интерфейс изглежда по следния начин:

```
[достъп]interface име_на_интерфейса
      extends име_на_интерфейс, ... {
   //методи
}

interface BankAccount {
   public void withdraw(int amount);
   public void deposit(int amount);
}
```

УПОТРЕБА 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 (тяло на методите)
- Ако един клас имплементира няколко интерфейса то този клас трябва да предостави реализация на всеки метод от всеки интерфейс

ПРИМЕР употреба на интерфейси

```
public  class CheckingAccount implements BankAccount {
   private int balance;

   public CheckingAccount(int initial) {
      balance = initial;
   }
   // implemented methods from BankAccount
   public void withdraw(int amount) {
      balance = balance - amount;
   }

   public void deposit(int amount) {
      balance = balance + amount;
   }

   public int getBalance() {
      return balance;
   }
}
```

АБСТРАКТНИ КЛАСОВЕ

- Абстрактният клас е нещо средно между интерфейс и клас
  - може да има дефинирани методи
  - може да има полета
- 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 абстрактен клас

```
public  abstract class BankAccount {
   protected int balance;

   public int getBalance() {
      return balance;
   }

   public void deposit(int amount) {
      balance = balance + amount;
   }

   public abstract void withdraw(int amount);
}
```

ПРИМЕР
наследяване на клас

```
public  class CheckingAccount extends BankAccount {
    public CheckingAccount () {
        balance = 0;
    }

    public void withdraw (int amount) {
        balance = balance - amount;
    }
}
```

ПРИМЕР
наследяване на клас

```
public  class SavingsAccount extends BankAccount {
    private int numberOfWithdrawals;

    public SavingsAccount() {
        balance = 0;
        numberOfWithdrawals = 0;
    }

    public void withdraw(int amount) {
        if (numberOfWithdrawals > 5) {
            throw new RuntimeException("Cannot make >5 withdrawals a
month");
        } else {
            balance = balance - amount;
            numberOfWithdrawals++;
        }
    }

    public void resetNumOfWithdrawals() {
    }
}
```

ПОЛИМОРФИЗЪМ

- Свойството на обектите да реагират по собствен начин в зависимост от типа си, на извикване на един и същ метод (същото име на метод)
- Свойството на обектите от различен тип да реагират на методи с едно и също име
- Свойството да се предефинира функционалност чрез наследяване

КЛАСЪТ ***Object***

- Всеки клас е наследник на ***java.lang.Object*** класа
- ***java.lang.Object*** съдържа методи, които се наследяват от всеки клас :
  - ***clone***
  - ***equals***
  - ***finalize***
  - ***getClass***
  - ***hashCode***
  - ***notify***
  - ***notifyAll***
  - ***toString***
  - ***wait***

ПРЕДЕФИНИРАНЕ НА МЕТОДИ ***OVERRIDING***

- Родителски клас
  - ако клас А наследява клас Б, то клас Б е родителски клас на клас А
  - съответно клас А е подклас на клас Б
- Ако клас Б съдържа метод, до който клас А има достъп, то клас А може да предефинира този метод

ТИП ПО ВРЕМЕ НА КОМПИЛАЦИЯ И ТИП ПО ВРЕМЕ НА ИЗПЪЛНЕНИЕ

- Тип по време на компилация
  - Тип, който се знае предварително – знае се по време на писане на кода – по време на компилация
  - По време на работа на приложението, типът по време на компилация никога не се променя за дадена инстанция
- Тип по време на изпълнение
  - Компилаторът не знае какъв тип ще е даден обект по време на работа на приложението

```
Object int1 = new Integer (10);
//compile type          //runtime type
Object int2 = new Integer (100);
int1.equals(int2);
```

ТИП ПО ВРЕМЕ НА КОМПИЛАЦИЯ И ТИП ПО ВРЕМЕ НА ИЗПЪЛНЕНИЕ

- Типът по време на компилация на ***int1*** е ***Object***, но все пак се извиква ***equals*** метода на класа ***Integer***
- Това се случва, защото се извиква метода на базата на типа по време на изпълнение

```
Object int1 = new Integer (10);
//compile type        //runtime type
Object int2 = new Integer (100);
int1.equals(int2);
```

ПРИМЕР полиморфизъм

```
public  abstract class BankAccount {
    ...
    public abstract void withdraw(int amount);
    ...
}

public  class CheckingAccount extends BankAccount {
    ...
    public void withdraw (int amount) {
       balance = balance - amount;
    }
    ...
}
```

ПРИМЕР полиморфизъм

```
public  class SavingsAccount extends BankAccount {
    ...
    public void withdraw(int amount) {
       if (numberOfWithdrawals > 5) {
          throw new RuntimeException("Cannot make >5
withdrawals a month");
       } else {
          balance = balance - amount;
          numberOfWithdrawals++;
       }
    }
```

ПРИМЕР полиморфизъм

```
public  static void main (String[] args) {
   BankAccount b1 = new CheckingAccount(10);
   BankAccount b2 = new SavingsAccount(10);
   b1.withdraw(5);
   //calls CheckingAccount.withdraw(int)
   b2.withdraw(5);
   //calls SavingsAccount.withdraw(int)
}
```

ПРИМЕР аргументи

В този случай може да се предаде обект от тип ***CheckingAccount*** или ***SavingAccount*** и компилаторът няма да разбере

```
public void foo (BankAccount account) {
   account.withdraw(5);
}
```

ПРЕДИМСТВА ПРИ УПОТРЕБА НА РОДИТЕЛСКИ ТИПОВЕ

- Може да се промени имплементацията на по – късен етап
- Не се налага да се променя кодът, защото се използват функции, които са общи (не са дефинирани в някой от подкласовете)
- Пример: имплементацията на метода ***get(int n)*** в класовете ***ArrayList*** и ***LinkedList***