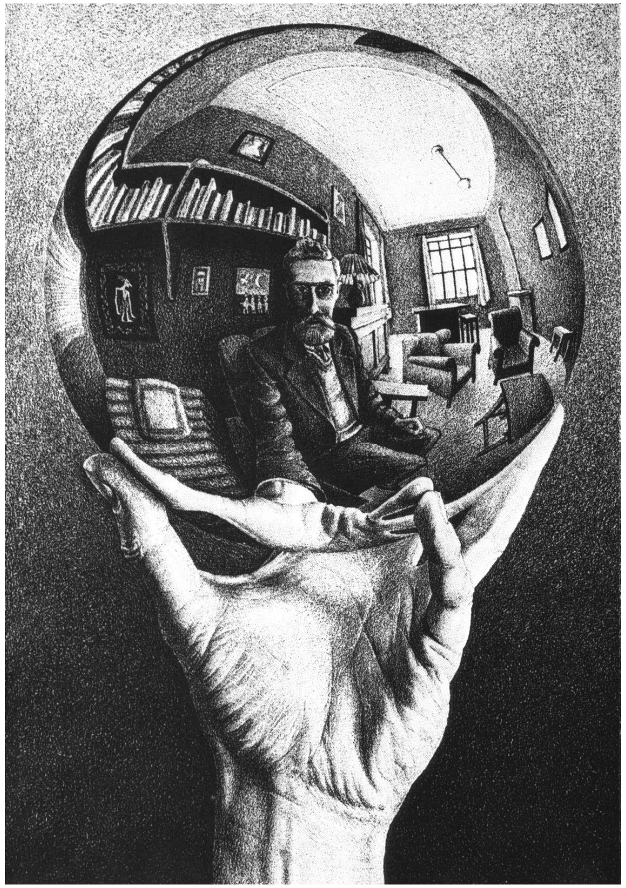
Reflexivita

(Java Reflection Model)



Å moǵnosŠļ ítaŞ vykonávaŞ resp. modifikovaŠ
program, ktorý sa práve vykonáva

Å je to vlastnosŞ ktorá sa vyskytuje v interpreteroch
(v interpretovaných jazykoch), nie v kompilátoroch
(v skutoļ ných jazykoch ako C, C++)

Preļ o ??

Å JAVA je niekde medzi, lebo sa kompiluje do byte kódu, ktorý je ale
interpretovaný

JAVA poskytuje

Å Introspection: triedy Class a Field pre ļ ítanie vlastného programu

Å Reflexívne volanie: triedy Method, Constructor

Nadtrieda a Podtrieda

(ilustraļ ný príklad)

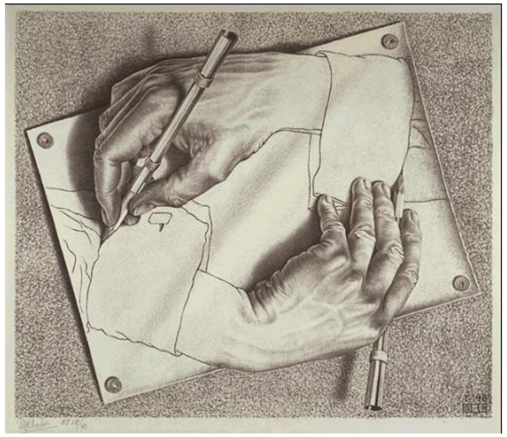
```
public class Nadtrieda implements Runnable {
           public int variabla;
           public int[] pole = {1,2,3};
           public String[] poleStr = {"janko", "marienka" };
           public Nadtrieda()  {   }
           public Nadtrieda(int a) {   }

           public void Too(double r) {   }
           public void run() {  … kvôli Runnable ... }
}
```

```
public class Podtrieda extends Nadtrieda {
      public Podtrieda(String s) { }

      public class Vnorena { }
      public interface Prazdny {}
}
```

Trieda Class<T>



Russellov paradox (antinómia)

$$S = \{X | X \notin X\}$$

Kaģdý objekt pozná metódu getClass():

```
        Class nt = new Nadtrieda().getClass();
```

Class:
Åhodnotou sú reflexívne obrazy tried náġho programu,
Åumoģní nám ļ ítaŠa spúġŠaŠļ asti náġho programu,
Åo.i. pozná metódu String getName()

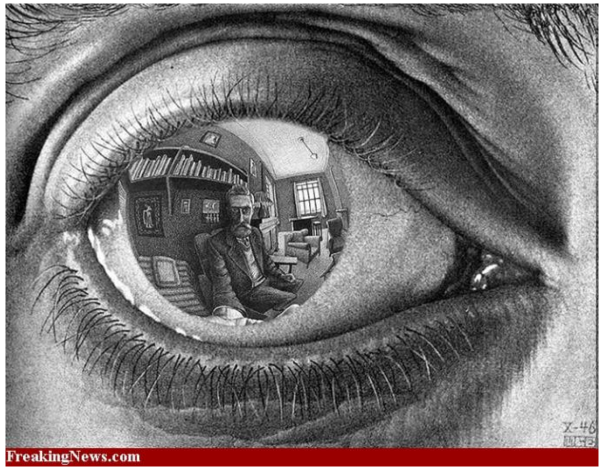
```
        System.out.println(nt.getName());        // Nadtrieda

Class nt1 = Nadtrieda.class;                      // Trieda.class
        System.out.println(nt1.getName());               // Nadtrieda
```

Åmeta-trieda:

```
        Class klas = Class.class;
```

Trieda Class<T>



```
try {
            Class pt = Class.forName("Podtrieda");          // forName(ňé  ò)
            System.out.println(pt.getName());               // Podtrieda
            Class nt2 = pt.getSuperclass();                 // getSuperClass()
            System.out.println(nt2.getName());                          // Nadtrieda
            for(Class cl:pt.getClasses())                   // getClasses()
                                                            // public classes &
interf
            System.out.print(cl.getName());                 // Podtrieda$Prazdny
                                                            // Podtrieda$Vnorena

} catch (ClassNotFoundException e) {
            e.printStackTrace();
}
```

Metódy Class<T>

Å T cast(Object obj) pretypuje obj do triedy T

Å static Class<?> forName(String name)
vráti Class objekt zodpovedajúci triede s menom name

Å Class[] getClasses() public triedy a interface implementované touto triedou

Å Constructor[] getConstructors()
vġetky konġtruktory triedy

Å Constructor<T> getConstructor(Class... parameterTypes)
konġtruktor triedy pre parameterTypes

Å Field[] getFields() vġetky pologky (premenné) triedy

Å Field getField(String name) pologka s menom name

Å Method[] getMethods() vġetky metódy triedy

Å Method getMethod(String name, Class... parameterTypes)

Å int getModifiers() atribúty triedy (public, abstract, é)

Å String getName() meno triedy

Å boolean isInstance(Object obj) je inġtanciou triedy ?

Å boolean isArray() je pole ?

Å boolean isPrimitive() je primitívny typ ? (int, double, booleané)

Class<T>

Trieda Class umoǵŔuje prístup k atribútom triedy

```
int m = nt.getModifiers();
if (Modifier.isPublic(m))
        System.out.println("public");
```

podobne:
isPrivate(), isProtected(), isStatic, isFinal(), isAbstract(), isFinal(), isSynchronized(),

Trieda Class umoǵŔuje prístup k interface triedy

```
Class[] theInterfaces = nt.getInterfaces();
for (int i = 0; i < theInterfaces.length; i++) {
        String interfaceName = theInterfaces[i].getName();
        System.out.println(interfaceName);
}
```

java.lang.Runnable

Premenné, konštruktory

```
Field[] publicFields = nt.getFields();
for (int i = 0; i < publicFields.length; i++) {
        String fieldName = publicFields[i].getName();
        Class typeClass = publicFields[i].getType();
        String fieldType = typeClass.getName();
        System.out.println("Name: " + fieldName + ", Type: " + fieldType);
 }
```

```
Name: variabla,  Type: int
Name: pole,      Type: [I
Name: poleStr,
Type: [Ljava.lang.String;
```

```
Class intArray = Class.forName("[I");
Class stringArray =
Class.forName("[Ljava.lang.String;");
```

```
Constructor[] theConstructors = nt.getConstructors();
 for (int i = 0; i < theConstructors.length; i++) {
   System.out.print("( ");
   Class[] parameterTypes =
theConstructors[i].getParameterTypes();
   for (int k = 0; k < parameterTypes.length; k ++) {
        String parameterString = parameterTypes[k].getName();
        System.out.print(parameterString + " ");
    }
   System.out.println(")");
 }
```

```
( )
( int )
```

Metódy

```
Method[] theMethods = nt.getMethods();
 for (int i = 0; i < theMethods.length; i++) {
        String methodString = theMethods[i].getName();
        System.out.println("Name: " + methodString);

        String returnString =  theMethods[i].getReturnType().getName();
        System.out.println("   Return Type: " + returnString);

        Class[] parameterTypes = theMethods[i].getParameterTypes();
        System.out.print("   Parameter Types:");
        for (int k = 0; k < parameterTypes.length; k ++) {
                String parameterString = parameterTypes[k].getName();
                System.out.print(" " + parameterString);
        }
        System.out.println();
 }
```

```
Name: Too
   Return Type: void
   Parameter Types: double
Name: run
   Return Type: void
   Parameter Types:
... Metódy Object-u
```

Je inġtanciou

cl.isInstance(obj) je true, ak obj je inġtanciou triedy reprezentovanie v cl.

Class nt = new Nadtrieda().getClass();

nt.isInstance(new Nadtrieda()) == true

class1.isAssignableFrom(class2) je true ak trieda reprezentovaná class1 je nadtriedou/nadinterface triedy reprezentovanej class2, teda do premennej typu reprezentovaneho class1 môǵeme priradiŠobjekt typu reprezentovaného class2.

Ergo:

cl.isAssignableFrom(obj.getClass()) == cl.isInstance(obj)

Prístup k premennej

```
if (Integer.class.isAssignableFrom(Integer.class)) {   // true
    Nadtrieda o = new Nadtrieda();
    Field f = o.getClass().getField("boxedInt");
    f.setAccessible(true);
    f.set(o, new Integer(88));                         // o.boxedInt = 88;
    System.out.println(f.get(o));                      // o.boxedInt;
}
if (int.class.isAssignableFrom(int.class)) {   // true
    Nadtrieda o = new Nadtrieda();
    Field f = o.getClass().getField("variabla");
    f.setAccessible(true);
    f.set(o, new Integer(66));                         // o.variabla = 66;
    alebo
    f.setInt(o, 77);                                   // o.variabla = 77;
    System.out.println(f.get(o));                      // o.variabla;
    System.out.println(f.getInt(o));                   // o.variabla;
}
```

```
public class Nadtrieda implements Runnable {
    public int variabla;
    public Integer boxedInt;
    public Nadtrieda()  {           }
    public Nadtrieda(int a) {   }
    public void Too(double r) {   }
    public void run() {  … kvôli Runnable ... }
}
```

Volanie konštruktora

```
try {
    Nadtrieda nt2 = (Nadtrieda)(nt.getConstructor(int.class).newInstance(3));
                                                        // new Nadtrieda(3)
} catch (InstantiationException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
} catch (IllegalArgumentException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
} catch (NoSuchMethodException e) {
    e.printStackTrace();
} catch (SecurityException e) {
    e.printStackTrace();
}
```

```
public class Nadtrieda implements Runnable {
    public int variabla;
    public Integer boxedInt;
    public Nadtrieda()  {          }
    public Nadtrieda(int a)  {   }
    public void Too(double r)  {   }
    public void run()  {  … kvôli Runnable ... }
}
```

Volanie konštruktora

V prípade konġtruktora bez argumentov:

```
Class classDefinition = Class.forName(className);
Object object = classDefinition.newInstance();
```

```
Class rectangleDefinition = Class.forName("java.awt.Rectangle");

// pole typov argumentov konġtruktora, t.j. Class[]
Class[] intArgsClass = new Class[] {int.class, int.class};

// daj mi konġtruktor s daným typom argumentov
Constructor intArgsConstructor =
      rectangleDefinition.getConstructor(intArgsClass);

// pole hodnôt argumentov konġtruktora, t.j. Object[]
Object[] intArgs = new Object[] {new Integer(12), new Integer(34)};
Rectangle  rectangle =
      (Rectangle) createObject(intArgsConstructor, intArgs);
```

Volanie metódy

```
try {

    (o.getClass()).getMethod("run").invoke(o);            // o.run();

    Method met = (o.getClass()).getMethod("Too",new Class[]{double.class});
    met.invoke(o,new Object[]{new Double(Math.PI)});// o.Too(Math.PI);

    (o.getClass()).getMethod("Too",double.class).invoke(o,Math.PI);
                                                        //  o.Too(Math.PI);

} catch (SecurityException | NoSuchFieldException | IllegalAccessException |
    IllegalArgumentException | InvocationTargetException |
    NoSuchMethodException e) {
    e.printStackTrace();
}
```

```
public class Nadtrieda implements Runnable {
    public int variabla;
    public Integer boxedInt;
    public Nadtrieda()  {          }
    public Nadtrieda(int a)  {    }
    public void Too(double r)  {    }
    public void run()  {  … kvôli Runnable ... }
}
```

Volanie metódy

```
public static String append(String firstWord, String secondWord) {
    String result = null;

    try {

        // pole typov argumentov metódy, t.j. Class[]
        Class[] parameterTypes = new Class[] {String.class};
        Class c = String.class;

        // daj mi metódu s daným typom argumentov
        Method concatMethod = c.getMethod("concat",
parameterTypes);

        // pole hodnôt argumentov metódy, t.j. Object[]
        Object[] arguments = new Object[] {secondWord};
        result = (String) concatMethod.invoke(firstWord, arguments);

    } catch (Exception e) {
     . . . .
    }
    return result;
```

Polia

(java.lang.reflect.Array)

```
int[] pole = (int[]) Array.newInstance(int.class, 5);      // int[] pole = new int[5];

 for(int i = 0; i < Array.getLength(pole); i++) {
    Array.set(pole, i, i);                                          // pole[i] = i;
    Array.setInt(pole, i, i);                                       // pole[i] = i;
 }
 for(int i = 0; i < Array.getLength(pole); i++ ) {
    System.out.println("pole["+i+"] = " + Array.get(pole, i));      // pole[i] = i;
    System.out.println("pole["+i+"] = " + Array.getInt(pole, i));   // pole[i] = i;
 }
```

```
pole[0] = 0
pole[1] = 1
pole[2] = 2
pole[3] = 3
pole[4] = 4
```

Polia

(java.lang.reflect.Array)

```
Nadtrieda o = new Nadtrieda();
Field f = o.getClass().getField("pole");
Object oo = f.get(o);
 if (oo.getClass().isArray()) {
   System.out.println(Array.getLength(oo));
   for(int i=0; i<Array.getLength(oo); i++)
        System.out.println(Array.getInt(oo,i));
 }
```

```
3
1
2
3
```

```
 Object ooo = o.getClass().getField("poleStr").get(o);
 if (ooo.getClass().isArray()) {
   System.out.println(Array.getLength(ooo));
   for(int i=0; i<Array.getLength(ooo); i++)
        System.out.println(Array.get(ooo,i));
 }
```

```
2
janko
marienka
```

Efektivita

```
Nadtrieda nt=new Nadtrieda();

start=System.nanoTime();
for(int i=0;i<MAX;i++)
    nt.Too(Math.PI);
end=System.nanoTime();

Method m=nt.getClass().getMethod("Too",double.class);
startReflex=System.nanoTime();
for(int i=0;i<MAX;i++)
    m.invoke(nt, Math.PI);
endReflex=System.nanoTime();
```

```
regular call: 0.05669715
reflexive call:1.47600883
Slowdown factor:26x

regular new: 0.56120261
reflexive new:2.3079218200000002
Slowdown factor:4x
```

JDK8 - funkcionálny interface

```
interface FunkcionalnyInterface { // koncept funkcie v J8
   public void doit(String s);     // jediná "procedúra"
}
                                   // „procedúra" ako argument
public static void foo(FunkcionalnyInterface fi) {
   fi.doit("hello");
}
                                // „procedúra" ako hodnota, výsledok
public static FunkcionalnyInterface goo() {
   return (String s) -> System.out.println(s + s);
}

foo(goo())
"hellohello"
```

Funkcie.java

JDK8 - funkcionálny interface

```
public interface FunkcionalnyInterface { //String->String
   public String doit(String s); // jediná "funkcia"
}
                                // "funkcia" ako argument
public static String foo(FunkcionalnyInterface fi) {
   return fi.doit("hello");
}
                               // "funkcia" ako hodnota
public static FunkcionalnyInterface goo() {
   return
        (String s)->(s+s);
}
System.out.println(foo(goo()));
"hellohello"
```

Funkcie.java

JDK8 - funkcionálny interface

```
public interface RealnaFunkcia {
   public double doit(double s);    // funkcia R->R
}
public static RealnaFunkcia iterate(int n, RealnaFunkcia f){
   if (n == 0)
       return (double d)->d;        // identita
   else {
       RealnaFunkcia rf = iterate(n-1, f);      // f^(n-1)
       return (double d)->f.doit(rf.doit(d));
   }
}
RealnaFunkcia rf = iterate(5, (double d)->d*2);
System.out.println(rf.doit(1));
```

Funkcie.java

Java 8

```
String[] pole = { "GULA", "cerven", "zelen", "ZALUD" };
Comparator<String> comp =
(fst, snd)->Integer.compare(fst.length(), snd.length());

Arrays.sort(pole, comp);
for (String e : pole) System.out.println(e);
```

```
GULA
zelen
ZALUD
cerven
```

```
Arrays.sort(pole,
    (String fst, String snd) ->
        fst.toUpperCase().compareTo(snd.toUpperCase()));

for (String e : pole) System.out.println(e);
```

```
cerven
GULA
ZALUD
zelen
```

Funkcia.java

forEach, map, filter v Java8

```
class Karta {
   int hodnota;
   String farba;
   public Karta(int hodnota, String farba) { … }
   public void setFarba(String farba) { … }
   public int getHodnota() { … }
   public void setHodnota(int hodnota) { … }
   public String getFarba() { … }
   public String toString() { … }
}
List<Karta> karty = new ArrayList<Karta>();
karty.add(new Karta(7,"Gula"));
karty.add(new Karta(8,"Zalud"));
karty.add(new Karta(9,"Cerven"));
karty.add(new Karta(10,"Zelen"));
```

MapFilter.java

forEach, map, filter v Java8

```
                                   [Gula/7, Zalud/8, Cerven/9, Zelen/10]
karty.forEach(k -> k.setFarba("Cerven"));
                                   [Cerven/7, Cerven/8, Cerven/9, Cerven/10]
Stream<Karta> vacsieKartyStream =
   karty.stream().filter(k -> k.getHodnota() > 8);
List<Karta> vacsieKarty =
   vacsieKartyStream.collect(Collectors.toList());
                                                  [Cerven/9, Cerven/10]


List<Karta> vacsieKarty2 = karty
   .stream()
   .filter(k -> k.getHodnota() > 8)
   .collect(Collectors.toList());
                                                  [Cerven/9, Cerven/10]
```

MapFilter.java

forEach, map, filter v Java8

```
List<Karta> vacsieKarty3 = karty
   .stream()
   .map(k->new Karta(k.getHodnota()+1,k.getFarba()))
   .filter(k -> k.getHodnota() > 8)
   .collect(Collectors.toList());
```

```
[Cerven/9, Cerven/10, Cerven/11]
```

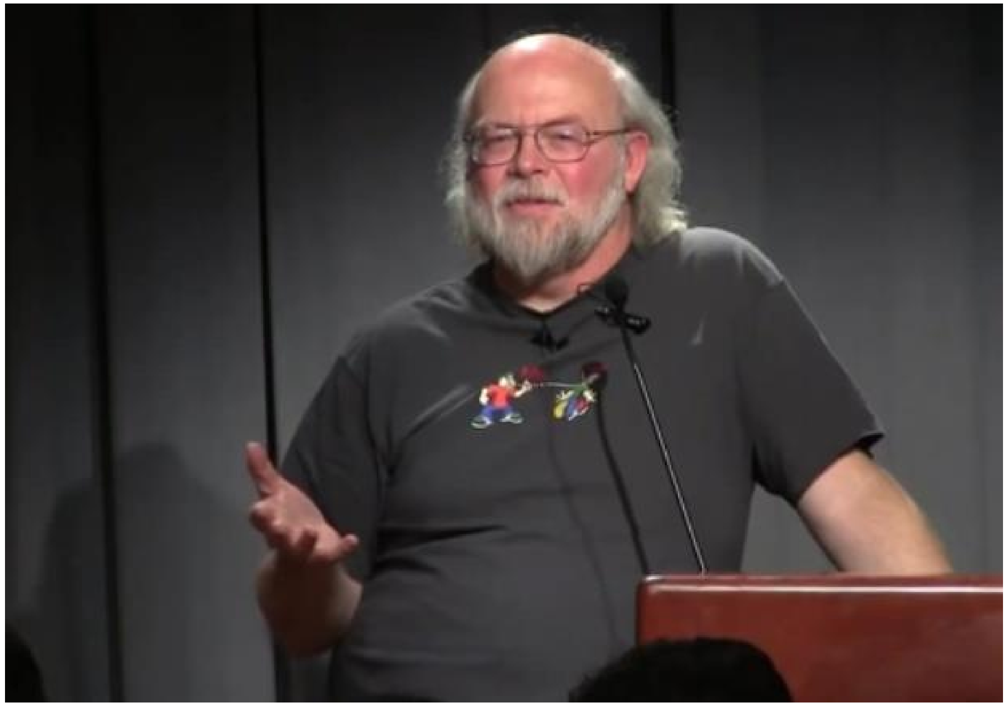
```
List<Karta> vacsieKarty4 = karty
   .stream()
   .parallel()
   .filter(k -> k.getHodnota() > 8)
   .sequential()
   .collect(Collectors.toList());
```

```
[Cerven/9, Cerven/10]
```

MapFilter.java

Break

(reklamná prestávna ï Scala publicity)



If I were to pick a language to use today other than Java, it would be Scala
-- James Gosling

Total-Prémie Top 10

IPSC pozvánka

http://ipsc.ksp.sk/

Prémiové body (umiestnenie v open division):
(počet tímov-vaše umiestnenie)/20 pre každého člena tímu (max.traja v tíme)

IPSC 2015 will take place from 18 June 2016, 11:00 UTC.
The practice session runs from 17 June 2016, 9:00 UTC.

Podobné:

ÅGoogle Code Jam
http://code.google.com/codejam

ÅACM Programming Contest
http://en.wikipedia.org/wiki/ACM_International_Collegiate_Programming_Contest

ÅTopcoder
http://www.topcoder.com/

Ako Ņalej

(ako nás stretnúŠ/nestretnúŠ)

JAVA2 alias Java EE (Pavel Petroviļ)

Å http://dai.fmph.uniba.sk/courses/java2/ (Login: java/vaja)

Å SieŠové aplikácie client/server

Å Distribuované výpoļ ty

Å Vyuģitie technológií XML v Jave

Å Práca s databázami z Javy

Å Servlety, JSP

VMA alias Vývoj mobilných aplikácií

ï Android Peter Borovanský, Windows ?, iPhone ?

Å http://dai.fmph.uniba.sk/courses/VMA/

Programovacie paradigmy

Å http://dai.fmph.uniba.sk/courses/PARA/