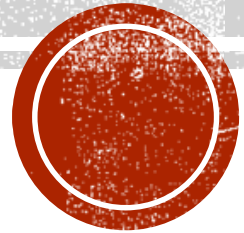
ОБЈЕКТНО ОРЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК ЈАВА — 2



Рефлексија

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.

РЕФЛЕКСИВНИ ПРОГРАМ

- 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 енумерисани тип `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 пакету, неопходно је да се позову одговорајући методи над објектима класе `Class`.

ПРИСТУП ОБЈЕКТУ КОЈИ ЧУВА ИНФОРМАЦИЈЕ О КЛАСИ

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;
```

Трећа могућност је позиво метода `forName` коме је прослеђено име класе:

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

Референца на објекат типа `Class` која представља надкласу датог `Class` објекта добија се са:

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

Одређивање имена класе добија се са:

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

Одређивање интерфејса који имплементира дата класа добија се са:

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

Одређивање поља дате класе добија се са:

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

Одређивање метода дате класе добија се са:

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

ДОБИЈАЊЕ РЕФЕРЕНЦЕ НА ОПИС КЛАСЕ

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;");
```

ДОБИЈАЊЕ ИНФОРМАЦИЈА О НАДКЛАСИ И ИНТЕРФЕЈСИМА

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());
  }
}
```

ДОБИЈАЊЕ ИНФОРМАЦИЈА О НАДКЛАСИ И ИНТЕРФЕЈСИМА (2)

Пример (наставак). 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
```

ДОБИЈАЊЕ ИНФОРМАЦИЈА О МЕТОДИМА

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();
   }
}
```

ДОБИЈАЊЕ ИНФОРМАЦИЈА О МЕТОДИМА (2)

Пример (наставак). Позивом претходно дефинисаног метода, тј извршењем кода:

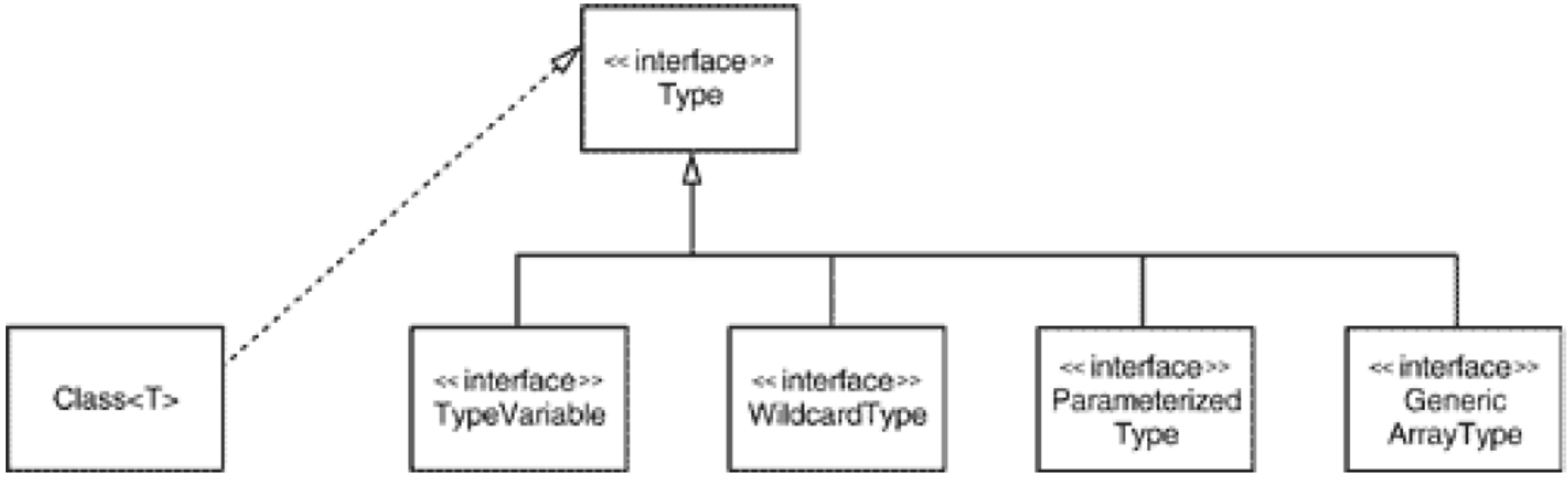
```
Polygon p = new Polygon();
showMethods(p);
```

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.



КЛАСА `java.lang.Class`

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

- `static Class forName(String className)`
  враћа `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.

КЛАСА `java.lang.Class` (2)

- `Constructor[] getConstructors()`
- `Constructor[] getDeclaredConstructors()`
  враћа 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()`
  враћа нови примерак класе који је креиран подразумеваним конструктором.
- `T cast(Object obj)`
  враћа `obj` ако је `null` или ако може бити конвертован у тип `T`, иначе избацује изузетак `BadCastException`.
- `T[] getEnumConstants()`
  враћа низ који садржи све енумерисане вредности уколико је `T` енумерисаног типа, иначе враћа `null`.
- `Class<? super T> getSuperclass()`
  враћа надкласу дате класе, или `null` ако `T` није класа или је `T` класа `Object`.

КЛАСА `java.lang.Class` (3)

- `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**.

ПРИБАВЉАЊЕ ИНФОРМАЦИЈА - ПРИМЕРИ

Пример 1. Написати програм који приказује информацију о класи за инстанце објектног типа - класе, интерфејсе и низове.

Решење је дато у репозиторијуму кода, доступном на вебу.

Пример 2. Написати програм који за дату класу/интерфејс, чији је назив дат преко командне линије, приказује информације о њој тако што реконструише заглавље те класе/интерфејса.

Решење је дато у репозиторијуму кода, доступном на вебу.

Пример 3. Написати програм који за дату класу, чији је назив дат преко командне линије, приказује информације о интерфејсима које та класа имплементира, њеним пољима и методама, као и информације о интерфејсима које имплементира њена надкласа, те о пољима и методама декларисаним у надкласи.

Решење је дато у репозиторијуму кода, доступном на вебу.

ИНТЕРФЕЈС `java.lang.reflect.Member`

`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).

КЛАСА `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()`
  враћа примерак класе `Class` за класу која садржи ово поље.
- `int getModifiers()`
  враћа цео број који описује модификаторе овог поља.
- `String getName()`
  враће ниску која представља име поља.

КЛАСА `java.lang.reflect.Method`

Класа `Method` имплеметира 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).

- `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.

КЛАСА `java.lang.reflect.Method` (2)

- `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**.

КЛАСА `java.lang.reflect.Constructor`

Класа `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.

КЛАСА `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.

КЛАСА `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 `flag` `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 ХИЈЕРАХИЈЕ

Пример. Илуструје хијерахију добијања информација о класи:

```
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
```

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
```

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

Пример. Илуструје хијерахију добијања информација о класи:

```
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.

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 ОБЈЕКАТА

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);
```

РЕФЛЕКСИЈА И НИЗОВИ

Класа `java.lang.reflect.Array` садржи следеће методе:

- `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.

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);
}
```

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

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

```
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);
        }
    }
}
```

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

Пример 12. Написати програм који упоређује време приступа члану низа директним путем, са временом приступа том истом чалну коришћењем рефлексије.

Решење је дато у репозиторијуму кода, доступном на вебу.