Објектно орјентисано програмирање

Владимир Филиповић

vladaf@matf.bg.ac.rs

Рефлексија и анотација

Владимир Филиповић
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 интензивно користи код Јава архитектуре за компоненте која се назива JavaBeans. Када се дода нова класа током дизајна или извршавања, RAD алати могу динамички да испитају које су могућности новододате класе.

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.

Примери референтних типова су java.lang.String, све класе-омотачи за примитивне типове као што су java.lang.Double, интерфејс java.io.Serializable и енумератор javax.swing.SortOrder.

Испитивање типа

За сваки претходно побројани елемент тј. тип, Јава виртуална машина формира немутирајући примерак класе java.lang.Class, који 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.

Може се рећи да ова класа представља улазну тачку за цео Reflection API. Осим класе java.lang.reflect.ReflectPermission, ниједна од класа из пакета java.lang.reflect нема јавне конструкторе. Дакле, да би се приступило класама у том палету, неопходно је да се позову одговорајући методи над објектима класе Class.

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());
  }
}
```

Математички факултет
vladaf@matf.bg.ac.rs

Испитивање типа (5)

Приликом извршавања претходног примера добијају се следећи резултати:

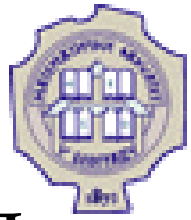
```
class java.lang.Object

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
```

Математички факултет

vladaf@matf.bg.ac.rs



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:

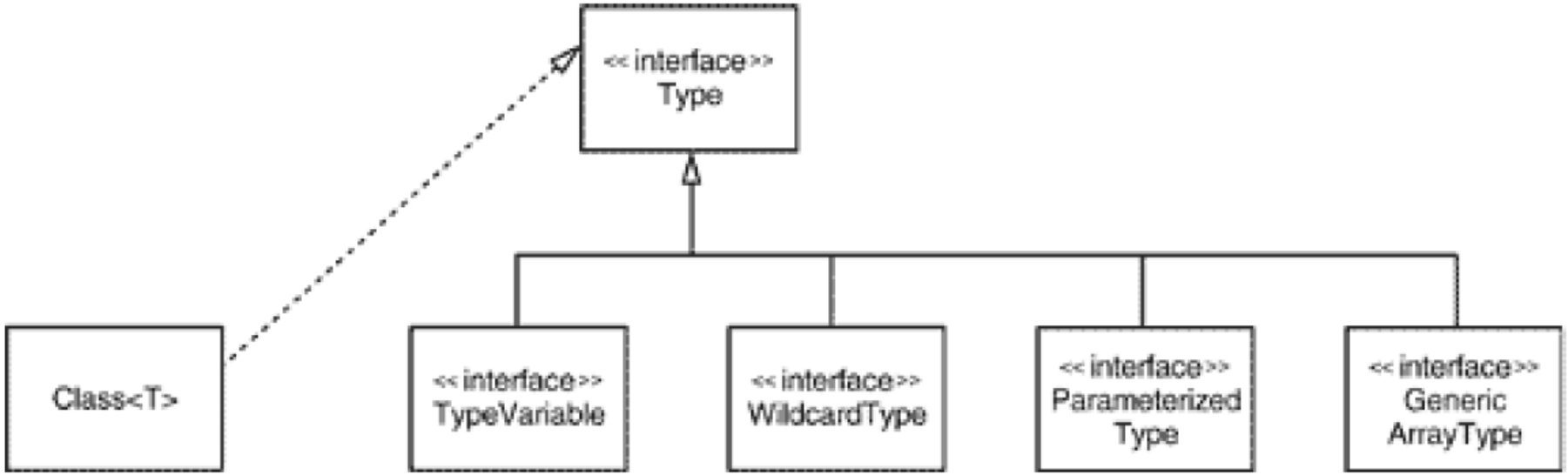
```
Name: equals
  Return Type: boolean
  Parameter Types: java.lang.Object
Name: getClass
  Return Type: java.lang.Class
  Parameter Types:
Name: intersects
  Return Type: boolean
  Parameter Types: double double double double
```

Математички факултет

vladaf@matf.bg.ac.rs

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 кориног 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.

Математички факултет

vladaf@matf.bg.ac.rs

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.

Класе за рефлексију (5)

java.lang.reflect
Пакет који садржи класе и интерфејсе за подршку рефлексији.

java.lang.reflect. Member
Интерфејс који одсликава идентификујуће информације о члану класе (пољу, методу или конструктору).

java.lang.reflect.Method
Имплементира интерфејс Member. Ова класа обезбеђује информације о методу, као и приступ том методу (било да се ради о методу класе или методу интерфејса, било да се ради о статичком методу или методу примерка).

- public Object invoke(Object implicitParameter, Object[] explicitParameters)
  позива метод који је описан са датим Method објектом, тако што му проследи дате параметре и као резултат врати резултат извршења метода.
  Ако се позива статички метод, онда се као имплицитни параметар прослеђује вредност null.
  Ако су параметри примитивног типа, тада се прослеђују вредности објеката омотача. Ако метод враће вредност примитивног типа, тада је потребно да се та вредност „размота “, тј. да се издвојити из објекта-омотача који враће метод invoke.

Математички факултет
vladaf@matf.bg.ac.rs

Класе за рефлексију (6)

java.lang.reflect.Method

- Class getDeclaringClass()
  враће примерак класе Class за класу у којој је дефинисан овај метод.
- Class[] getExceptionTypes()
  враће низ објеката типа Class, који представљају типове изузетака које може избацити овај метод.
- int getModifiers()
  враће цео број који описује модификаторе овог метода. За анализу враћене вредности користе се методе дефинисане у класи Modifier.
- String getName()
  враће ниску која представља име метода.
- Class[] getParameterTypes()
  враће низ Class објеката који представља типове параметара за метод.
- Class getReturnType()
  враће примерак класе Class који представља тип који враће дати метод.

Математички факултет
vladaf@matf.bg.ac.rs

Класе за рефлексију (7)

java.lang.reflect.Field

Класа Field имплеметира интерфејс Member. Она обезбеђује информације о једном пољу – било статичком пољу, било пољу примерка. Она такође обезбеђује и динамички приступ пољу, тј. очитавање и модификацију његовог садржаја.

- Object get(Object obj)
  враће вредност поља које је описано Field објектом obj.
- void set(Object obj, Object newValue)
  поставља вредност поља описаног Field објектом obj на вредност newValue.
- Class getDeclaringClass()
  враће примерак класе Class за класу која садржи ово поље.
- int getModifiers()
  враће цео број који описује модификаторе овог поља.
- String getName()
  враће ниску која представља име поља.

java.lang.reflect.Package

Објекти примерци ове класе садрже информације о имплементацији и спецификацији Јава пакета.

Математички факултет
vladaf@matf.bg.ac.rs

Класе за рефлексију (8)

java.lang.reflect.Constructor

Имплементира интерфејс Member. Ова класа обезбеђује информације и приступ конкретном конструктору класе.

- Object newInstance(Object[] args)
  креира нови примерак класе.
- T newInstance(Object... parameters)
  креира нови примерак класе конструисан са датим параметрима.
- Class getDeclaringClass()
  враће примерак класе Class за класу у којој је дефинисан дати конструктор.
- Class[] getExceptionTypes()
  враће низ објеката типа Class, који представљају типове изузетака које може избацити овај конструктор.
- int getModifiers()
  враће цео број који описује модификаторе овог конструктора. За анализу враћене вредности користе се методе дефинисане у класи Modifier.
- String getName()
  враће ниску која представља име конструктора.
- Class[] getParameterTypes()
  враће низ Class објеката који представља типове параметара за конструктор.

Математички факултет
vladaf@matf.bg.ac.rs

Класе за рефлексију (9)

java.lang.reflect.Modifier

Ова класа обезбеђује статичке методе и константе за декодирање модификатора. Скуп модификатора је енкодиран као цео број где различити битови указују да ли је у скуп укључен дати модификатор, или не.

- static String toString(int modifiers)
  враће ниску са модификаторима који одговарају скупу представљеном помоћу битова.
- static boolean isAbstract(int modifiers)
- static boolean isFinal(int modifiers)
- static boolean isInterface(int modifiers)
- static boolean isNative(int modifiers)
- static boolean isPrivate(int modifiers)
- static boolean isProtected(int modifiers)
- static boolean isPublic(int modifiers)
- static boolean isStatic(int modifiers)
- static boolean isStrict(int modifiers)
- static boolean isSynchronized(int modifiers)
- static boolean isVolatile(int modifiers)
  тестира одговарајући бит од modifiers који одговара датом модификатору.

Математички факултет
vladaf@matf.bg.ac.rs

Класе за рефлексију (10)

java.lang.reflect.AccessibleObject

Надкласа класе Field. Помоћу ње се дефинише могућност приступа објекту.

- void setAccessible(boolean flag)
  поставља маркер доступности за објекат на који се примењује рефлексија. Вредност аргумента true указује да је искључена провера приступа од стране језика Јава и да се могу испитивати и подешавати чак и приватне особине објекта.
- boolean isAccessible()
  враће маркер доступности за објекат на који се примењује рефлексија.
- static void setAccessible(AccessibleObject[] array, boolean flag)
  погодан метод за постављање маркера доступности flag за читав низ објеката.

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

java.lang.reflect.Array

- static Object get(Object array, int index)
- static xxx getXxx(Object array, int index)
  (xxx је један од примитивних типова boolean, byte, char, double, float, int, long, short.) Ови методи враћу вредност датог низа која се налази на датој позицији index.
- static void set(Object array, int index, Object newValue)
- static setXxx(Object array, int index, xxx newValue)
  (xxx је један од примитивних типова boolean, byte, char, double, float, int, long, short.) Ови методи смештају нову вредност newValue у дати низ на дату позицију index.
- static int getLength(Object array)
  враће дужину датог низа.
- static Object newInstance(Class componentType, int length)
- static Object newInstance(Class componentType, int[] lengths)
  враће нови низ чије су компоненте датог типа componentType, а чија је димензија одређена другим аргументом.

Математички факултет

vladaf@matf.bg.ac.rs

Рефлексија и низови (2)

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

```
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
```

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..
}
```

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

Анотације (забелешке)

Анотације (још се називају и забелешке) обезбеђују податке о програму које саме по себи нису део програма.

- Анотације су специјална врста коментара
  - Исто као коментари, анотације се мењају нити утичу на семантику програма, тј. на понашање програма током његовог извршавања.
- Анотације су мета-описи
  - За разлику од коментара, анотацијама могу приступити и њих могу користити други развијени програмски алати или чак и сам програм који развијамо.

Анотације (2)

Анотације се користе у различите сврхе:

- Обезбеђивање додатних информација преводиоцу – анотације се могу користити од стране преводиоца ради детектовања грешака или ускраћивања упозорења.
- Процесирање у времену превођења и у времену испоручивања – софтверски алати могу процесирати анотације ради генерисања кода, XML датотека, итд.
- Процесирање у времену извршавања — неке анотације су доступне за испитивање у времену извршавања.

Анотације се могу применити на декларације: декларације класа, поља, метода и других елемената програма.

Када се примењују на декларације, конвенција је да се свака анотација пише у новој линији.

Формат анотације

У свом најпростијем облику, анотација има следећи формат:

```
@Entity
```

Знак @ указује преводиоцу да се ради о анотацији.

У примеру који следи, име анотације је Override:

```
@Override
void mySuperMethod() { ... }
```

Анотација може да садржи елементе, било именоване или неименоване, и да поставе вредности за те елементе:

```
@Author(
   name = "Benjamin Franklin",
   date = "3/27/2003"
)
class MyClass() { ... }
```

или

```
@SuppressWarnings(value = "unchecked")
void myMethod() { ... }
```

Формат анотације (2)

Ако постоји само један именовани елемент, тада се име елемента може уклонити приликом доделе вредности, као у:

```
@SuppressWarnings("unchecked")
void myMethod() { ... }
```

Ако анотација нема елемената (маркерска), тада се и заграде могу уклонити, што је и био случај у примеру са @Override анотацијом.

Такође је могуће поставити више анотација на исту декларацију:

```
@Author(name = "Jane Doe")
@EBook
class MyClass { ... }
```

Тип анотације може бити новонаправљени (енг. custom) тип, или један од типова из пакета java.lang или java.lang.annotation у оквиру Java SE API. У претходним примерима, анотације Override и SuppressWarnings су предефинисани типови Јава анотације, а Author и Ebook су новонаправљени типови анотације.

Предефинисани тип анотације

Предефинисани типови анотације већ постоје и користе се у језику Јава.

Ови типови анотације су дефинисани у пакету java.lang.

То су:

- Deprecated
- Override
- SuppressWarnings
- SafeVarargs

Поред тога, у Java SE 8 ће се појавити још један тип анотације, назван FunctionalInterface.

Предефинисани тип анотације (2)

Анотација Deprecated указује да означени елеменат више није потребан (застарео је) и да се надаље неће користити.
Кад год програм користи метод, класу или поље које је анотирано Deprecated анотацијом, Јава преводилац генерише упозорење.
Када елеменат постане застарео, то такође треба и документовати коришћењем the Javadoc тага @deprecated.
И таг и анотација почињу симболом @, иза кога код Javadoc тага следи мало слово d а код анотације велико слово D.

Пример.

```
// Javadoc komentar
 /**
  * @deprecated
  * objasnjenje zasto je metod nepotreban i kako se preporucuje da se radi
  */
 @Deprecated
 static void deprecatedMethod() { }
```

Предефинисани тип анотације (3)

Анотација Override информише преводилац да анотирани елемент треба да превазиђе елеменат који је декларисан у надкласи.

Пример.

```
// mark method as a superclass method
// that has been overridden
@Override
int overriddenMethod() { }
```

Иако се у Јава програмирању не захтева да се приликом превазилажења метода користи ова анотација, њено коришење помаже у превенцији грешака.

Ако метод маркиран са анотацијом Override некоректно превазиђе метод надкласе, тада преводилац генерише грешку.

Предефинисани тип анотације (4)

Анотација @SuppressWarnings налаже преводиоцу да не приказује конкретни тип упозорења, које би у супротном било приказано.

Пример. У коду који следи се користи застарели метод deprecatedMethod, за који преводилац обично генерише упозорење. У овом случају, међутим, анотација метода useDeprecatedMethod је блокирала генерисање једног типа упозорења.

```
// koristi zastareli metod i nalaze prevodiocu da ne generise upozorenje
@SuppressWarnings("deprecation")
void useDeprecatedMethod() {
   objectOne.deprecatedMethod();
}
```

Спецификација језика Јава разликује две категорије упозорења преводиоца: застарелост (енг. deprication) и непровереност (енг. unchecked). Непроверена упозорења се могу појавити кад се ради са старим кодом, кодом који је писан пре развоја генеричких класа и метода.

Предефинисани тип анотације (5)

Да би се блокирало генерисање обе категорије упозорења, користи се следећа синтакса:

```
@SuppressWarnings({"unchecked", "deprecation"})
```

Анотација @SafeVarargs , када се примени на метод или на конструктор, обезбеђује да се у телу метода/контруктора не извршавају операције које могу бити несигурне за параметре променљивог типа овог метода/конструктора, тј. за varargs параметре. Ако се користи овај тип анотације, тада се код преводиоца блокира и генерисање упозорења непроверивости који би се односили на коришћење varargs параметара.

Анотација @FunctionalInterface , која ће бити уведена у Java SE 8, указује да декларација типа треба да буде функционални интерфејс, као што је дефинисано у спецификацији језика Јава.

Креирање новог типа анотације

Креирање новог типа анотације је слично креирању интерфејса, при чену декларацији типа анотације претходи знак @.

Анотација не сме садржавати кључну реч extends. Међутим, анотације имплицитно наслеђују интерфејс Annotation.

Тело анотације се састоји од декларације метода (без тела метода). Матоди унутар тела анотације се понашају као поља.

Пример. Тип анотације Description, којом се нпр. описује програмска датотека, може да има следећи облик:

```
@Retention( RetentionPolicy.RUNTIME )
@interface Description
{
   String author();
   String date();
}
```

Креирање новог типа анотације (2)

Када анотација датог типа придружује декларацији, потребно је обезбедити одговарајуће вредности за чланове типа анотације.

Пример (наставак). Када је креиран тип анотације Description, тада се њиме могу анотирати класе и методе (нпр. Test и testMethod):

```
@Description( author = "Vlado", date = "22/10/2011,23/10/2011" )
public class Test {
   @Description( author = "Vlado", date = "22/10/2011" )
   public static void testMethod(){
      System.out.println( "Welcome to Java" );
      System.out.println( "This is an example of Annotations" );
   }
   public static void main( String args[] ){
      testMethod();
      showAnnotations();
   }
}
```

Креирање новог типа анотације (3)

У примеру није приказан метод showAnnotations, који коришћењем рефлексије приказује на стандардном улазу анотације које су придружене класи и методама.

Метод showAnnotations ће бити приказан и детаљно анализиран нешто касније, у делу презентације који се односи на коришћење рефлексије ради испитивања анотација.

Мета-анотације (1)

Пример. У претходном примеру се, приликом креирања новог типа анотације Description, користила анотација Retention. Ова анотација Retention је предефинисана (већ постоји), служи за анотирање анотација, па се због тога назива **мета-анотација**.
Мета-анотације су дефинисане у пакету java.lang.annotation.
То су: @Retention, @Documented, @Target и @Inherited.
Поред ових, у у Java SE 8 ће се појавити још један тип мета-анотације, назван @Repeatable.
Мета-анотација @Retention одређује како се памти анотација:
RetentionPolicy.SOURCE – анотација се памти само на нивоу изворног кода и бива игнорисана од стране преводиоца.
RetentionPolicy.CLASS – анотација се памти од стране преводиоца током превођења, али је игносрише JVM.
RetentionPolicy.RUNTIME – анотација се памтио од стране JVM, па се може користити у окружењу извршавања.

Мета-анотације (2)

Мета-анотација @Documented указује да се анотација која је маркирана са Documented треба документовати коришћењем Javadoc алата (подразумевано је да се анотације не укључују у Javadoc). За више информација консултовати документацију о Javadoc алатима.

Мета-анотација @Target означава ограничење типа Јава елемента на који се може примењивати тако маркирана анотација. Могући циљ маркиране анотације, тј. вредност мета-анотације је нека од следећа четири вредности:

ElementType.ANNOTATION_TYPE примењује се на тип анотације.

ElementType.CONSTRUCTOR примењује се на конструктор .

ElementType.FIELD примењује се на поље или на особину.

ElementType.LOCAL_VARIABLE примењује се на локалну променљиву.

ElementType.METHOD примењује се на метод.

Мета-анотације (3)

ElementType.PACKAGE примењује се на декларацију пакета.

ElementType.PARAMETER примењујњ се на параметар метода.

ElementType.TYPE примењује се на ма који елеменат класе.

Мета-анотација @Inherited указује да тип анотације може бити наслеђен из надкласе, што није подразумевано понашање. Када корисник испитује тип анотације, а класа нема анотацију датог типа, тада ће се надкласа испитивати за дати тип анотације. Ова мета-анотација се односи само на анотације класе.

Мета-анотација @Repeatable указује да означена анотација може бити примењена више пута на исту декларацију.

Испитивање анотација

Reflection API садржи методе за испитивање анотација.

Старији начин за испитивање анотација су методи getAnnotation(Class<T>) и getAnnotation(Class<T>), који су дефинисани у класама Class и Method респективно.
Ови методи враћају једну анотацију, исто као и новији метод getAnnotationByType(Class<T>) класе AnnotatedElement, под претпоставком да постоји анотација захтеваног типа.

У Java SE 8 су укључени и додтни методи који пролазе кроз контејнер анотација и једним позивом врћз све анотације датог типа. Такав је метод getAnnotations(Class<T>) класе AnnotatedElement.

Испитивање анотација (2)

Пример. Метод који следи приказује анотације:

```
public static void showAnnotations()
{
   Test test = new Test();
   try{
      Class c = test.getClass();
      Description annotation1 = (Description) c.getAnnotation( Description.class );
      System.out.println( "Name of the class: " + c.getName() );
      System.out.println( "Author of the class: " + annotation1.author() );
      System.out.println( "Date of Writing the class: " + annotation1.date() );
      Method m = c.getMethod( "testMethod" );
      Description annotation2 = m.getAnnotation( Description.class );
      System.out.println( "Name of the method: " + m.getName() );
      System.out.println( "Author of the method: " + annotation2.author() );
      System.out.println( "Date of Writing the method: " + annotation2.date() );
   } catch (NoSuchMethodException ex){
       System.out.println( "Invalid Method..." + ex.getMessage() );
   }
}
```

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.