# Полиморфизъм

Ненко Табаков, Пламен Танов, Любомир Чорбаджиев

Технологично училище “Електронни системи”
Технически университет, София

23 март 2009 г.

**Забележка:** Тази лекция е адаптация на:

- Robert Toscano: *Polymorphism* from *6.092: Java for 6.170* (MIT OpenCourseWare: Massachusetts Institute of Technology)
  **Лиценз:** Creative commons BY-NC-SA

# Съдържание

# Полиморфизъм

- Способността на обектите да реагират по собствен начин на един и същ метод в зависимост от действителния си тип
- Способността на обекти от различен тип да реагират на методи с едно и също име
- Възможността да се предефинира функционалност на наследения клас
- Java управлява коя предефинирана версия на даден метод трябва да бъде извикана.

# Клас `Object`

- Всеки базов клас (т.е. клас, който не наслядява друг клас) по подразбиране наследява класа `java.lang.Object`.
- Класът `java.lang.Object` съдържа методи, които се наследяват от всички други класове.
- Методите на класа `java.lang.Object` включват: `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `toString`

# Предефиниране (override) на методи (заместване на методи)

- Предефинирането (override) на методи се различава от предефинирането (overloading) на методи.
- Родителски клас (супер клас)
  - Ако класът `D` наследява класа `B`, то класът `B` е родителски клас (базов клас, супер клас) на класа `D`
  - Класът `D` е наследник (подклас, производен клас) на класа `B`.
- Ако класът `B` съдържа метод, който е достъпен в класа `D`:

```
1 public void foo(int arg) {...}
```

  то класът `D` може да предефинира (замести, override) този метод, като дефинира метод със същата сигнатура.

# Методът `equals`

- Методът `equals` е дефиниран в базовият клас `java.lang.Object`:

```
1 public boolean equals(Object o);
```

- Всички класове наследяват този метод от класа `java.lang.Object`.
- Дефиницията на този метод в базовия клас `java.lang.Object` проверява равенството на препратките – т.е. проверяв дали две препратки сочат към един и същ обект в паметта.
- В случай, че класът трябва да поддържа идеята за сравняване на различни инстанции, то класът трябва да предефинира (замести, override) този метод.

# Пример: използване на `equals`

```
1 CheckingAccount c1 = new CheckingAccount(100);
2 CheckingAccount c2 = new CheckingAccount(100);
3 c1.equals(c1); //== true
4 c2.equals(c2); //== true
5 c1.equals(c2); //== false
```

# Предефиниране (заместване) на `equals`

```
1  public class CheckingAccount extends BankAccount {
2          ...
3          public boolean equals (Object o) {
4              if (o instanceof CheckingAccount) {
5                      CheckingAccount c = (CheckingAccount)o
6                      return balance == c.balance;
7              } else {
8                      return false;
9              }
10         }
11         ...
12 }
```

# Използване на `equals`

```
1 CheckingAccount c1 = new CheckingAccount(100);
2 CheckingAccount c2 = new CheckingAccount(100);
3 c1.equals(c1); //== true
4 c2.equals(c2); //== true
5 c1.equals(c2); //== true
```

# Полиморфно поведение на `equals`

```
1 Object o1 = new CheckingAccount(100);
2 Object o2 = new CheckingAccount(100);
3 o1.equals(o1); //== true
4 o2.equals(o2); //== true
5 o1.equals(o2); //== true
```

# Тип по време на компилация и тип по време на изпълнение

- Тип по време на компилация
  - Тип на обекта, който е известен предварително — по време на писане и компилиране на приложението.
  - Този тип на обекта не се променя по време на изпълнение на приложението
- Тип по време на изпълнение
  - Компилаторът не може да знае какъв ще бъде действителния тип на обекта по време на изпълнение на приложението – той може да бъде произволен подтип на типа по време на компилация.

```
1 Object o1 = new CheckingAccount(100);
2 Object o2 = new CheckingAccount(100);
3 o1.equals(o1); //== true
4 o2.equals(o2); //== true
5 o1.equals(o2); //== true
```

# Полиморфно поведение на метод

- Типът по време на компилация на обектите `o1` и `o2` е `java.lang.Object`.
- При изпълнение на кода се извиква метода `equals` на класа `CheckingAccount`.
- Изборът на метод се прави на базата на типа по време на изпълнение
- Изборът на метод в зависимост от типа по време на изпълнение се нарича динамично свързване на методи.

```
1 Object o1 = new CheckingAccount(100);
2 Object o2 = new CheckingAccount(100);
3 o1.equals(o1); //== true
4 o2.equals(o2); //== true
5 o1.equals(o2); //== true
```

# Пример: базов абстрактен клас

```
1 public abstract class BankAccount {
2         ...
3         public abstract void withdraw (int amount);
4         ...
5 }
```

- Класовете наследници — `CheckingAccount` и `SavingsAccount` наследяват `BankAccount` и предефинират (заместват, override) метода `withdraw()`.

# Пример: полиморфно поведение на метод

```
1 BankAccount b1 = new CheckingAccount(10);
2 BankAccount b2 = new SavingsAccount(10);
3
4 b1.withdraw(5);
5 //calls CheckingAccount.withdraw(int)
6
7 b2.withdraw(5);
8 //calls SavingsAccount.withdraw(int)
```

# Пример: предаване на аргументи

```
1 public void chargeAccount (BankAccount account) {
2   ...
3   account.withdraw(5);
4   ...
5 }
```

- При дефиниране на методи се предпочита използването на типа на възможно най-базовият клас
- Като действителен аргумент на метода може да се предаде обект от типа `CheckingAccount` и типа `SavingsAccount`.