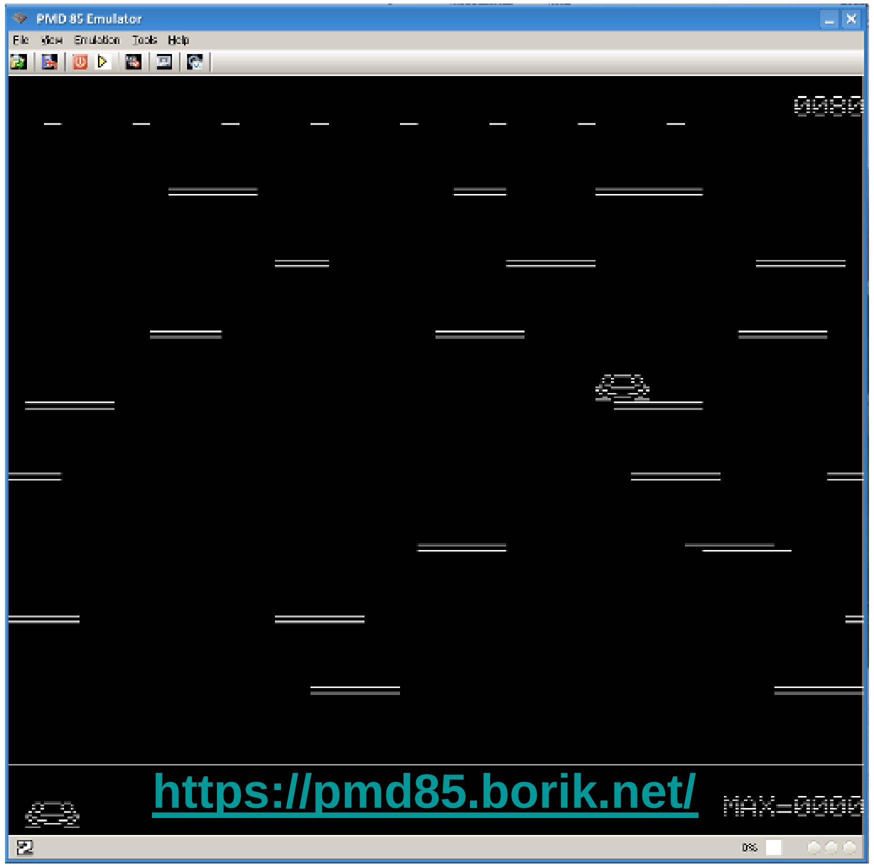
Quadterm 2

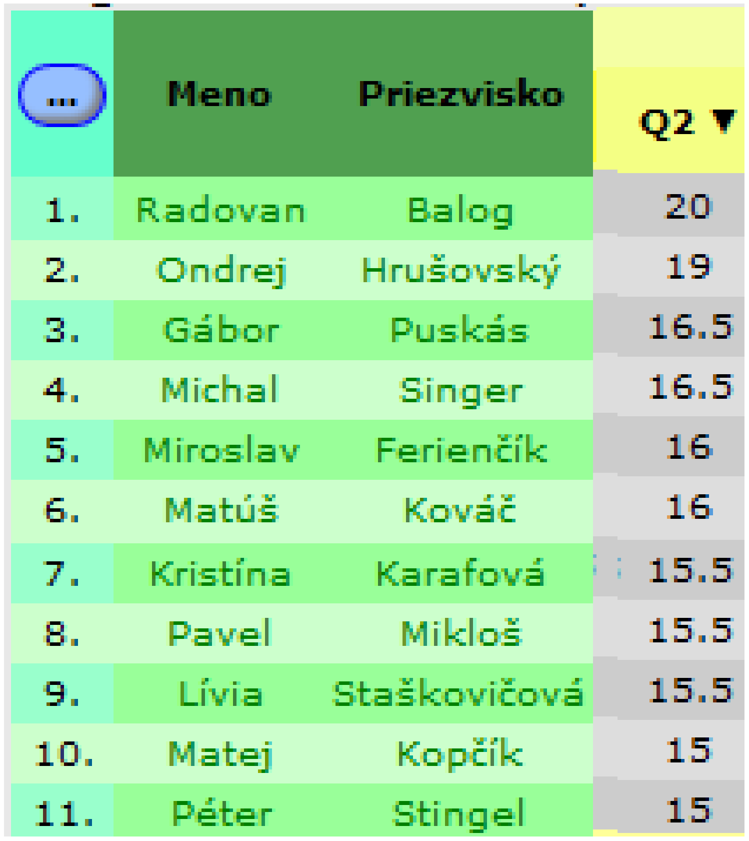
Å Ø 11,26, median: 12,5

Å korelácia

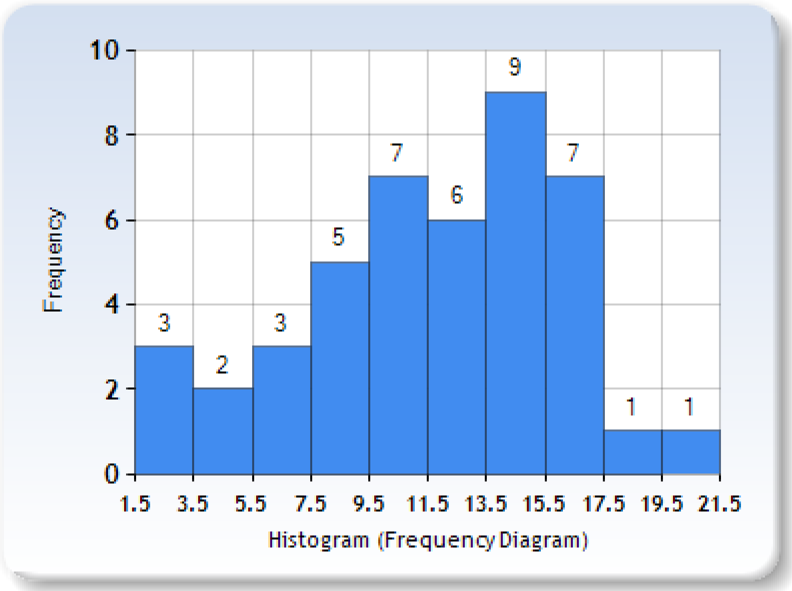
Å inverzná korelácia



https://pmd85.borik.net/



	…	Meno	Priezvisko	Q2 ▼
	1.	Radovan	Balog	20
	2.	Ondrej	Hrušovský	19
● ○	3.	Gábor	Puskás	16.5
○	4.	Michal	Singer	16.5
○	5.	Miroslav	Ferienčík	16
○	6.	Matúš	Kováč	16
● ○	7.	Kristína	Karafová	15.5
● ○	8.	Pavel	Mikloš	15.5
● ○	9.	Lívia	Staškovičová	15.5
	10.	Matej	Kopčík	15
● ○	11.	Péter	Stingel	15



Posledná prednáġka

(informatívna)

Tri témy (nijako nesúvisiace):

Å backtracking ï ako forma prehǍadávania stavovéo priestoru (grafu),

Å reflexivita ï nieļ o nevídané v kompilovaných jazykoch,

Å funkcionálna java 1.8 ï funkcionálne programovanie uġ aj Jave (od 1.8)

Závereļné slovo

DFS/BFS/Bactracking

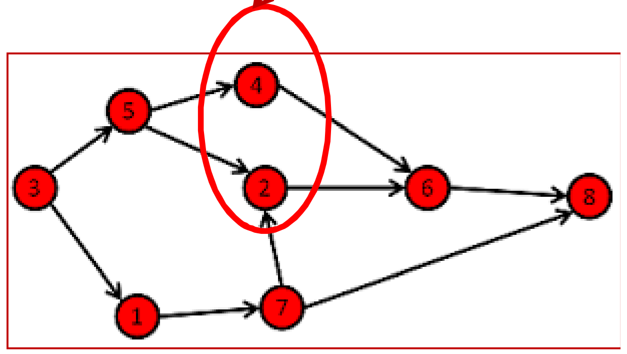
Ide o prehǍdávanie stavového priestoru, abstrakcia pre stav môǵe byŠ

```
interface State {
    public State();                            // počiatočný stav hľadania
    abstract boolean isFinalState();           // test na koncový stav hľadania
    abstract State[] next();                   // nasledujúci/susedný stav
    abstract boolean isCorrect();              // test na korektnosť stavu
}
```

State(5).next()

Naivné prehǍdávanie pre acyklický graf, ktoré sa na cyklickom zacyklí

```
public class Search<S extends State> // hľadáme cestu do finálneho stavu
public void searchWhichLoops(S s) {
    if (s.isFinalState())
          add(s);// pridaj do zoznamu riešení
    else
          for (State ns : s.next())
             search((S) ns); // rekurzia do susedov
}
```



Aby sa to nezacyklilo

(objavila Ariadna pri hľadaní Thezea v labyrinte s Minotaurom)

```
public void search(S s, ArrayList<S> visited) {
   if (s.isFinalState())
        add(s); // pridaj do zoznamu riešení
   else
        for (State ns : s.next()) {
             if (!visited.contains(ns)) {// nebol si ?
                  visited.add((S) ns); // označ
                  search((S) ns, visited);
                  visited.remove(ns); // odznač
             }
        }
}
```

ARIADNE GAVE THESEUS A BALL OF THREAD
WHICH HELPED HIM ESCAPE THE LABYRINTH
AFTER HE KILLED THE MINOTAUR

BTW, je to depth-first alebo breadth-first ?

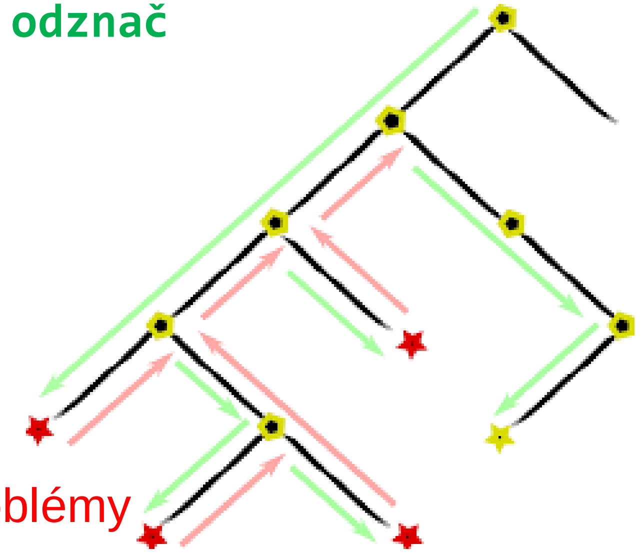
Backtracking

(orezáva podstromy urļ ite neobsahujúce rieġenie)

```
public void search(S s, ArrayList<S> visited) {
   if (s.isFinalState())
        add(s);
   else
        for (State ns : s.next()) { // môže to viesť k riešeniu ?
                if (!visited.contains(ns) && ns.isCorrect()) {
                        visited.add((S) ns); // označ
                        search((S) ns, visited);
                        visited.remove(ns); // odznač
                }
        }
}
```

Ġikovný `isCorrect` výrazne zredukuje zväļ ġa exponenciálny priestor stavov, ale ten aj tak zostane exponenciálny

Preto: backtrack nepouģívame na neexponenciálne problémy



Ako by vyzeral BFS

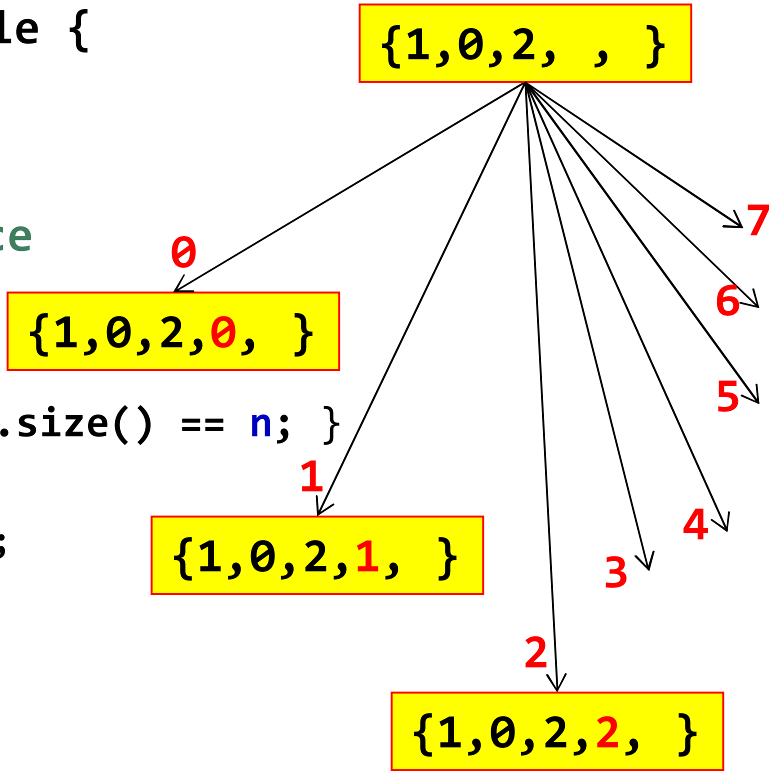
```
private void search(ArrayList<S> queue, ArrayList<S> visited, boolean DFS) {
    while (queue.size() > 0) {
          S s = queue.remove(0);          // vyber prvý z fronty
          if (s.isFinalState())           // ak si už v cieli
                add(s);                   // pridaj do zoznamu riešení
          else
              for (State ns : s.next()) {
                  if (!visited.contains(ns) && ns.isCorrect()) {
                     visited.add((S) ns);
                     if (DFS)             // ak depth-first search
                         queue.add(0, (S) ns); // pridaj na začiatok fronty
                     else                 // ak breadth-first search
                         queue.add(queue.size(), (S) ns);// pridaj na koniec
                  }
              }
    }
}
```

n=5

k=8

n-tice s prvkami 0..(k-1)

```
class NTuple implements State, Cloneable {
   int n; // dĺžka n-tice
   int k; // prvky n-tice sú 0..k-1
   ArrayList<Integer> v; // prvky n-tice
   public NTuple(int n, int k) { …
   public boolean equals(Object o) { …
   public boolean isFinalState() { return v.size() == n; }
   public NTuple[] next()  {
    HashSet<NTuple> next = new HashSet<NTuple>();
    for(int i=0; i<k; i++) {
       NTuple s = new NTuple(n,k);
       s.v = (ArrayList<Integer>)v.clone();
       s.v.add(i);
       next.add(s);
    }
   return next.toArray(new NTuple[]{});
}
public boolean isCorrect() { return true; }
```



n=5

k=8

Aplikovaná kombinatorika

(rôzne verzie **isCorrect**)

- dostaneme n-prvkové variácie s opakovaním, $k^n = 8^5 = 32768$

{8*8*8*8*8}

```
public boolean isCorrect() {
   return new HashSet(v).size() == v.size(); // je to množina
```

{8*7*6*5*4}

- dostaneme n-prvkové variácie s bez opakovania, n*(n-1)*..*(n-k+1)=8*7*6*5*4 = 6720

```
int size = v.size();
Integer last = v.get(size-1); // prepíšeme hrozný test na opakujúce prvky
return (size == 0 || v.indexOf(last) == size-1); // posledný sa už nachádza ?
```

```
return (size < 2 || v.get(size-2) < v.get(size-1)); // prvky sú ostro-rastúce
```

- dostaneme k-prvkové kombinácie bez opakovania, $\binom{k}{n} = \binom{8}{5}$ = 56

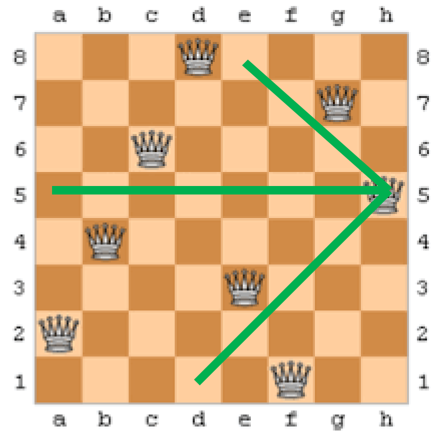
```
return (size < 2 || v.get(size-2) <= v.get(size-1)); // prvky sú neklesajúce
```

- dostaneme k-prvkové kombinácie s opakovaním, $\binom{n+k-1}{k-1} = \binom{5+8-1}{8-1}$ = 792

- ak n=k, tak sú to permutácie, n! = 5 ! = 120.

8-dám

(zmena len v **isCorrect**)



Staļ í len zmeniŠisCorrect

```
public boolean isCorrect() {
   int size = v.size();
   Integer last = v.get(size-1); // posledná dáma neohrozuje predchádzajúce
   for(int i = 1; i <= size-1; i++)
        if (v.get(size-i-1) == last || // sa nesmie stretnúť s inou v riadku
            v.get(size-i-1) == last + i || // ani na uhlopriečke
            v.get(size-i-1) == last - i)   // ani na druhej uhlopriečke
               return false; // ak to nastane, nikdy z toho nebude riešenie
   return true; // inak je to zatiaľ korektné čiastočné riešenie
}
```

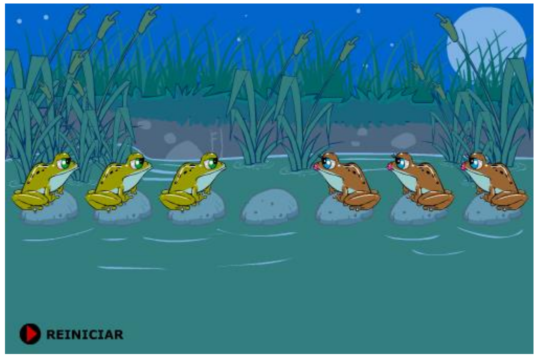
```
[[2, 4, 7, 3, 0, 6, 1, 5], [2, 4, 1, 7, 0, 6, 3, 5], [2, 4, 1, 7, 5, 3, 6, 0], [2, 4, 6, 0, 3, 1, 7, 5], [2, 6, 1, 7, 4, 0, 3, 5], [2, 6, 1, 7, 5, 3, 0, 4], [2, 5, 3, 0, 7, 4, 6, 1], [2, 5, 3, 1, 7, 4, 6, 0],
[2, 5, 1, 6, 4, 0, 7, 3], [2, 5, 1, 6, 0, 3, 7, 4], [2, 5, 1, 4, 7, 0, 6, 3], [2, 5, 7, 1, 3, 0, 6, 4], [2, 5, 7, 0, 3, 6, 4, 1], [2, 5, 7, 0, 4, 6, 1, 3], [2, 0, 6, 4, 7, 1, 3, 5], [2, 7, 3, 6, 0, 5, 1, 4],
[5, 0, 4, 1, 7, 2, 6, 3], [5, 7, 1, 3, 0, 6, 4, 2], [5, 1, 6, 0, 3, 7, 4, 2], [5, 1, 6, 0, 2, 4, 7, 3], [5, 2, 6, 3, 0, 7, 1, 4], [5, 2, 6, 1, 3, 7, 0, 4], [5, 2, 6, 1, 7, 4, 0, 3], [5, 2, 4, 6, 0, 3, 1, 7],
[5, 2, 4, 7, 0, 3, 1, 6], [5, 2, 0, 6, 4, 7, 1, 3], [5, 2, 0, 7, 3, 1, 6, 4], [5, 2, 0, 7, 4, 1, 3, 6], [5, 3, 1, 7, 4, 6, 0, 2], [5, 3, 0, 4, 7, 1, 6, 2], [5, 3, 6, 0, 2, 4, 1, 7], [5, 3, 6, 0, 7, 1, 4, 2],
[6, 4, 2, 0, 5, 7, 1, 3], [6, 0, 2, 7, 5, 3, 1, 4], [6, 2, 7, 1, 4, 0, 5, 3], [6, 2, 0, 5, 7, 4, 1, 3], [6, 1, 5, 2, 0, 3, 7, 4], [6, 1, 3, 0, 7, 4, 2, 5], [6, 3, 1, 7, 5, 0, 2, 4], [6, 3, 1, 4, 7, 0, 2, 5],
[4, 7, 3, 0, 6, 1, 5, 2], [4, 7, 3, 0, 2, 5, 1, 6], [4, 0, 7, 5, 2, 6, 1, 3], [4, 0, 7, 3, 1, 6, 2, 5], [4, 0, 3, 5, 7, 1, 6, 2], [4, 1, 5, 0, 6, 3, 7, 2], [4, 1, 3, 5, 7, 2, 0, 6], [4, 1, 3, 6, 2, 7, 5, 0],
[4, 1, 7, 0, 3, 6, 2, 5], [4, 6, 1, 3, 7, 0, 2, 5], [4, 6, 1, 5, 2, 0, 3, 7], [4, 6, 1, 5, 2, 0, 7, 3], [4, 6, 3, 0, 2, 7, 5, 1], [4, 6, 0, 2, 7, 5, 3, 1], [4, 6, 0, 3, 1, 7, 5, 2], [4, 2, 0, 6, 1, 7, 5, 3],
[4, 2, 0, 5, 7, 1, 3, 6], [4, 2, 7, 3, 6, 0, 5, 1], [0, 6, 4, 7, 1, 3, 5, 2], [0, 6, 3, 5, 7, 1, 4, 2], [0, 4, 7, 5, 2, 6, 1, 3], [0, 5, 7, 2, 6, 3, 1, 4], [1, 7, 5, 0, 2, 4, 6, 3], [1, 4, 6, 3, 0, 7, 5, 2],
[1, 4, 6, 0, 2, 7, 5, 3], [1, 5, 7, 2, 0, 3, 6, 4], [1, 5, 0, 6, 3, 7, 2, 4], [1, 3, 5, 7, 2, 0, 6, 4], [1, 6, 4, 7, 0, 3, 5, 2], [1, 6, 2, 5, 7, 4, 0, 3], [3, 6, 0, 7, 4, 1, 5, 2], [3, 6, 2, 7, 1, 4, 0, 5],
[3, 6, 4, 1, 5, 0, 2, 7], [3, 6, 4, 2, 0, 5, 7, 1], [3, 1, 4, 7, 5, 0, 2, 6], [3, 1, 6, 4, 0, 7, 5, 2], [3, 1, 6, 2, 5, 7, 0, 4], [3, 1, 6, 2, 5, 7, 4, 0], [3, 1, 7, 5, 0, 2, 4, 6], [3, 1, 7, 4, 6, 0, 2, 5],
[3, 5, 0, 4, 1, 7, 2, 6], [3, 5, 7, 2, 0, 6, 4, 1], [3, 5, 7, 1, 6, 0, 2, 4], [3, 7, 4, 2, 0, 6, 1, 5], [3, 7, 0, 4, 6, 1, 5, 2], [3, 7, 0, 2, 5, 1, 6, 4], [3, 0, 4, 7, 5, 2, 6, 1], [3, 0, 4, 7, 1, 6, 2, 5],
[7, 3, 0, 2, 5, 1, 6, 4], [7, 1, 4, 2, 0, 6, 3, 5], [7, 1, 3, 0, 6, 4, 2, 5], [7, 2, 0, 5, 1, 4, 6, 3]]
```

92 rieģení

```
[>>><_<<, >>_<><<, >_><><<, ><>_><<, ><><>_<, ><><><_, ><><_<>, ><_<><>, _<><><>, <_><><>, <<>_><>, <<><>_>, <<><_>>, <<_<>>>, <<<_>>>]
[>>_><<<, >><>_<<, >><><_<, >><_<><, >_<><><, _><><><, <>_><><, <><>_><, <><><>_, <><><_>, <><_<>>, <_<><>>, <<_><>>, <<<>_>>, <<<_>>>]
```

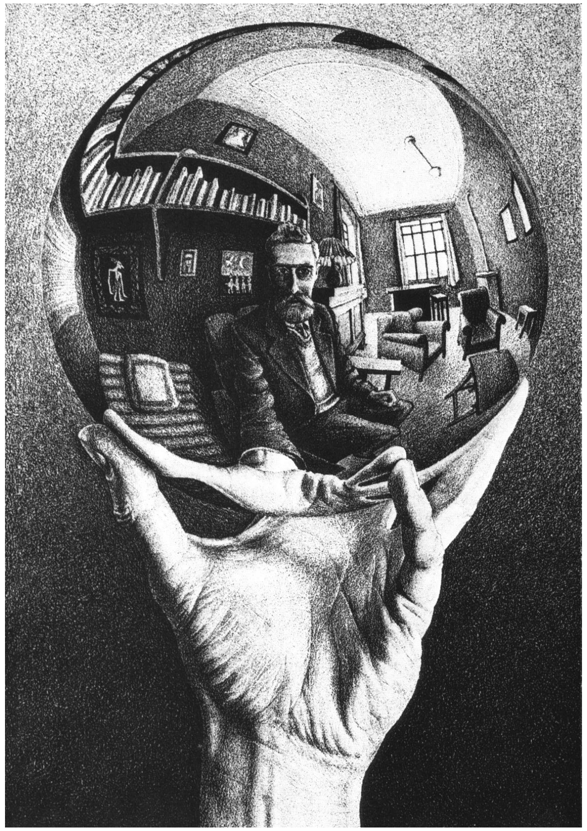
Ģabky

```
class ZabkyState implements State {
String z7;                      // šesť žiab, 3 pravé >>>, 3 ľavé <<<
public ZabkyState() {           // počiatočný stav
   z7 = ">>>_<<<";
}
public boolean isFinalState() { // koncový stav
   return z7.equals("<<<_>>>");
}
public ZabkyState[] next()  {
   ArrayList<ZabkyState> nxt = new ArrayList<ZabkyState>();
   nxt.add(new ZabkyState(z7.replace("_<", "<_"))); // ľavá cez medzeru
   nxt.add(new ZabkyState(z7.replace(">_", "_>"))); // pravá cez medzeru
   nxt.add(new ZabkyState(z7.replace("_><", "<>_"))); // ľavá cez pravú
   nxt.add(new ZabkyState(z7.replace("><_", "_<>"))); // pravá cez ľavú
   nxt.remove(this); nxt.remove(this);
   return nxt.toArray(new ZabkyState[]{});
}
```



Reflexivita

(Java Reflection Model)



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

Å je to vlastnosŞ ktorá sa vyskytuje v interpretovaných jazykoch, napr. exec a eval v Pythone, nie v kompilolvaných (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)

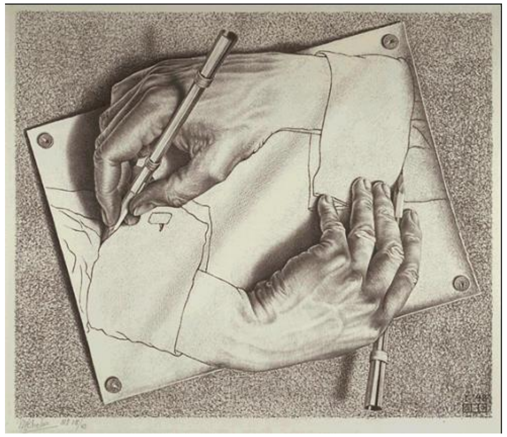
```
public class Nadtrieda implements Runnable {
            public int variabla;
            public int[] pole = {1,2,3};
            public String[] poleStr = {"janko", "marienka" };
            public Nadtrieda()  {   }
            public Nadtrieda(int a) {   }

            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)

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

Kaġdý objekt pozná metódu getClass():

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

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

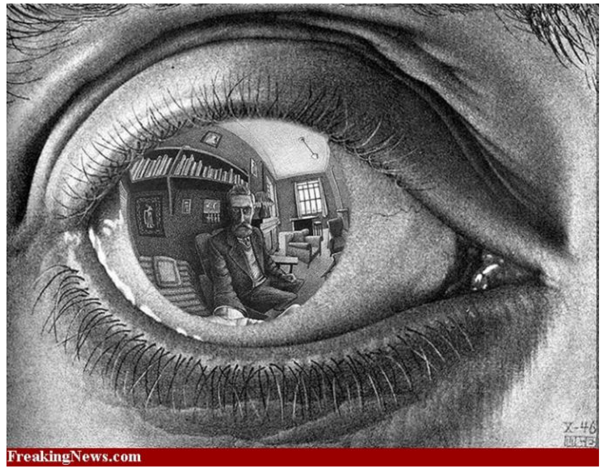
```
        System.out.println(nt.getName());          // Nadtrieda

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(ñé  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 poloģky (premenné) triedy

Å Field getField(String name) poloģka 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 )
```

Premenné, konštruktory

```
for (Field f : nt.getFields() ) {
        String fieldName = f.getName();
        Class typeClass = f.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;");
```

```
for (Constructor c : nt.getConstructors()) {
   System.out.print("( ");

   for (Class parameterType : c.getParameterTypes() ) {
        String parameterString = parameterType.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);

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

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

```
GULA
zelen
ZALUD
cerven
```

```
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]
```

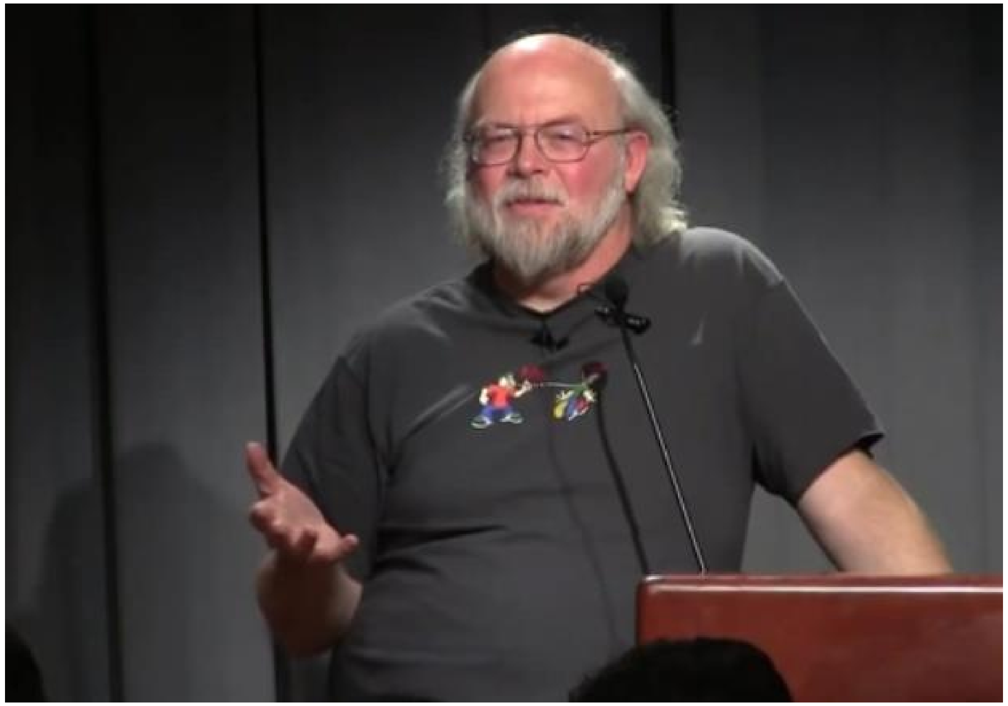
```
List<Karta> vacsieKarty4 = karty
    .stream()
    .parallel()
    .filter(k -> k.getHodnota() > 8)
    .sequential()
    .collect(Collectors.toList());
```

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

MapFilter.java

Prekvapivé finále

(Scala publicity)



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

Anketa

Å výsledky ankety dôleġite pre Ņalġí rok, pre vylepġenie bak.programu

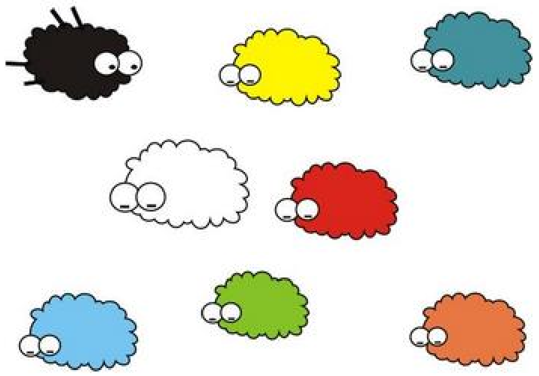
Å výsledky ankety ļítajú:

ï vyuļujúci

ï budúci ġtudenti

ï garant a vedúci katedry

ï v ojedinelých prípadoch dekan



Å Minianketa:

Å https://docs.google.com/forms/d/1isuNPsEitnqvpxG1Wr7RrCX_8tfFOSlQPvsrIHneUgc/edit

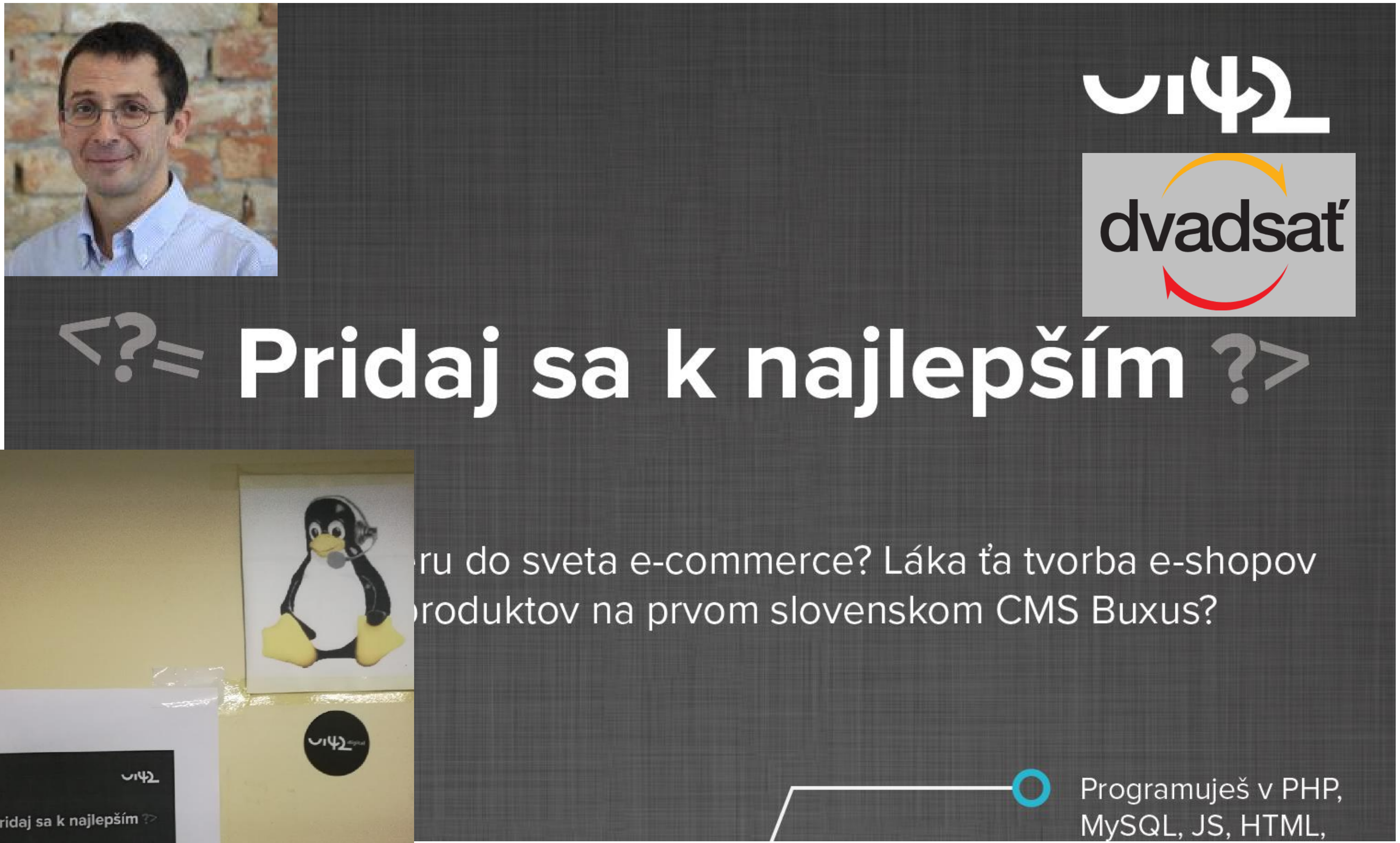
Nebuď ovce!

Total-Prémie Top 10

Å TBA

Nic není černobílé!

Ġikových je ako ġafránu



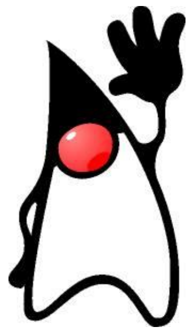
<?= Pridaj sa k najlepším ?>

ru do sveta e-commerce? Láka ťa tvorba e-shopov
roduktov na prvom slovenskom CMS Buxus?

Programuješ v PHP,
MySQL, JS, HTML,

Ako a ļ o Ņalej

(ako nás stretnúŠ/nestretnúŠ)



VMA alias Vývoj mobilných aplikácií (ja) paralelne beģí iOS (fy. touch4IT.sk)

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

Programovacie paradigmy (Konkurentné GO, Funkcionálny Haskell, Logický Prolog)

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

FPRO ï Funkcionálne programovanie (ja) ï magisterskom prog.

http://dai.fmph.uniba.sk/courses/FPRO/

Efektívne algoritmy (T.VinaŚ)

Programovanie - 5 alias C# (p.Salanci)

JAVA2 alias Java EE (Pavel Petroviļ) ï presunuté do magisterského prog. ☹

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

Å SieŠové aplikácie client/server, Distribuované výpoļ ty

Å Vyuģitie technológií XML v Jave, Servlety, JSP

IPSC pozvánka

(ĂNatri svojho profákañ)

http://ipsc.ksp.sk/

Svetová programátorská súŠaģ 1-3 ļ lenných tímov, organizuje naġa banka KSP.sk
bez obmedzení (open division)
Åkedysi sme boli LazySnails (MW+1)
Återaz GigaStep (s dvomi ex-ġtudentami) 254/790

IPSC 2017 will take place from ??????????????????????????????
The practice session runs from

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

Podobné súťaže:
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/

Historické jadro KSP.sk

(roļ níky 1-5 z dneģných 35)

Organizaļ ne

Å oprava quad-midtermov v H3, prídite bez registrácie v AISe,

Å na preplnený termín 15.6. bude rozġírený ï p.Gyarfaġ ponúkol pár miest v H6

Å v ļ ase 3.-12.6. som mimo, bez javovátka, asi ani nebudem odpovedaŠna maily,

Å preto píġte na prog4java@, snáŅ vás niekto obslúģi,

Å moje projekty, ktoré chcú prísŠna skúġku 15.6. opravím/obodujem urļ ite do 14.6., takģe body za projekt budú, ale feedback niļ moc...

Å Kurz bol aký bol vŅaka celému tímu: Peter, Juraj, Andrej, Patrícia, Jozef.