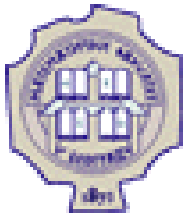
Објектно орјентисано програмирање

Владимир Филиповић

vladaf@matf.bg.ac.rs

Математички факултет

vladaf@matf.bg.ac.rs

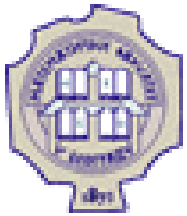


Рефлексија (самоиспитивање)

Библиотека за рефлексију обезбеђује веома богат скуп алата за писање програма који манипулишу Јава кодом на динамичан начин.

Рефлексија се може користи за:

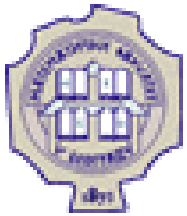
- Анализирање могућности класа током извршавања;
- Истраживање објеката током извршавања;
- Имплементацију генеричког кода за манипулацију са низовима;
- Коришћење примерака класе Method који раде на сличан начин као што у језику C++ раде показивачи на функцију.



Рефлексија (2)

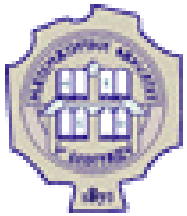
- Програм који анализира могућности класа назива се рефлексивни програм.
- Рефлексија је уведена у Јаву почев од верзије 1.1.
- Сваки елеменат је или примитивног типа или референтног (објектног) типа.
  Сви референтни типови наслеђују класу java.lang.Object.
- Класе, енумератори, низови и интерфејси су референтног типа.
- Постоји фиксиран скуп примитивних типова:
  boolean, byte, short, int, long, char, float и double.

Математички факултет
vladaf@matf.bg.ac.rs



Испитивање типа

- За сваки претходно побројани елемент тј. тип,
  Јава виртуална машина формира немутирајући примерак класе java.lang.Class
- Он обезбеђује методе за истраживање run-time особина објекта, укључујући информације о пољима, методама и типовима.
- Примерак класе Class такође обезбеђује и могућност да се „у лету“ креирају нове класе и објекти.



Испитивање типа (2)

Референца на објекат типа Class се може добити позивом метода getClass над примерком дате класе:

```
Class c = mystery.getClass();
```

Алтернативно, до ње се може доћи коришћењем поља class над самом класом:

```
Class c = MysteryClass.class;
```

Или позивом метода forName коме је прослеђено име класе:

```
Class c = Class.forName("MysteryClass");
```

Референца на објекат типа Class која представља надкласу датог Class објекта:

```
Class s = c.getSuperclass();
```

Одређивање имена класе:

```
String s = c.getName();
```

Одређивање интерфејса који имплементира дата класа:

```
Class[] interfaces = c.getInterfaces();
```

Одређивање поља дате класе:

```
Field[] fields = c.getFields();
```

Одређивање метода дате класе:

```
Method[] methods = c.getMethods();
```

Испитивање типа (3)

Пример.

```
Class c = "foo".getClass();
Class c2 = System.console().getClass();
enum E { A, B };
Class c3 = A.getClass();
byte[] bytes = new byte[1024];
Class c4 = bytes.getClass();
```

Пример.

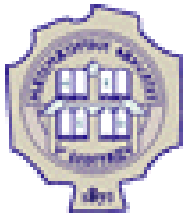
```
boolean b;
Class c = b.getClass();   // compile-time error
Class c2 = boolean.class;  // correct
```

Пример.

```
Class c = Class.forName("com.duke.MyLocaleServiceProvider");
Class cDoubleArray = Class.forName("[D");  // double[].class
Class cStringArray = Class.forName("[[Ljava.lang.String;");
```

Математички факултет

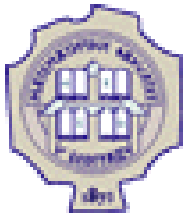
vladaf@matf.bg.ac.rs



Испитивање типа (4)

Пример. Илуструје како испитати које класе наслеђује и интерфејсе имплементира дата класа.

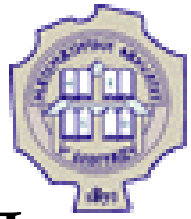
```
public static void showType(String className) throws ClassNotFoundException {
  Class thisClass = Class.forName(className);
  String flavor = thisClass.isInterface() ? "interface" : "class";
  System.out.println(flavor + " " + className);
  Class parent = thisClass.getSuperclass();
  if (parent != null) {
    System.out.println("extends " + parent.getName());
  }
  Class[] interfaces = thisClass.getInterfaces();
  for (int i=0; i<interfaces.length; ++i) {
    System.out.println("implements " + interfaces[i].getName());
  }
}
```



Испитивање типа (5)

Приликом извршавања претходног примера добијају се следећи резултати:

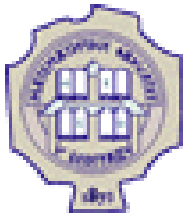
```
class java.lang.Object

class java.util.HashMap
  extends java.util.AbstractMap
  implements java.util.Map
  implements java.lang.Cloneable
  implements java.io.Serializable

class Point
  extends java.lang.Object
```



Испитивање типа (6)

Пример. Илуструје како испитати које методе садржи дата класа.

```
static void showMethods(Object o) {
   Class c = o.getClass();
   Method[] theMethods = c.getMethods();
   for (int i = 0; i < theMethods.length; i++)
   {
      String methodString = theMethods[i].getName();
      System.out.println("Name: " + methodString);
      System.out.println(" Return Type: " + theMethods[i].getReturnType().getName());
      Class[] parameterTypes = theMethods[i].getParameterTypes();
      System.out.print(" Parameter Types:");
      for (int k = 0; k < parameterTypes.length; k ++)
      {
          System.out.print(" " + parameterTypes[k].getName());
      }
      System.out.println();
   }
}
```



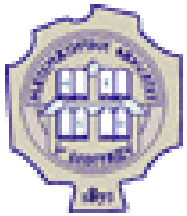
Испитивање типа (7)

Позивом претходно дефинисаног метода, тј извршењем кода:

```
Polygon p = new Polygon();
showMethods(p);
```

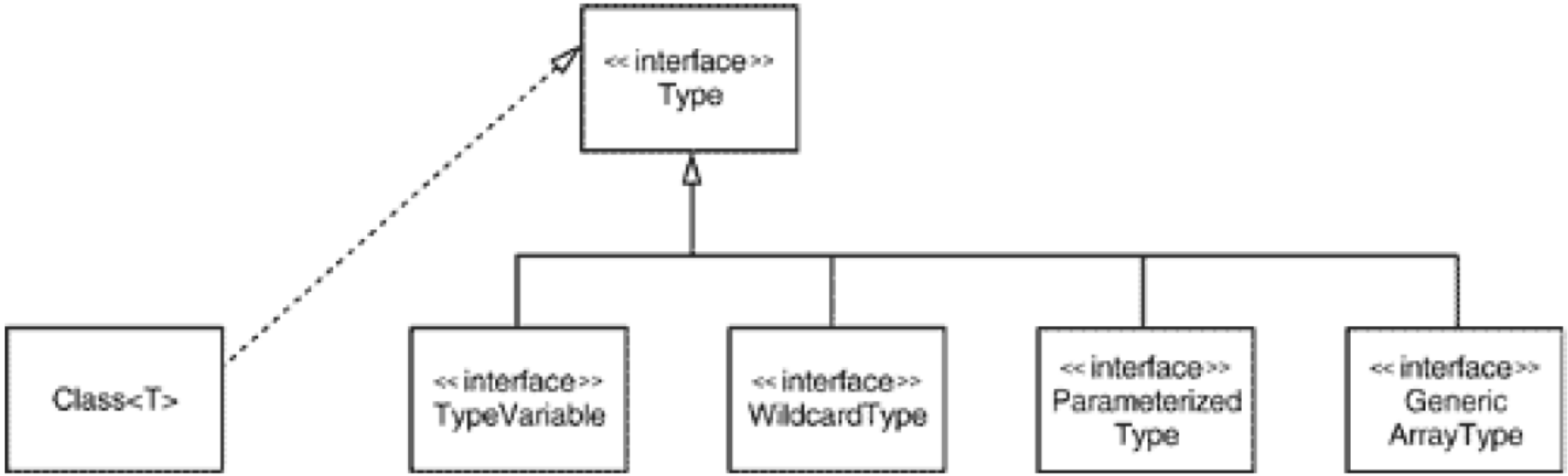
Добија се излаз следећег облика:

```
Name: equals
  Return Type: boolean
  Parameter Types: java.lang.Object
Name: getClass
  Return Type: java.lang.Class
  Parameter Types:
Name: intersects
  Return Type: boolean
  Parameter Types: double double double double
```

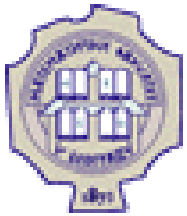


Класе за рефлексију

Хијерархија класа и интерфејса које се односе на типове.



Математички факултет
vladaf@matf.bg.ac.rs

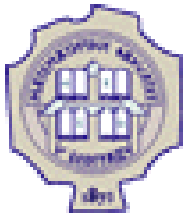


Класе за рефлексију (2)

java.lang.Class

Примерци класе Class представљају класе и интерфејсе у Јава апликацији која се извршава. Дакле, тип сваког објекта кријаног током рада Јава апликације представљен је са примерком класе Class.

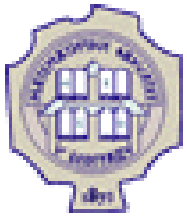
- static Class forName(String className)
  враће Class објекат који представља класу са именом className.
- Object newInstance()
  враће нови примерак класе описане датим Class објектом.
- Field[] getFields()
- Field[] getDeclaredFields()
  getFields враће низ Field објеката који садржи јавна поља дате класе и њених надкласа; getDeclaredField враће низ Field објеката за сва поља декларисана у класи описаној датим Class објектом.
- Method[] getMethods()
- Method[] getDeclaredMethods()
  враће низ Method објеката: getMethods враћа јавне методе и садржи и наслеђене методе; getDeclaredMethods враће све методе класе описане датим Class објектом или интерфејса, али резултат не садржи наслеђене методе.



Класе за рефлексију (3)

java.lang.Class

- Constructor[] getConstructors()
- Constructor[] getDeclaredConstructors() 1.1
  враће низ Constructor објеката који садржи све јавне контрукторе (код метода getConstructors) или све конструкторе (код метода getDeclaredConstructors) класе која је представљена датим Class објектом**.**
- T newInstance()
  враће нови примерак класе који је креиран подразумеваним конструктором.
- T cast(Object obj)
  враће obj ако је null или ако може бити конвертован у тип T, иначе избацује изузетак BadCastException.
- T[] getEnumConstants()
  враће низ који садржи све енумерисане вредности уколико је T енумерисаног типа, иначе враће null.
- Class<? super T> getSuperclass()
  враће надкласу дате класе, или null ако T није класа или је T класа Object.



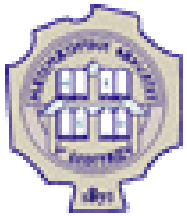
Класе за рефлексију (4)

java.lang.Class

- Constructor<T> getConstructor(Class... parameterTypes)
- Constructor<T> getDeclaredConstructor(Class... parameterTypes)
  одређује јавни конструктор или конструктор са датим типовима параметара.
- Field getField(String name)
- Field[] getFields()
  враће јавно поље са датим именом или низ свих поља.
- Field getDeclaredField(String name)
- Field[] getDeclaredFields()
  враће поље које је декларисано у датој класи и које има име name, или низ свих поља, при чему је поље описано примерком класе java.lang.reflect.Field.

Математички факултет

vladaf@matf.bg.ac.rs



Испитивање хијерархије

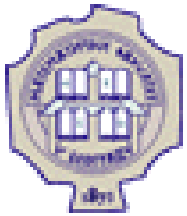
```
public static void traverse(Object o){
  for (int n = 0; ; o = o.getClass())
   {
      System.out.println("L"+ ++n + ": " + o +  ".getClass() = " + o.getClass());
      if (o == o.getClass())
          break;
  }
}
```

```
public static void main(String[] args){
   traverse(new Integer(3));
}
```

L1: 3.getClass() = class java.lang.Integer
L2: class java.lang.Integer.getClass() = class java.lang.Class
L3: class java.lang.Class.getClass() = class java.lang.Class

Математички факултет

vladaf@matf.bg.ac.rs



Рефлексија и низови

Пример. Илуструје како се креира и манипулише низовима чије димензије нису познате до извршења програма.

```
public static void testArray()
{
   Class cls = String.class;
   int i=10;
   Object arr = Array.newInstance(cls, i);
   Array.set( arr, 5, "this is a test");
   String s = (String) Array.get(arr, 5);
   System.out.println(s);
}
```

Рефлексија и динамичко повезивање

Пример. Приликом извршавања следећег кода:

```
Employee e;
e = new MonthlyEmployee();
Class c = e.getClass();
System.out.println("class of e = " + c.getName());
e = new HourlyEmployee();
c = e.getClass();
System.out.println("class of e = " + c.getName());
```

добија се следећи резултат:

```
class of e = MonthlyEmployee
class of e = HourlyEmployee
```

Математички факултет

vladaf@matf.bg.ac.rs

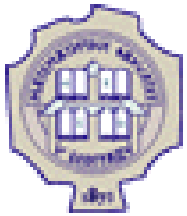
Рефлексија и динамичко повезивање (2)

Пример. Приликом извршавања следећег кода:

```
Employee e;
e = new MonthlyEmployee();
Class c = e.getClass();
c = c.getSuperclass();
System.out.println("base class of e = " + c.getName());
c = c.getSuperclass();
System.out.println("base of base class of e = " + c.getName());
```

добија се следећи излаз:

```
base class of e = Employee
base of base class of e = java.lang.Object
```



Читање вредности за поља

```
e = new MonthlyEmployee();
Field fields[] = c.getFields();
for(int i = 0; i < fields.length; i++) {
        System.out.print(fields[i].getName() + "= ");
        System.out.println(fields[i].getInt(e));
}
```

На излазу се добија:
number= 111
level= 12

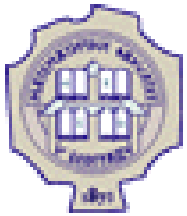
e је примерак класе Employee или њене подкласе

Уочава се да на излазу нису приказана поља која нису јавна.

Метод getDeclaredFields враће сва поља која су декларисана у класи, али искључује поља наслеђена из надкласа.

Математички факултет

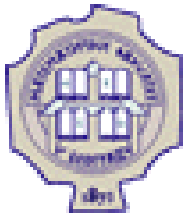
vladaf@matf.bg.ac.rs



Постављање вредности за поља

Пример. Увећање вредности поља level објекта e за 1 се постиже следећом секвенцом наредби:

```
Employee e;
e = new MonthlyEmployee();
Class c = e.getClass();
Field f = c.getField("level");
f.setInt(e,f.getInt(e)+1);
```



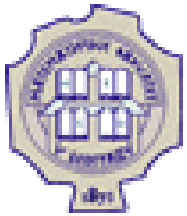
Испитивање модификатора

Пример. Приликом извршавања следећег кода:

```
Employee e;
e = new MonthlyEmployee();
Class c = e.getClass();
 int m = c.getModifiers();
 if (Modifier.isPublic(m))
        System.out.println("public");
 if (Modifier.isAbstract(m))
        System.out.println("abstract");
 if (Modifier.isFinal(m))
        System.out.println("final");
```

добија се следећи излаз:

```
public final
```



Позив метода примерка

Може се позвати метод објекта тј. примерка дате класе, у ком случају се при позиву морају проследити и имплицитни и експлицитни аргументи.

Пример. Позив метода print објекта e се постиже следећом секвенцом наредби:

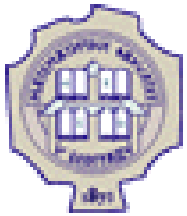
```
Employee e = new HourlyEmployee();
Class c = e.getClass();
Method m = c.getMethod("print", null);
m.invoke(e, null);
```

Као резултат, на излазу се добија:

```
I'm a Hourly Employee
```

Математички факултет

vladaf@matf.bg.ac.rs



Динамичко креирање објекта

За позивање конструктора са аргументима, потребно је користити класу Constructor:

```
Constructor c = ...
Object newObject = c.newInstance( Object[] initArguments )
```

Пример. Нека је класа UniversalPrinter дефинисана на следећи начин:

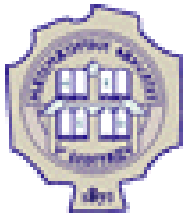
```
class UniversalPrinter {
  public void print(String empType) {
    Class c = Class.forName(empType);
    Employee emp = (Employee ) c.newInstance();
    emp.print();
  }
}
```

Шта је резултат извршавања следећег кода?

```
UniversalPrinter p = new UniversalPrinter();
String empType;
empType = "HourlyEmployee";
p.print(empType);
empType = "MonthlyEmployee";
p.print(empType);
```

Математички факултет

vladaf@matf.bg.ac.rs



Имплементација Јава рефлексије

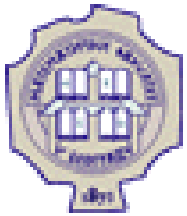
Током извршавања Јава програма, JVM учитава бајт-код тј. class датотеке и креира објекте који представљају те класе.

Објекат који представља класу садржи име (поље типа String), листу поља (свако је типа Field), листу метода…

```
class Field {
   String name;
   Class  type;
   Class  clazz;
   int offset;

   Object get(Object obj) {
        if (clazz.isInstance(obj)) {
              f = ((char*)obj) + offset;
              return (type.primitive = TRUE ? wrap(f)  :  (Object)f);
        }
    }
}
```

```
class Class {
              String   name;
              Field[]  fields;
              Method[] methods;
              boolean  primitive;

              bool isInstance...
              Object newInstance..
}
```

Математички факултет
vladaf@matf.bg.ac.rs



Шта рефлексија не подржава

- Рефлексија је искључиво самоиспитивање
  - Није могуће додати/модификовати поља (тј. мењати структуру класе)
  - Није могуће додати/модификовати методе (тј. мењати понашање)
- Преко рефлексије није доступна имплементација
  - Рефлексијом се не одсликава програмерска логика
- Велики утицај на перформансе
  - Код је много спорији него у случају када се иста операција реализује директним путем…
- Резултујући код је веома комплексан

Математички факултет

vladaf@matf.bg.ac.rs

Захвалница

Велики део материјала који је укључен у ову презентацију је преузет из презентације коју је раније (у време када је он држао курс Објектно орјентисано програмирање) направио проф. др Душан Тошић.

Хвала проф. Тошићу што се сагласио са укључивањем тог материјала у садашњу презентацију, као и на помоћи коју ми је пружио током конципцирања и реализације курса.

Надаље, један део материјала је преузет од колегинице Марије Милановић.

Хвала Марији Милановић на помоћи у реализацији ове презентације.