Predavanje 9.a

OBJEKTNO-ORIJENTISANO PROGRAMIRANJE

2024/25.

Agenda

Modifikatori

Ispitivanje pripradnosti tipu

Refleksija

Modifikatori

AGENDA

Modifikatori

Ispitivanje pripradnosti tipu

Refleksija

Na šta se modifikatori mogu primeniti?

`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

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

```
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

```
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.*;
```

Ispitivanje pripradnosti tipu

AGENDA

Modifikatori

Ispitivanje pripradnosti tipu

Refleksija

instanceof operator

```
class Parent {
}
class Child extends Parent {
}


class Test {
    public static void main(String[] args)
    {
        Parent cobj = new Child();
        if (cobj instanceof Child)
            System.out.println("cobj is instance of Child");
        else
            System.out.println(
                "cobj is NOT instance of Child");
    }
}
```

cobj is instance of Child

instanceof operator

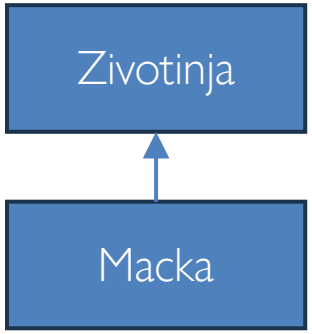
```
class Parent {
}
class Child extends Parent {
}


class Test {
    public static void main(String[] args)
    {
        Parent cobj = new Parent();
        if (cobj instanceof Child)
            System.out.println("cobj is instance of Child");
        else
            System.out.println(
                "cobj is NOT instance of Child");
    }
}
```

pobj is NOT instance of Child

instanceof operator

```
public class Test {
public static void main(String[] args) {
    Macka m = new Macka();
    Zivotinja z = new Zivotinja();
    System.out.println(z instanceof Zivotinja); // true
    System.out.println(m instanceof Zivotinja); // true
    System.out.println(z instanceof Macka); // false
    System.out.println(m instanceof Macka); // true
    z = m;
    System.out.println(z instanceof Zivotinja); // true
    System.out.println(z instanceof Macka); // true
    }
}
```



Class objekat

- Reprezentuje klase u Javi.
  - I primitivni tipovi (int, double, char, ...) su reprezentovani Class objektima – imaju svoje wraper-ske klase.
- Ima privatni konstruktor
  - jer su objekti kreirani od strane JVM-a, koja poziva metodu defineClass apstraktne klase java.lang.ClassLoader.
- Neki Metodi
  - **`public static Class forName( String className)-`** vraća Class objekat za naziv klase ili baca izuzetak
  - **`public Object newInstance()`** - kreira objekat za tip koji predstavlja.
  - **`public Class getSuperclass()`** - vraća nadklasu, a ako je primitivni tip, polje ili Object vraća null.
  - **`public String getName()`** - vraća naziv Class objekta.

Kako do Class objekta

1. Pomoću statičke metode forName klase Class:

```
Class a = Class.forName("Test");
```

2. Pomoću metode getClass nasleđene iz klase Object:

```
Class a = ts.getClass();
```

3. Korišćenjem polja class same klase

```
Class a = ImeKlase.getClass();
```

getClass()

```
public class Test {
public static void main(String[] args) {
    Macka m = new Macka();
    Zivotinja z = new Zivotinja();
    proveriClass(m); proveriInstance(m); // m je Zivotinja? m je tipa Zivotinja? false true
    proveriClass(z); proveriInstance(z); // z je Zivotinja? z je tipa Zivotinja? true true
    z = m;
    proveriClass(z); proveriInstance(z); // z je Zivotinja? z je tipa Zivotinja? false true
    proveriClass(z,"Macka"); proveriClass(z,"Zivotinja"); // z je Macka? z je Zivotinja? T F
}

public static void proveriClass(Zivotinja z) {
        System.out.println(z.getClass() == Zivotinja.class);
}
public static void proveriInstance(Zivotinja z) {
        System.out.println(z instanceof Zivotinja);
}
public static void proveriClass(Object z, String s) {
    try {
        System.out.println(z.getClass() == Class.forName(s));
    } catch (ClassNotFoundException e) { e.printStackTrace();} }
}
```

Refleksija

AGENDA

Modifikatori

Ispitivanje pripradnosti tipu

Refleksija

Refleksija

- Sposobnost programa da pregleda, ispita i modifikuje sopstvenu strukturu i ponašanje u toku izvršavanja.
- Refleksija u Javi - ispitivanje i modifikacija ponašanja metoda, klasa i interfejsa u toku izvršavanja programa. Potrebne klase se nalaze u java.lang.reflect paketu i omogućavaju:
  - dobijanje informacija o klasi kojoj pripada prosleđeni objekat i metodama koje može da izvrši
  - moguće pozivanje metoda bez obzira na njihovu vidljivost.
- Koristi se za:
  - Analiziranje mogućnosti klasa tokom izvršavanja
  - Istraživanje objekata tokom izvršavanja
  - …

java.lang.reflect

- Klase:
  - java.lang.reflect.Field - repezentuje atribute članice,
  - java.lang.reflect.Method - reprezentuje metode članice,
  - java.lang.reflect.Constructor – reprezentuje konstruktore.
- Ako postoji objekat klase Class, onda se objekti tipa Field, Method i Constructor dobijaju pomoću sledeće tri metode klase:
  - getFields() - daje atribut javnih atributa članica;
  - getMethods() - daje atribut javnih metoda članica;
  - getConstructors() - daje atribut javnih konstruktora.

Upotreba - naivna

```
import java.lang.reflect.Method;

public class Test {
public static void main(String[] args) {
    Macka m = new Macka();
    Zivotinja z = new Zivotinja();
        for(Method met : z.getClass().getMethods())
                System.out.println(met.getName());
    }
}

class Zivotinja {
        String ime;
    void jedi() { System.out.println("Zivotinja: jedem nesto");    }
}

class Macka extends Zivotinja {
        int dnevnoGrami;
        void jedi() { System.out.println("Macka: jedem samo meso");    }
    void mjauci() { System.out.println("Mjau");     }
}
```