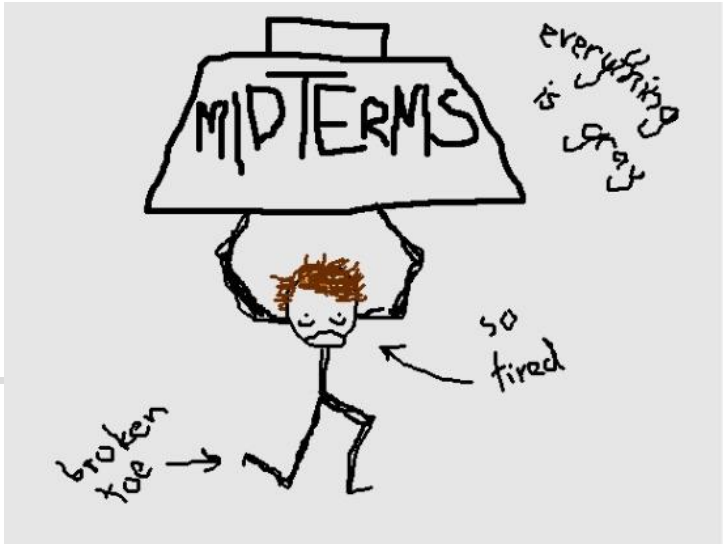
Komponenty jazyka

Peter Borovanský
KAI, I-18

borovan 'at' ii.fmph.uniba.sk
http://dai.fmph.uniba.sk/courses/JAVA/

Midterm quadterm



Naše termíny sú:

- 11.apríla 18:00 je Midterm A
- 17. marca je quadterm na cvičeniach, každý na svojom cvičení
- 3.marca je Bypass excelencie, prihlásiť sa treba do 1.3. mailom

Prémiové úlohy:

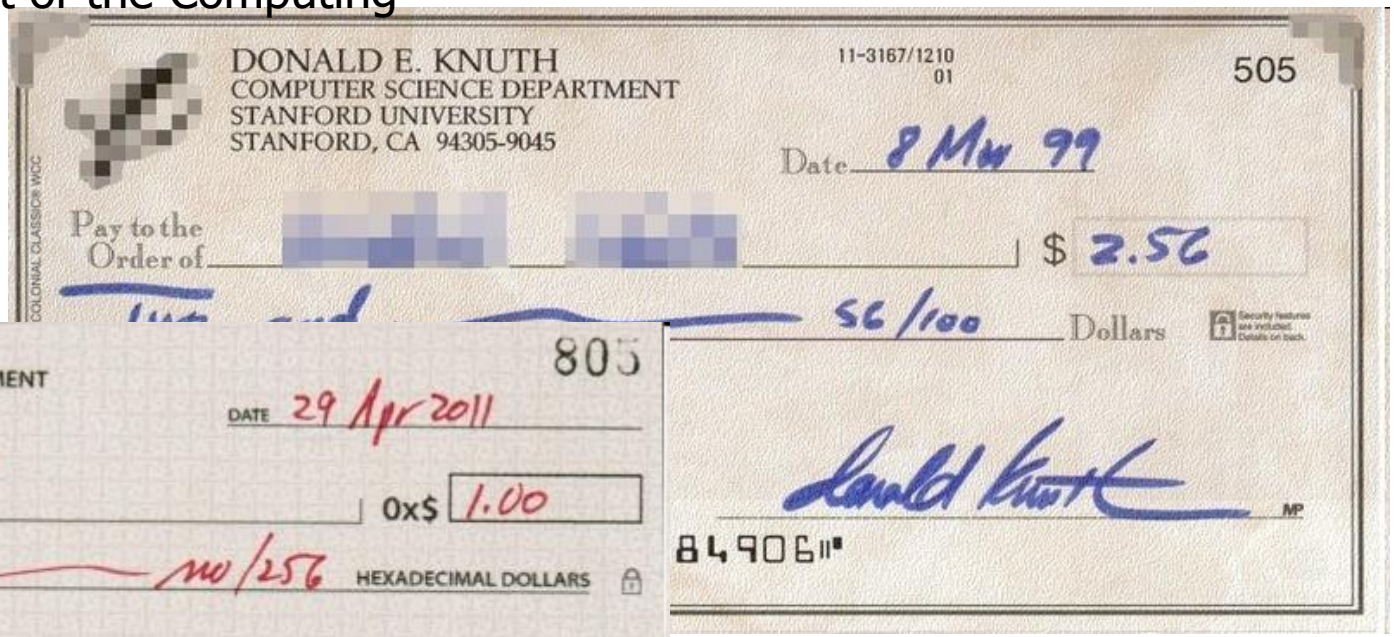
- sa rozbehli ☺
- riešte ich, najmä, ak vám štandardný kurz príde ako *slabá káva,*
- všetky body z prémií sa vám počítajú do známky, len dve kritéria na automatické A a Fx ich ignorujú, viac na stránke predmetu
- časom budeme zverejnovať týždňový rebríček Galéria Top 10

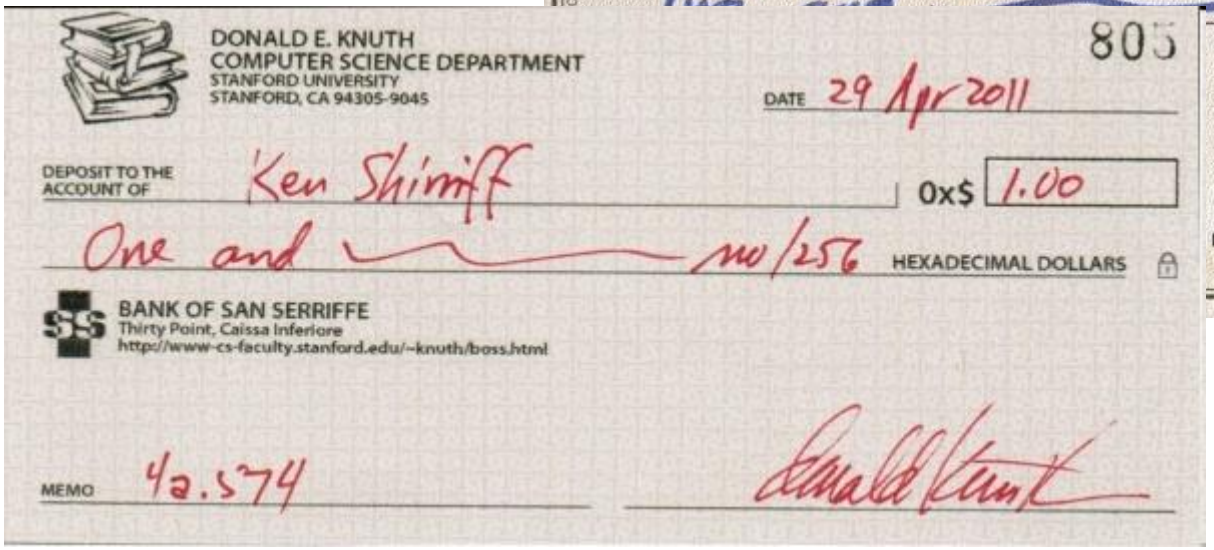
Blogy (diskusia):

- prosíme, nezamieňať si to s inými socialnymi sieťami, a otázky so statusmi…
- striktne zakázane zverejnovať (tam) riešenia, návody, uplné i čiastočné. ...

Donald E.Knuth

Naše chyby v zadaniach si priznáme, sme radi, že ich odhalíte
čím skôr, a ak sú relevantné (nie typo a gramatika), tak aj oceníme
Errata: The Art of the Computing



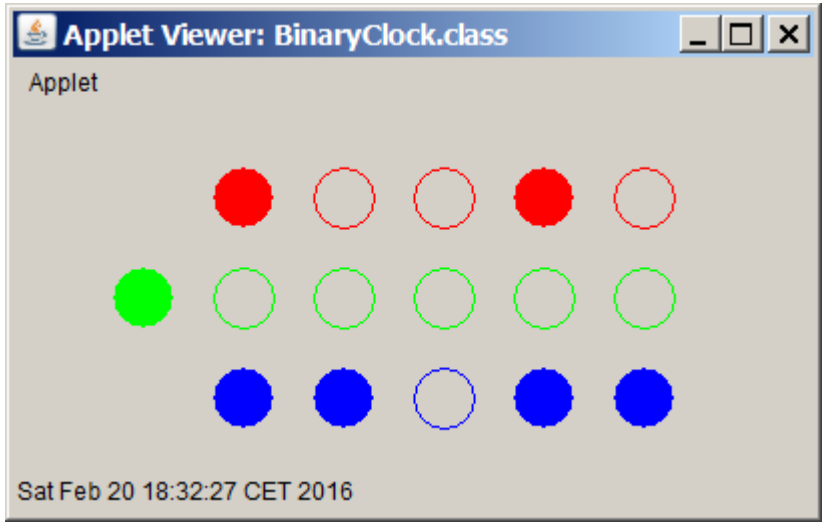


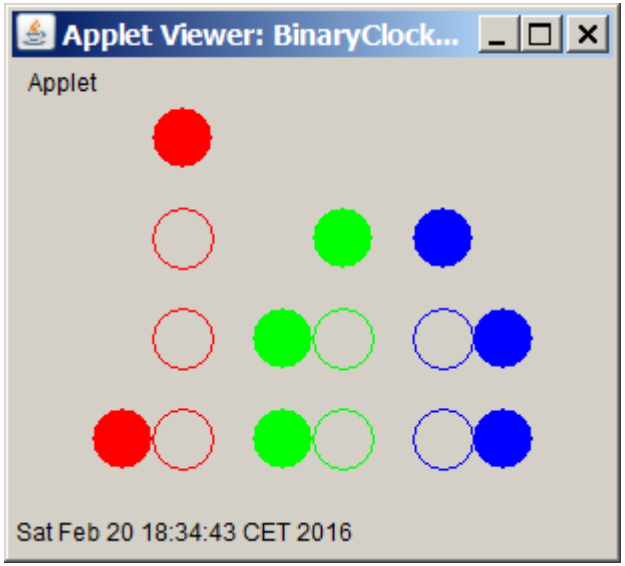
Testy
(sú iné ako ste čakali)

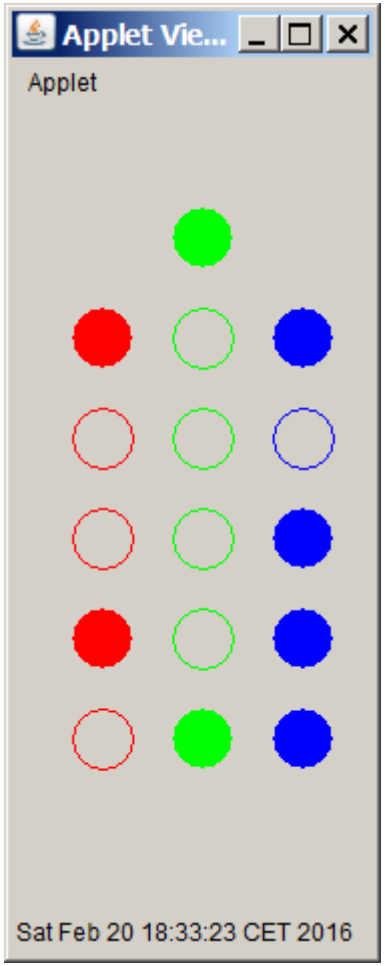
- používame techniku štandardných java junit testov, viac v prednáške
- všeobecne: dodržiavanie predpísaných mien súborov, tried a metód
- keďže rekord submitov je > 15, vás 60, neprezeráme všetky, ale len posledný, preto do posledného dajte všetky dobré riešenia celej zostavy
- na cvičení bude-bola súťaž o najlepšieho testera (v písaní junit testov)
- testy nezverejňujeme (okrem Quad), ani neplánujeme, ale zdrojáky ľ ítame !
- ak to nie je uvedené v zadaní, môžete sa pýtaŠna rozsah vstupov
- 90 s. je timeout pre každého, ak to nie je inak junit testom upravené
- automatické testy typu *obodujem sa sám* sú pripravené.

Na cvičení nebolo

- precvičili sme si binárne operácie s číslami, >>,<<,>>>,^
- naprogramovali sme konzolovú verziu binárnych hodiniek
- argumenty vyčítané z príkazového riadku, z poľa args

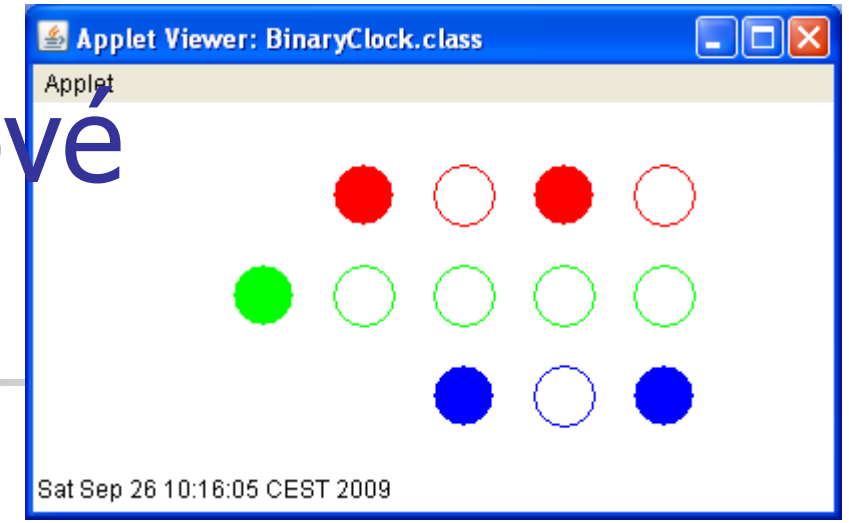






Súbor BinaryClock,BinaryClock1,BinaryClock2.java

Binárne konzolové hodinky




```
1010
10000
101
--------
```

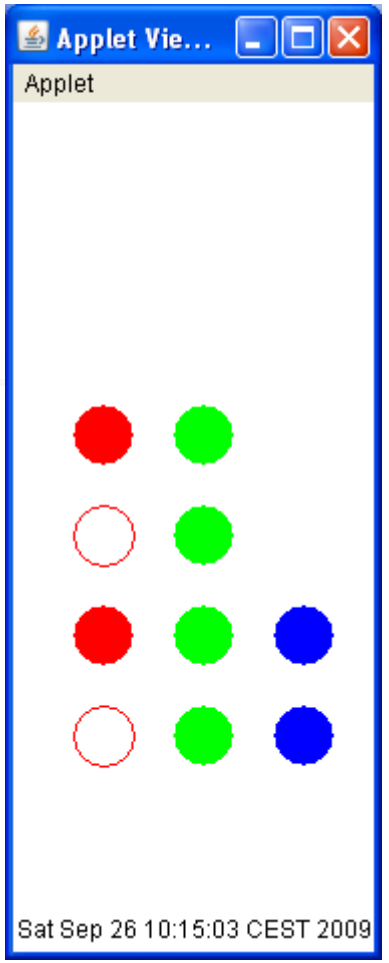
```
int second = 0;
for(;;) {                                                    // loop forever
  Calendar calendar = new GregorianCalendar();               // aktualny casu
  int hodina = calendar.get(Calendar.HOUR_OF_DAY);// hodina, format 0..24
  int minuta = calendar.get(Calendar.MINUTE);                // minuta
  int sekunda = calendar.get(Calendar.SECOND);               // sekunda
  if (second != sekunda) {                                   // nevieme pouzit Timer
    System.out.println(Integer.toBinaryString(hodina));
    System.out.println(Integer.toBinaryString(minuta));
    System.out.println(Integer.toBinaryString(sekunda));
    System.out.println("--------------------");
    second = sekunda;                                  // tak urobime takyto CPU killer hack
  }
}
```

Súbor BinaryWatch2.java

Binárne konzolové hodinky

riadok(hodina,minuta,sekunda);

```
1 1 0
0 1 0
1 1 1
0 1 1
-------
```



Rekurzívna procedúra, ktorá vypíše najprv horné bity čísla (okrem posledného) a potom posledný bit

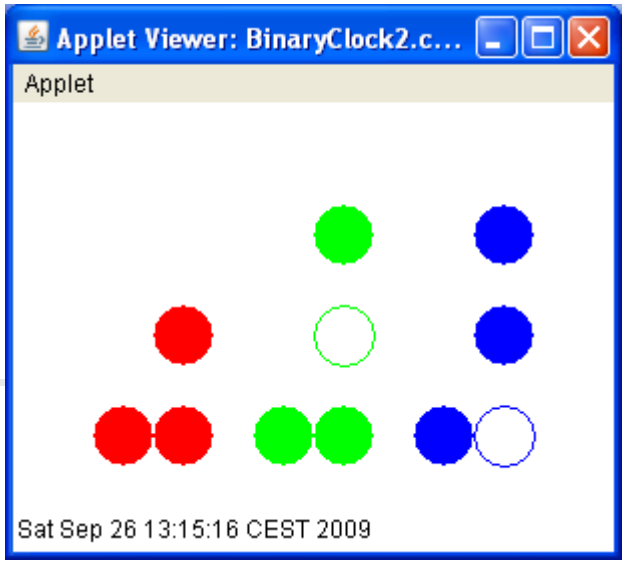
```
static void riadok(int h, int m, int s) {
   if (h+m+s == 0)
      return;
   riadok(h/2, m/2, s/2);
   System.out.println(" "+ (h&1) + " " + (m&1) + " " + (s&1));
```

Čo by sa stalo, ak vymením posledné dva riadky kódu ?

Súbor BinaryWatch5.java

Binárne konzolové hodinky

00 01 01
01 00 01
11 11 10



riadok(hodina/10,hodina%10,minuta/10,minuta%10,sekunda/10, sekunda%10);

```
static void riadok(int h1, int h2, int m1, int m2, int s1, int s2) {
  if (h1+h2+m1+m2+s1+s2 == 0)
     return;
 riadok(h1/2, h2/2, m1/2, m2/2, s1/2, s2/2);
 System.out.println((h1&1)+(h2&1)+" "+(m1&1)+(m2&1)+" "+(s1&1)+(s2&1));
```

Súbor BinaryWatch4.java

Argumenty príkazového riadku

Iná možnosť, ako dostať do programu naše dáta, je pomocou príkazového riadku, ktorým sa spúšťa interpretovaný program. Pole arg je reťazcové pole argumentov, ktoré sú v príkazovom riadku oddelené medzerou.

```
public class MainArg {

  public static void main(String[] args) {

    for(int i = 0; i < args.length; i++)
          System.out.println(i+ "=" + args[i]);
  }

}
```

args[0] = ”prvy”
args[1] = ”druhy”
args[2] = ”treti”

args.length = 3

```
> java MainArg prvy druhy treti
0=prvy
1=druhy
2=treti
```

Súbor: MainArg.java

Celočíselné argumenty

Napriek tomu, že každý argument príkazového riadku je reťazec, možeme ho chcieť intepretovať ako napr. číslo

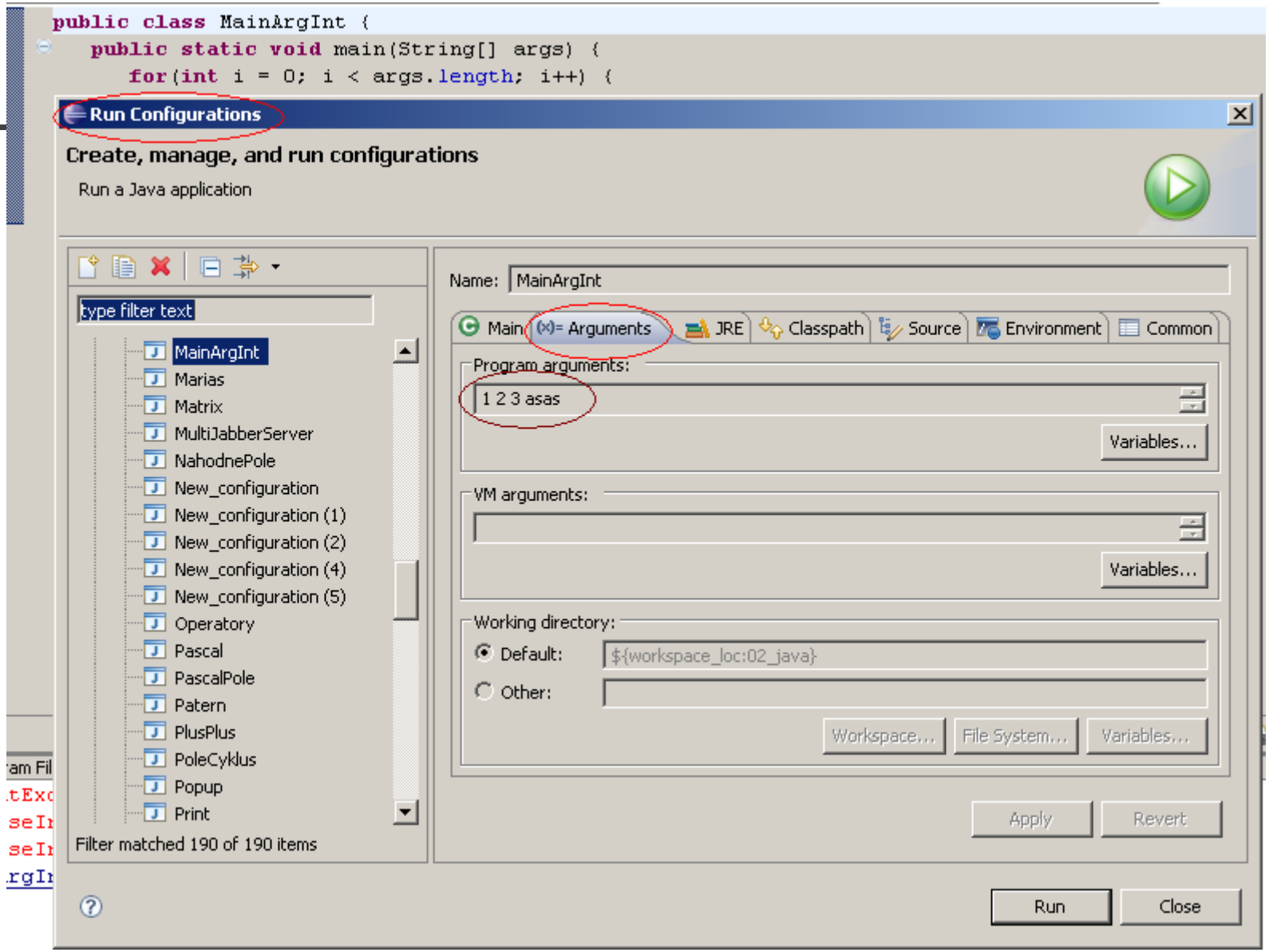
```
public class MainArgInt {

   public static void main(String[] args) {

    for(int i = 0; i < args.length; i++) {
       int n = Integer.parseInt(args[i]);
       System.out.println("args["+i+ "]=" + n);
    }
  }
}
```

konverzia String->int môže byť neúspešná

```
> java MainArgInt 1 2 3 asas
args[0]=1
args[1]=2
args[2]=3
Exception in thread "main" java.lang.NumberFormatException:
For input string: "asas"
at ... at MainArgInt.main(MainArgInt.java:4)
```

Súbor: MainIntArg.java

Ako zadať argumenty

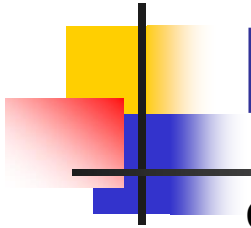


Fibonacci

```
public class Fibonacci {

  public static void main(String[] args) {
     int N = Integer.parseInt(args[0]);
     long a = 1;
     long b = 0;
     while (N-- > 0) {
            System.out.println(b);
            a = a + b;
            b = a - b;
     }
  }
}
```

```
> java Fibonacci 10
10
0
1
1
2
3
5
8
13
21
34
```

Súbor: Fibonacci.java



Komponenty jazyka

dnes bude:

- konverzie (základných) typov, reťazce a práca s nimi
- polia (pohľad C++ programátora)
- statické metódy (procedúry a funkcie) a statické bloky
- testovanie (prvý JUnit test)

cvičenia:

- programy s poľami, testovanie a ľadenie (eclipse debugger)
- manipulácia s reťazcami

literatúra:

- Thinking in Java, 3rd Ed. (http://www.ibiblio.org/pub/docs/books/eckel/TIJ-3rd-edition4.0.zip) – 3:Controlling Program Flow, 4:Initialization & Cleanup,
- Naučte se Javu – úvod
  - http://interval.cz/clanky/naucte-se-javu-operatory-a-ridici-prikazy/,
  - http://interval.cz/clanky/naucte-se-javu-staticke-promenne-a-metody-balicky/
- Java (http://v1.dione.zcu.cz/java/sbornik/)

Pretypovanie, konverzie

```
char    c = 'A';
int     i = (int) c;        // konverzia do nadtypu
char    d = (char)i;        // redukcia do podtypu (cast)
```

- rozširujúce konverzie (do nadtypu):
  byte->short->int->long->float->double
- zužujúce konverzie (do podtypu):
  double->float->long->int->short->byte

```
short s = 300;                  // 16 bit [2^15-1 .. 2^15]
byte b ;                        // 8 bit [-128..127]

b = (byte) s;                   // s = 300, b = 44      (300-256)
b = (byte) 255;                 // b = -1               (255-256)
byte bb = 126;                  // bb = 126
bb += 3;                        // bb = -127            (126+3-256)
bb = -126;                      // bb = -126
bb += -5;                       // bb = 125             (-126-5+256)
```

Súbor Konvert.java

Príklad statických metód z tried
Integer, String, Double a Boolean

Konverzie z/do String

Integer.toString(k,2).length()
Integer.toString(x,5).length()

String -> int
Integer.valueOf("123")
Integer.parseInt("123")

int -> String
String.valueOf(123)
Integer.toString(123,10)

Integer.toBinaryString(31) //11111
Integer.toOctalString(15) // 17
Integer.toHexString(255) // ff

String -> double
Double.valueOf("3.1415")
Double.parseDouble("3.1415")

double -> String
String.valueOf(Math.PI)
Double.toString(Math.PI)

String -> Boolean
Boolean.valueOf("true")
Boolean.parseBoolean("false")

Boolean -> String
String.valueOf(**true**)
Boolean.toString(**false**)

Súbor KonverzieString.java

Reťazce – metódy

```
byte[] bajty = {
    (byte)'E', (byte)'v', (byte)'a'};
char[] znaky = {
    'M', 'a', 'r', 't', 'i', 'n', 'a'};

String
 s1 = new String("cao"),
 s2 = new String(s1),
 s3 = new String(bajty),
 s4 = new String(bajty, 1, 2),
 s5 = new String(znaky),
 s6 = new String(znaky, 3, 4);
```

```
String
  t1 = new String("ahoj");
  t2 = new String("ahoi");
  t3 = new String("AHOJ");

  t1.compareTo(t2);                  // 1 >
  t2.compareTo(t1);                  // -1 <
  t1.compareToIgnoreCase(t3);// 0
  t1.equals(t3);                     // false
  t1.equalsIgnoreCase(t3);      // true
```

"" != null

Prázdny reťazec nie je neinicializovaný reťazec

Kvíz - 1

```
static String s1;
static String s2 = null;
static String s3 = "";
```

```
System.out.println(s1 == s2);     true
System.out.println(s1 == s3);     false
                                                    (NPE)
System.out.println(s1.length()); java.lang.NullPointerException
System.out.println(s2.length()); java.lang.NullPointerException
System.out.println(s3.length()); 0
```

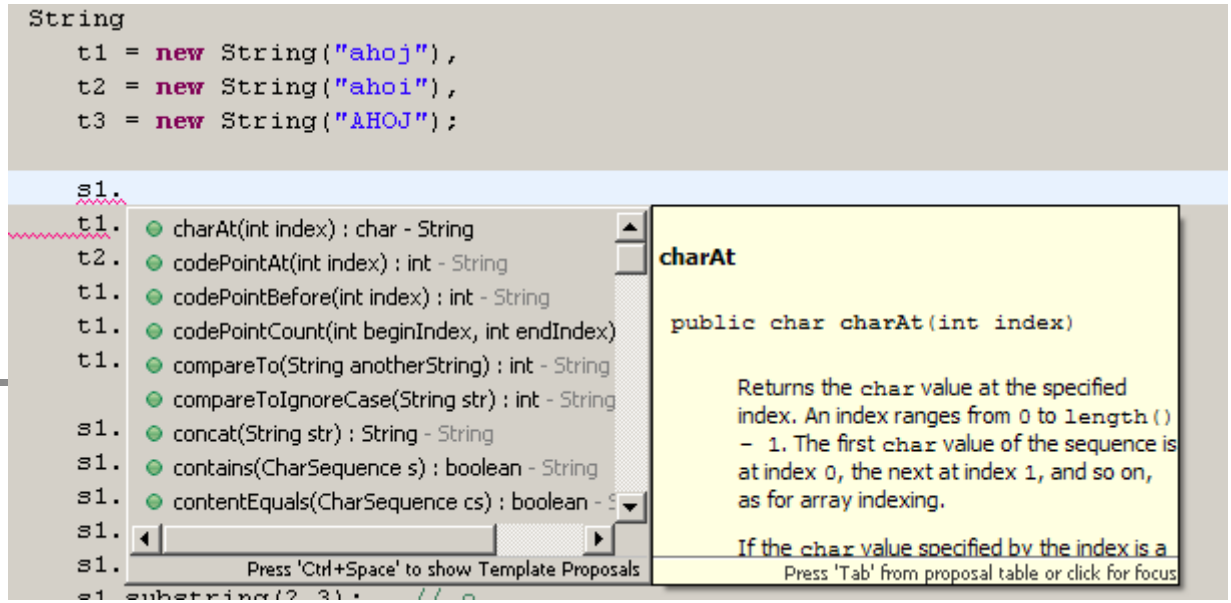
== porovnáva pointre
.equals, .compareTo porovnáva reťazce

Kvíz - 2

s.equals("java") môže padnúť, ak s=null
"java".equals(s) NIKDY NEPADNE na NPE

```
static String s4 = "java";
static String s5 = new String("java");
static String s6 = "ja"+"va";
static String s7 = "ja";

System.out.println(s4 == s5);          false
System.out.println(s4 == s6);          true
s7 += "va";
System.out.println(s4 == s7);          false

System.out.println(s4.equals(s5));     true
System.out.println(s4.equals(s6));     true
System.out.println(s4.equals(s7));     true
```

Metódy



niet nad kontextový help...

```
s1.toLowerCase()        // ahoj
s1.toUpperCase()        // AHOJ
s1 + s2                 // ahojahoi
s1.concat(s2)           // ahojahoi
s1.replace('h', 'H')    // aHoj
s1.substring(2)         // oj
s1.substring(2,3)       // o
s1.charAt(2)            // o
s1.indexOf('o')         // 2
```

- používajte kontextový help
- naučte sa používať JDK API, search
- prestavu o existujúcich metódach

```
s1.trim().toUpperCase().substring(2).indexOf('O');
```

Súbor: Retazce.java

Reťazce – metódy

```
String s = "male a VELKE";
int  i = s.indexOf('a');                  // prvé 'a'
int i = s.indexOf('a', i + 1);            // ďalšie 'a'
i = s.lastIndexOf('a');                   // posledné 'a'
i = s.lastIndexOf('a', i - 1);            // predposledné 'a'
i = s.lastIndexOf("VEL");                 // podreťazec

String a[] = {"Peter", "Marek" };
String s = String.format("Ahoj %s, tu je %s", a);

Character.isDigit('1')                            // true
char b= '1'; if (b >= '0' && b <= '9') ...if (b >= 48 && b <= 58) ...
Character.isLetter('A')                           // true
Character.isLowerCase('b')                        // true

Character.digit('5', 10)                          // 5
Character.digit('F', 16)                          // 15
```

Súbor: Retazce.java

Polia jednorozmerné

- *typ*[] – je typ 1-rozmerného poľa
- **new** *typ*[size] – vytvorenie/alokácia
- pole.length – dĺžka poľa
- pole[i] – indexovanie poľa
- polia majú VŽDY indexy 0..N-1

```
public class Jednoduche {

 public static void main(String[] args) {
 final int MAX = 20;                                  // konštanta – veľkosť poľa

 // int[] poleInt;                                    // definícia poľa
 // poleInt = new int[MAX];                           // vytvorenie poľa
 int[] poleInt = new int[MAX];                        // definícia poľa s vytvorením
   for (int i = 0;  i < poleInt.length;  i++) {       // i < MAX
     poleInt[i] = i + 1;                              // inicializácia poľa
     System.out.print(poleInt[i] + "  ");
   } // for
  }  // main
} // class
```

typ elementu poľa

Súbor: Jednoduche.java

Napriek tomu, že následujúce rady sú kus za okrajom samozrejmosti, dovoľujem si ich uviesť (pre vaše dobro).

Dobré rady

- ak je len trochu možné, vytvorte/alokujte pole ZÁROVEŇ s jeho deklaráciou. Predpokladá to, že v mieste deklarácie poľa poznáte jeho veľkosť. Ušetrite si chyby, keď píšete do nevytvoreného poľa.
  inak: deklarácia ***int[] prvocisla*** žiadne pole nevytvorí. Jediné, čo urobí, že existuje null-referencia/smerník ***prvocisla***, ktorý by chcel ukazovať na pole.

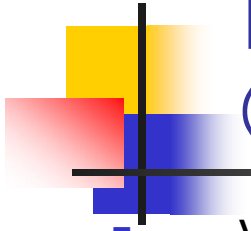
- ak to je možné, inicializujte pole hneď, ako ho deklarujete. Bonusom je, že sa vám aj automaticky vytvorí, príklad
  int[] prvocisla = { 2, 3, 5, 7, 11, 13, 19 }; // má dĺžku 7, indexy 0..6

- pole dĺžky N nikdy nebude mať iné indexy ako 0..(N-1).
  ešte inak: pole[pole.length] vždy skončí s ArrayIndexOutOfBoundsException.

- najprirodzenejší cyklus pre pole je for(int i=0; i<pole.length; i++) …
  ešte inak: pascalistický zlozvyk for(int i=1; i<=pole.length; i++) je kandidátom na ArrayIndexOutOfBoundsException

Polia v Java vs. C++

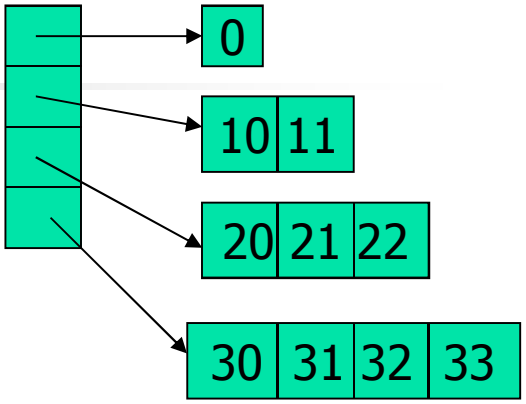
(porovnanie pre C++ programátora)



- v C++ po deklarácii poľa `int P[100]` sa vám pole automaticky naalokuje
- v Jave toto `int[] P` je deklarácia a toto `P = new int[100]` alokácia
- v Jave aj C++ pole inicializujete podobne `int P[] = { 1,2,3,4 }`, `int[] P = { 1,2,3,4 }`
- v Jave sa vytvorené pole inicializuje hodnotami
  - 0 pre číselné typy,
  - '\u0000' pre char,
  - false pre boolean,
  - null iné
- v Jave sa nedá indexovať za hranice poľa, kontroluje hranice
- pole je referenčný typ v Jave aj C++
- pole1 = pole2; je priradením referencií nie kopírovanie polí
- ak potrebujeme kopírovať poľe:
  - C++: `void*`**`memcpy`**`(void *dest, void *source, size_t num)`
  - Java: `System.`**`arraycopy`**`(`*`src,srcPos,dst,dstPos,count`*`)`

- *typ*[][] – je typ 2-rozmerného poľa,
- pole[i,j] píšeme ako pole[i][j],
- new *typ*[M][N] vytvorí pole MxN

Polia dvojrozmerné



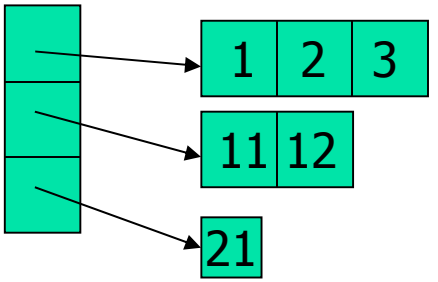
- java nemá klasické viacrozmerné polia (matice),
- viacrozmerné polia môžu byť „zubaté" (jagged)

```
public class Dvojite {

  public static void main(String[] args) {
    int[][] a = new int[4][];                    // hlavné pole
    for (int i = 0;  i < a.length;  i++) {
      a[i] = new int[i + 1];                     // podpole

      for (int j = 0;  j < a[i].length;  j++) {
        a[i][j] = i * 10 + j;
        System.out.print(a[i][j] + "  ");
      } // for
      System.out.println();
    } // for
  } // main
} // class
```

```
0
10  11
20  21  22
30  31  32  33
```

Súbor: Dvojite.java

Inicializácia poľa



- inicializácia dvojrozmerného poľa

```
int[][] a = { { 1, 2, 3},          a[0][0] = 1; a[0][1] = 2; a[0][2] = 3; a[0].length == 3
              {11, 12},            a[1][0] = 11; a[1][1] = 12; a[1].length == 2
              { 21} };             a[2][0] = 21; a[2].length == 1
                                   a.length == 3
```

- vytvorenie 3-rozmernej matice matice 5x5x5

```
int[][][] d = new int [5][5][5];          // definícia s vytvorením
```

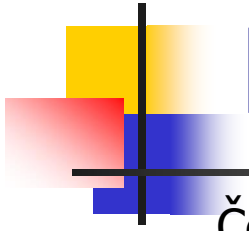
- vytvorenie 2-rozmernej matice "matice" 5x5, ktorej prvky sa vytvoria neskôr

```
int [][][] e = new int[5][5][];
e[0][1] = new int [8];                    ... ok
```

- nesprávne vytvorenie

```
int[][][] f = new int[5][][5]             .... Chyba - nemôžem vytvoriť "maticu", ktorej
f[0]?[1] = new int[8]                     .... druhý rozmer nepoznám ale tretí poznám
```

Upozornenie:
tento slajd nie je v Jave



Kvíz pre C++ programátora

Čo spraví nasledujúci program

```
#include <stdio.h>
void main() {
  int a[][] = { { 1,2,3 }, { 11, 12 }, { 21 }}; }
> gcc test.c
test.c:4: error: array type has incomplete element type
```

a čo tento:

```
#include <stdio.h>
void main() {
  int a[][3] = { { 1,2,3 }, { 11, 12 }, { 21 }};
  printf("%d\n",sizeof(a[0])/sizeof(int));
  printf("%d\n",sizeof(a[1])/sizeof(int));
  printf("%d\n",sizeof(a[2])/sizeof(int));
> gcc test.c
> a.out
3
3
3
```

Poučenie:
medzi poliami v C++ a Jave
sú subtilné rozdiely

pascalistu poznáš podľa ArrayIndexOutOfBoundsException: N, kde N je dĺžka jeho poľa

Polia a cykly

Najčastejšie chyby s poľami:
- pole nie je vytvorené len deklarované (chýba new)
- index mimo rozsahu – a[100] s indexom 100 (neostrá podmienka v cykle)

```
final static int MAX = 100;
public static void main(String[] args) {
    char[] poleChar = new char[MAX];
```

- `for (int i = 0;  i < poleChar.length;  i++) { . . . }` // for-to-do
- `for (int i = MAX-1;  i >= 0;  i--) { . . . }` // for-downto-do

-
```
int j=MAX;
while (j-- > 0) { . . . }
```
// while

-
```
int i=0;
do { . . .
} while (++i < MAX);
```
// do-while

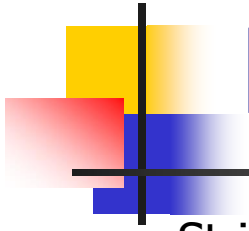
- `for (char ch:poleChar)  System.out.println(ch);` // for-each

for (*typPrvkuPola* prvokPola:pole) *tu vidím prvokPola, neviem jeho index*
// prechádza postupne prvky poľa bez toho, aby sme vedeli ich index

Súbor: PoleCyklus.java

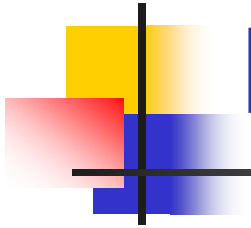
null nie je:

- **new String[0]**
- **new String[]{}**



Kvíz - nájdite rozdiely

String [] p1;	p1.length == ?	NullPointerException
String [] p2 = null;	p2.length == ?	NullPointerException
String [] p3 = new String[0];	p3.length == ?	0
String [] p4 = new String[]{};	p4.length == ?	0
String [] p5 = new String[]{""};	p5.length == ?	1
	p5[0].length() == ?	0
String [] p6 = new String[]{null};	p6.length == ?	1
	p6[0].length() == ?	NullPointerException



Bubble sort

Buble sort je bezpochyby najobľúbenejší triediaci algoritmus medzi študentami.
•ale aj ten možno pokaziť, viď Chyba1, Chyba2, Chyba3, ...

```
public class BubbleSort {

                                                                    1
  public static void main(String[] args) {                         2
     int[] a = {4,5,2,12,1,2,3};                                    2
     for (int i = 0; i < a.length ; i++) {        // cyklus for-to-do    3
        for (int j = a.length-1; j>i ; j--) {     // cyklus for-downto-do 4
           if (a[j-1] > a[j]) {                                     5
              int temp = a[j];                                      12
              a[j] = a[j-1];
              a[j-1] = temp;
           } // if
        } // for
     } // for
     for (int elem:a)                             // cyklus for-each-element
           System.out.println(elem);
  }
}
```

Súbor: BubleSort.java

Náhodné číslo náhodné pole

```
dlzka pola = 9
a[0] = 39
a[1] = 203
a[2] = 402
a[3] = 24
a[4] = 65
a[5] = 144
a[6] = 95
a[7] = 490
a[8] = 108
```

```
import java.util.*; // používame triedu Random

public class NahodnePole {

  public static void main(String[] args) {

    Random rand = new Random();          // inic.generátor náhod.čísiel

    int[] a = new int[rand.nextInt(20)];     // náhodná dĺžka z [0..20)
    System.out.println("dlzka pola = " + a.length);

    for(int i = 0; i < a.length; i++) {
      a[i] = rand.nextInt(500);              // plní náhodnými číslami [0..500)
      System.out.println("a[" + i + "] = " + a[i]);
    }
  }
}
```

Súbor: NahodnePole.java

Pole ako (vstupný) argument

```
public class Sort {
  public static void bubleSortuj(int[] a) {
      for (int i = 0; i < a.length ; i++) {          // cyklus for-to-do
        for (int j = a.length-1; j>i ; j--) {        // cyklus for-downto-do
          if (a[j-1] > a[j]) {
            int temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
           } // if
        } // for
      } // for
  }
  public static void vypis(int[] a) {              // int[7] a – je chyba, lebo
    for (int i:a) System.out.println(i + ",");     // int[7] nie je typ pol'a
    System.out.println();
  }
  public static void main(String[] args) {
      int[] poleInt = {4,5,2,12,1,2,3};
      bubleSortuj(poleInt);
      vypis(poleInt);
  }
}
```

v Jave nenájdete analógiu chaosu
* a & parametrov z C++

Súbor: Sort.java

Pole ako výstupná hodnota

```
public static int[] generuj(int velkost) {      // generuj pole danej veľkosti

   int[] retValue = new int[velkost];           // deklaruj a vytvor lokálne pole
   Random rand = new Random();
   for(int i=0; i<velkost; i++)
       retValue[i] = rand.nextInt(100);         // naplň lokálne pole
    return retValue;                            // vráť referenciu na pole ako
 }                                              // výsledok funkcie

public static void main(String[] args) {
      int[] poleInt = generuj(20);              // pri volaní funkcie si definujeme
                                                // premennu, do ktorej uložíme
                                                // referenciu na vytvorené pole
```

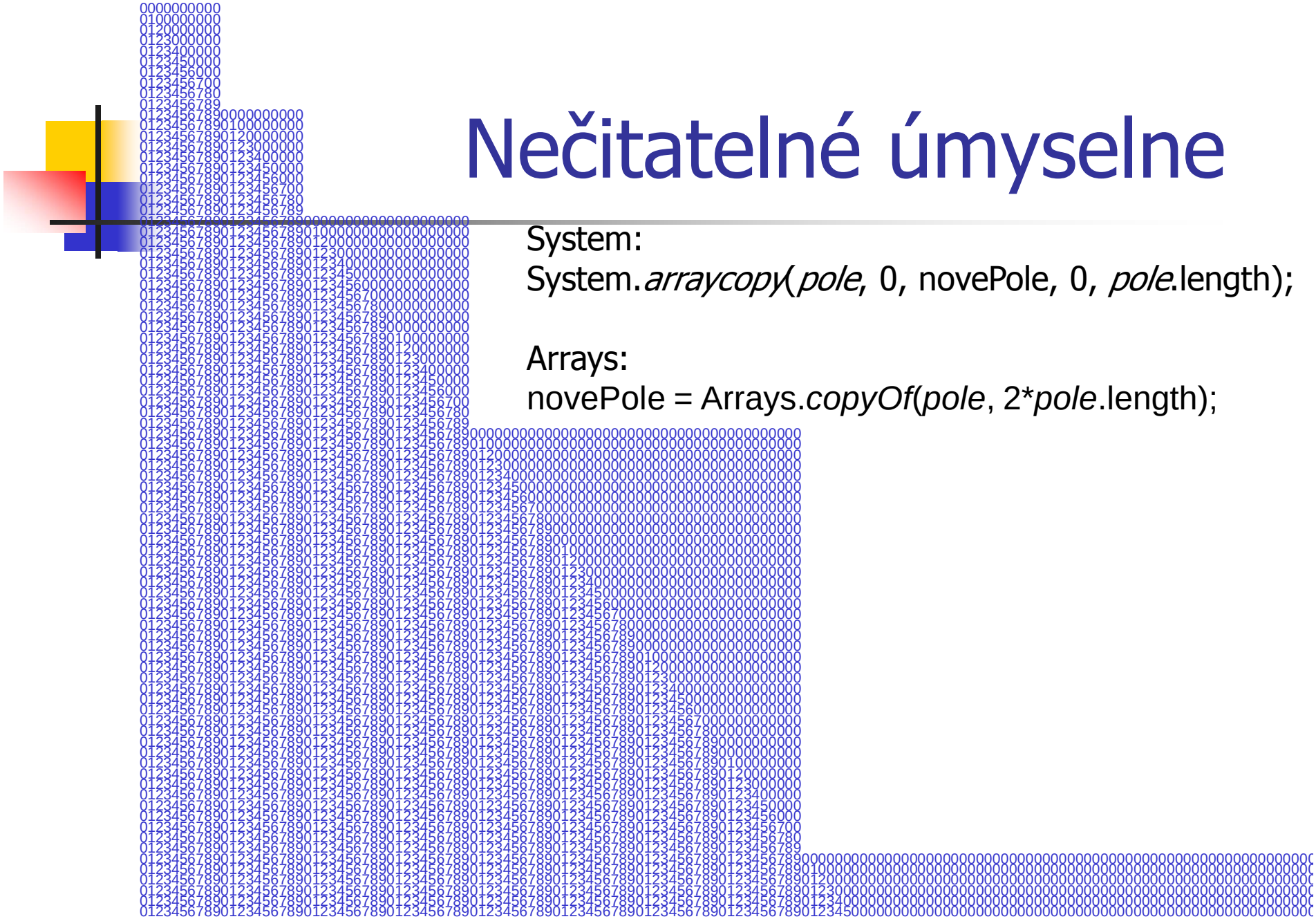
Súbor: Sort.java

Ak postupne pridávame prvky do poľa, ktorého rozmery pri vytvorení sme neodhadli dobre, časom potrebujeme zväčšiť pole – preventívne na 2násobok

Realokácia poľa

```
public class Reallocate {
  static int[] pole = new int[10];        // staticke pole inicializovane na dlzku 10
  static int pocet = 0;                   // pocet zapisanych prvkov v poli

  static void pridajDoPola(int x) {
     if (!(pocet < pole.length)) {        // ak uz je pole plne
       int[] novePole = new int[2*pole.length];    // realouj pole, t.j.
       for(int i=0; i<pole.length; i++)    // vytvor pole dvojnasobnej velkosti
          novePole[i] = pole[i];           // prekopiruj do neho hodnoty stareho pola
       pole = novePole;                    // zahod stare pole
     }
     pole[pocet++] = x;                    // pridaj prvok
  }
  public static void main(String[] args) {
     for(int i=0; i<100; i++) {
        pridajDoPola(i%10);
        for(int elem:pole) System.out.print(elem);
        System.out.println();
     }
  }
```

Súbor: Reallocate.java



Nečitatelné úmyselne

System:
System.*arraycopy*(*pole*, 0, novePole, 0, *pole*.length);

Arrays:
novePole = Arrays.*copyOf*(*pole*, 2**pole*.length);

Súbor: Reallocate1.java, Reallocate2.java

Triedy java.util.Arrays, java.lang.System

užitočné statické metódy na prácu s poľami

```
import java.util.Arrays;                          // používam triedu z balíka java.utils

int[] a = new int[10];                            // pole primitívneho typu int
Arrays.fill(a, -1);                               // vyplň pole nulami, memset
System.arraycopy(a, 11, b, 3, 7);                 // kópia od a[11]->b[3] 7 prvkov
                                                  // memcpy

String[] s = {"janko","marienka","jozko","mracik"};
String[] s_copy = new String[4];
System.arraycopy(s, 0, s_copy, 0, s.length);      // kópia poľa
Arrays.sort(s);                                   // triedenie poľa
for(String elem:s) System.out.print(elem+",");    // janko,jozko,marienka,mracik,
                                        // binárne vyhľadávanie v utriedenom poli

System.out.println(Arrays.binarySearch(s, "sandokan"));   // nenachádza sa: -5
System.out.println(Arrays.binarySearch(s, "marienka"));   // nachádza sa: 2

Arrays.equals(s, s_copy);                         // porovnanie polí- false
```

Súbor:Array.java

Predávanie argumentov

Základné typy sa prenášajú hodnotou, ostatné (polia, objekty, ...) referenciou

```
public class Test1 {

  static int zmena(int i) {
    i++; return i;
  }

  public static void main(String[] args) {
    int j, k = 4;
    j = zmena(k);
    System.out.println(
     "k=" + k + ", j=" + j);
  }
}
```

k=4, j=5

```
public class Test2 {

  static int[] zmena(int[] x) {
    int[] c = x;
    x[0] = 99;
    return c;
  }

  public static void main(String[] args) {
    int[] a = {0,1,2,3};
    int[] b = zmena(a);
    System.out.println("a="+a[0]+
                       "b="+b[0]);
  }
}
```

a=99, b=99

Súbor: Test1.java, Test2.java

Sú collections lepšie ?

(a môžeme ich už používať?)

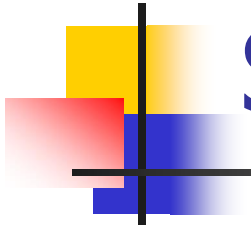
```
public static void bubleSortuj(int[] a) {
   for (int i = 0; i < a.length ; i++) {
     for (int j = a.length-1; j>i ; j--) {
       if (a[j-1] > a[j]) {
          int temp = a[j];
          a[j] = a[j-1];
          a[j-1] = temp;
        }
     }
   }
 }
```

elapsed time:30s.

```
public static void
       bubleSortuj(ArrayList<Integer> a) {
    for (int i = 0; i < a.size() ; i++) {
      for (int j = a.size()-1; j>i ; j--) {
        if (a.get(j-1) > a.get(j)) {
           Integer temp = a.get(j);
           a.set(j,a.get(j-1));
           a.set(j-1,temp);
        }
      }
    }
  }
```

elapsed time:112 s.

Súbor: SortObama1.java, SortObama2.java



Statické metódy

doposiaľ sme až na pár skrytých prípadov používali len statické metódy, premenné a konštanty.

Statické metódy:

- predstavujú klasické procedúry/funkcie ako ich poznáme z C++,
- existujú automaticky, ak použijeme (importujeme) danú triedu,
- existujú bez toho, aby sme vytvorili objekt danej triedy,
- referencujú sa *menom*, napr. *vypis(pole),* alebo *menom triedy.meno metódy*, konštanty, napr. *Math.cos(fi), Math.PI, Systém.out.println(5),*
- ak aj metóda nemá argumenty, prázdne zátvorky sa do jej definície a do volania aj tak píšu (à la C++), napr. *System.out.println**();***

- syntax deklarácie statickej metódy je
  [**public**] **static** *typ meno***(***argumenty***) {** telo **}**

- ak ide o procedúru (nie funkciu), výstupný typ je **void**

Statické premenné a bloky

statický inicializačný blok

```
public class Prvocisla {

  public static final int MAX = 1000;          // v statickom inic.bloku vidíme len
  public static       int cisla[] = new int[MAX];// statické premenné triedy
  static {                                  // vykoná sa raz, po zavedení triedy do pamäte
    int pocet = 2;                                    public static void main(String[] args) {
    cisla[0] = 1;                                      for (int i=1;i<Prvocisla.cisla.length; i++)
    cisla[1] = 2;                                          System.out.print(cisla[i] + "  ");
                                                      }
  dalsi:                                            }
    for (int i = 3;  pocet < MAX;  i += 2) {
      for (int j = 2;  j < pocet;  j++)
        if (i % cisla[j] == 0)
          continue dalsi;
      cisla[pocet] = i;
      pocet++;
    }
  }
```

Súbor: Prvocisla.java

Statické metódy vidia len statické premenné a môžu volať len statické metódy (bez vytvorenia objektu).

Rekurzia

```
public class Fibonacci {

  public static void main(String[] args) {
     int N = Integer.parseInt(args[0]);
     while (N-- > 0)
       System.out.println(fib(N));
  }
  public static long fib(int n) {
     //return (n < 2)?1:fib(n-1)+fib(n-2);  // fajnšmekerská verzia
     if (n < 2)
       return 1;
     else
       return fib(n-1)+fib(n-2);
  }
}
```

miesto na výstupný typ metódy
void je „prázdny" typ
t.j.nevracia výstup (procedúra)

výstupný typ metódy

výstupná hodnota metódy

kým sa nedozvieme viac, všetky metódy sú static
inak nerozumieme chybe:
Cannot make a static reference to the non-static method fib(int) from the type Fibonacci

Súbor Fibonacci.java

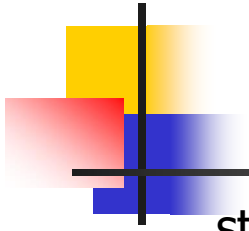
Globálne a forward deklarácie

(neexistujú)

- globálne premenné neexistujú
- oblasť viditeľnosti premennej/metódy je aj pred jej deklaráciou (nepotrebujeme forward deklarácie)

```
public class Metody {
  static void tlac1() {
    System.out.println(i);
  }
  public static void main(String[] args) {
    tlac1();
    tlac2();
  }
  static int i = 5;
  static void tlac2() {
    System.out.println(i);
  }
}
```

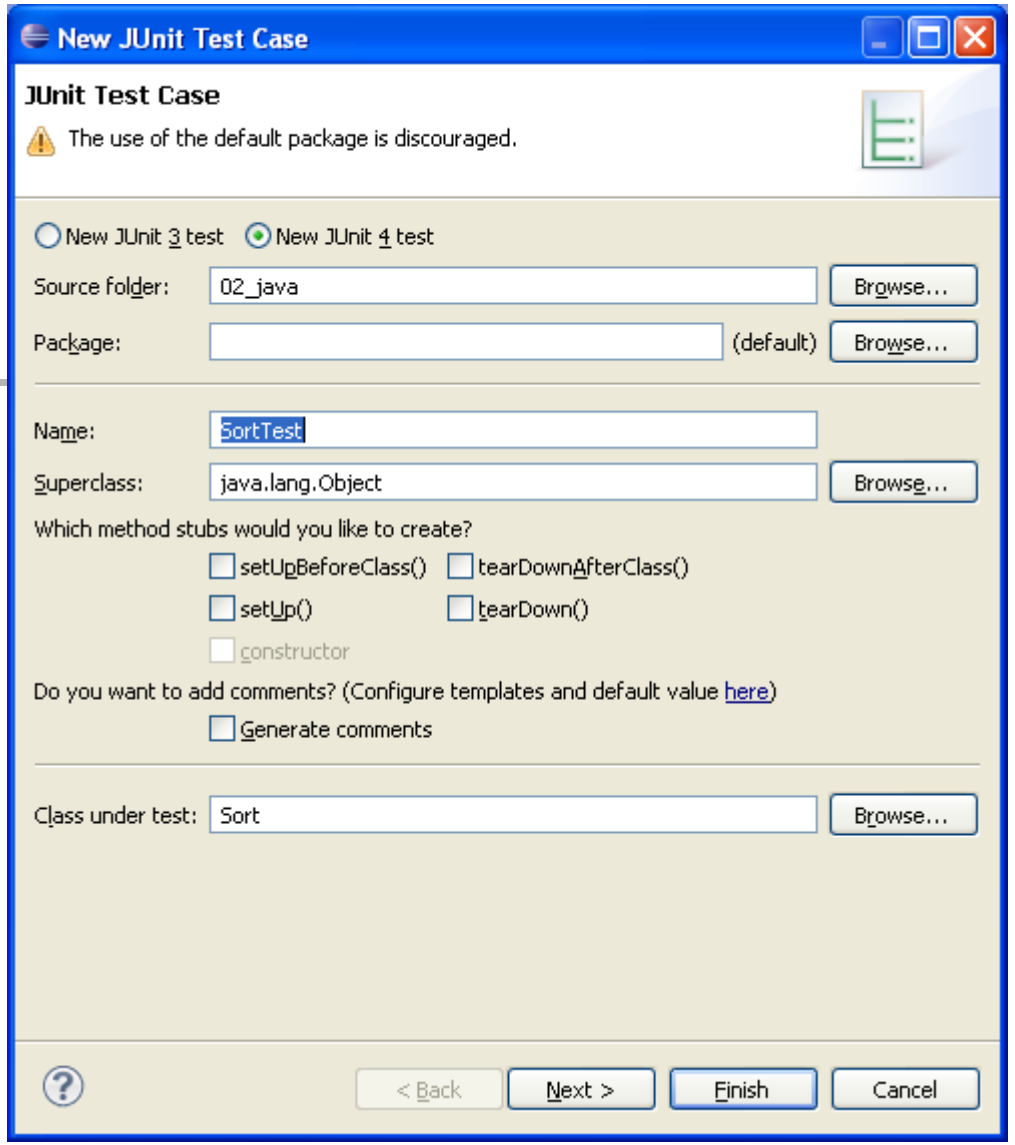
Oblasť viditeľnosti premenných

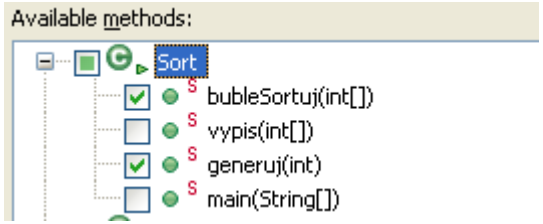


```
static void tlac() {
   int i = 6; int q;
   System.out.println(i);
   {                                    // vnorený blok
//     int i = 7;                       // chyba - dvojnásobná deklarácia
//     long i = 7;                      // chyba - dvojnásobná deklarácia
     int j = 8;
     System.out.println(j);
   }                                    // koniec vnoreného bloku
//     System.out.println(j);           // chyba - j nie je viditeľná
  }
static void tlac2() {
  int i = 6; int q;
  System.out.println(i);
//   for (int i = 1;  i < 5;  i++)      // chyba, i už je definovaná
    System.out.println(i);
}
```

Testovanie

- testovanie je minimálne rovnako náročné, ako programovanie
- Java poskytuje nástroj/podporu vo forme tzv. JUnit testov, ktoré si postupne predstavíme
- vytvorme prvý JUnit Test SortTest k triede Sort,
- budeme testovať metódy generuj a bubleSortuj





JUnit Test čarodejník vytvorí kostru testu, ktorú upravujeme

Prvý JUnit Test

```
import static org.junit.Assert.*;
import org.junit.Test;
public class SortTest {
```

testujeme, či generuj vytvorí pole správnej veľkosti

```
@Test
public void testGeneruj() {
  int testPole[] = Sort.generuj(100);
  if (testPole == null)
    fail("ziadne pole");
  assertNotNull("ziadne pole",testPole);
  assertEquals("velkost pola",
        testPole.length, 100);
  assertTrue("velkost pola",
        testPole.length == 100);
}
```

```
@Test(timeout=10) // ms
public void testBubleSortuj() {
  int testPole[] = Sort.generuj(10000);
  Sort.bubleSortuj(testPole);
  for(int i=0; i+1<testPole.length; i++)
    if (testPole[i] >testPole[i+1])
      fail("neutriedene");
}
```

testujeme, či triedenie utriedi pole v danom časovom limite

Súbor SortTest.java

Čo ponúka JUnit Test

http://www.vogella.de/articles/JUnit/article.html#junit_intro

org.junit.Assert poskytuje metódy:

fail("tu to zlyhalo")
assertTrue(n>0)
assertsEquals("test n", n, 100)
assertsEquals("realny test", pi, 3.14,0.01)
assertNull("null referencia", pole)
assertNotNull("not null referencia", pole)
assertSame("rovnake", pole1, pole2)
assertNotSame("rozne", pole1, pole2)
assertTrue("podmienka", pole.length>0)

... a mnoho d'alších

@Anotácie:

@Test
@Before
@After
@Ignore
... a d'alšie

```
@Test(expected=IndexOutOfBoundsException.class) public
    void testBubleSortuj() {
    // toto nebude dobrý test, lebo
    ignoruje nesprávne indexovanie
```