JAVA

Modifikatori *revisited*

Modifikatori vidljivosti i prepisivanje metoda

- Vidljivost metoda u dete klasi kojim se prepisuje metod roditeljske klase može imati **istu ili veću vidljivost u odnosu na prepisani metod iz roditeljske klase**.

  Drugim rečima, prepisani metod u dete klasi ne sme da bude manje dostupan/vidljiv od istog u roditeljskoj.

  Npr. metod koji svaka Javina klasa ima, jer ga nasleđuje od `Object` klase je

  ```
  public String toString()
  ```

  ne može u prepisanoj verziji imati bilo koju drugu vidljivost jer bi svaka druga bila restriktivnija od `public` vidljivosti.

Prepisivanje statičkih metoda

```
public class Animal {
    public static void testClassMethod() {
        System.out.println("The class" + " method in Animal.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance " + " method in Animal.");
    }
}
public class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The class method" + " in Cat.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method" + " in Cat.");
    }
}
public class Test {
    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        Animal.testClassMethod();          // stampa: The class method in Animal.
        myAnimal.testInstanceMethod();     // stampa: The instance method in Cat.
    }
}
```

Prepisivanje statičkih metoda

- Dakle, polimorfizma nema kod statičkih metoda. Zašto?

  Vezivanje poziva sa kodom metoda koji pripadaju objektima se vrši **dinamički** (late binding), tj. u vreme izvršavanja.
  Vezivanje poziva sa kodom statičkih metoda se vrši **statički** (early binding) , tj. u vreme kompajliranja.

Nadklasa / Podklasa	Metod objekta	`static` metod
Metod objekta	prepisuje	compile-time greška
`static` metod	compile-time greška	sakriva

`public`	Classes Interfaces	Constructors Methods Field variables	
`protected`		Constructors Methods Field variables	
`private`		Constructors Methods Field variables	
`no modifier`	Classes Interfaces	Constructors Methods Field variables	
`static`		~~Constructors~~ Methods Field variables	Code block
`final`	Classes	~~Constructors~~ Methods Field variables	
`abstract`	Classes Interfaces	~~Constructors~~ Methods ~~Field variables~~	

static blokovi koda

```
class StaticCodeExample {
    static int counter=0;
    static {
          counter++;
          System.out.println("Static Code block: counter: " + counter);
    }
    StaticCodeExample() {
          System.out.println("Constructor: counter: " + counter);
    }
    static {
          System.out.println("This is another static block");
    }
}
public class RunStaticCodeExample {
    public static void main(String[] args) {
    StaticCodeExample sce = new StaticCodeExample();
          System.out.println(“main: sce”);
    StaticCodeExample sce1 = new StaticCodeExample();
          System.out.println("main: sce1”);
    }
}
```

static blokovi koda se izvršavaju tačno jednom i to pri učitavanju definicije klase.

izlaz

```
Static Code block: counter: 1
This is another static block
Constructor: counter: 1
main: sce
Constructor: counter: 1
main: sce1
```

import static

- `import` omogućava upotrebu naziva klase bez navođenja imena paketa kom klasa pripada.
- `import static` omogućava upotrebu statičkih članova bez navođenja naziva klase kojoj pripadaju

uvoz `static` člana

```
import static java.lang.Math.PI;
class Sphere {
        double volume() {
                return 4.0/3.0*PI*radius*radius*radius;
        }
}
```

umesto

```
class Sphere {
        double volume() {
                return 4.0/3.0*Math.PI*radius*radius*radius;
        }
}
```

uvoz svih `static` članova

```
import static java.lang.Math.*;
```