



Ret'azce a polia

(a iné odlišnosti Java a C++)



Peter Borovanský
KAI, I-18

borovan 'at' ii.fmph.uniba.sk

<http://dai.fmph.uniba.sk/courses/JAVA/>

Zrnko múdrosti



Charles Babbage and Augusta Ada King, Countess of Lovelace

The designer of the analytical engine
and its programmer

Při dvou příležitostech jsem byl členy **parlamentu** dotázán:
"Řekněte, pane Babbage, když do toho vašeho stroje
zadáme špatné údaje, vypočítá správný výsledek?"

Nejsem schopen pochopit, jaký druh zmatení myšlenek
dokáže vyplodit takovou otázku...

Charles Babbage

Status quo

CV01

- 83 riešení, 47 plné 2 body, priemer 1.72 😊
- github obsahuje pekné riešenia, aj riešenia hodné komentovania
- zapojila sa aj umelá inteligencia – riešenie typu „*loop hell*“, resp. „*loop to hell*“

```
//riešenie od chat gpt pokus3 s jasnejšími inštrukciami
import java.util.Arrays;
public class Sedemciferne {
    public static int najvacsie7() {
        int out = -1;
        for (int a = 9; a >= 1; a--) {
            for (int b = 9; b >= 1; b--) {
                if (b == a) continue;
                for (int c = 9; c >= 1; c--) {
                    if (c == a || c == b) continue;
                    for (int d = 9; d >= 1; d--) {
                        if (d == a || d == b || d == c) continue;
                        for (int e = 9; e >= 1; e--) {
                            if (e == a || e == b || e == c || e == d) continue;
                            for (int f = 9; f >= 1; f--) {
                                if (f == a || f == b || f == c || f == d || f == e) continue;
                                for (int g = 9; g >= 1; g--) {
                                    if (g == a || g == b || g == c || g == d || g == e || g == f) continue;
                                    int num = a * 1000000 + b * 100000 + c * 10000 + d * 1000 + e * 100 + f * 10 + g;
                                    if (num % a == 0 && num % b == 0 && num % c == 0 && num % d == 0 && num % e == 0 && num % f == 0 && num % g == 0) {
                                        out = num;
                                        break;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
        if (out > 0) break;
    }
    if (out > 0) break;
}
if (out > 0) break;
}
if (out > 0) break;
}
if (out > 0) break;
}
if (out > 0) break;
}
return out;
}
```

A System for Detecting Software Similarity

MOSS

príbeh s prekvapivým happy end

- porovnávame riešenia aj s riešeniami LS 2021/2022
- jednoduché kamufláže kódu nemá zmysel robiť
- často vás prezradia details, resp. bizarnosti

```

public class Sedemciferne {
    public static int najvacsie7(){

// vo for cykle prechadzame od najvacsieho sedemciferneho po najmensie sedemciferne
    for (int i = 10000000; i >= 1000000; --i){
        if (rozneCifry(i)) return i;
    }

    return 0;
}

// pre kazde cislo n (postupne prechadzame vsetkymi sedemcifernymi) kontrolujeme, ci splna podmienky
public static boolean rozneCifry(int n){
    int temp = n;
    // pole reprezentuje cifry od jedna po devat
    int pole[] = {0,0,0,0,0,0,0,0};
    int i;
    int x =10;
    // v tomto cykle si napocitavame, kolkokrat sa ktora cifra nachadza v danom cisle n
    // zaroven zistujeme, ci tam nahodou nemame ziadnu nulu - vtedy vrati false
    for (i=0;i<7;i++){
        if (temp % x == 0) return false;
        pole[(temp % x) -1 ]++;
        temp/=10;
    }
    // ak sa niektera cifra nachadza viac nez 2-krat, vratime false
    if (max(pole) > 1) return false;

    // v cykle prejdeme vsetky cifry a kontrolujeme, ci nie su delitelmi nasho vstupneho cisla n
    // ak je nejake delitelom, vratim nulu
    for (i=0; i <9; i++){
        if (pole[i] == 1){
            if (n % (i+1) != 0) return false;
        }
    }
    return true;
}

// vrati navyssiu hodnotu zo vstupneho pola
public static int max(int pole[]){
    int x = pole[0];
    for (int i = 1; i < 9; i++)
        if (pole[i]>x) x = pole[i];
    return x;
}

public static void main(String[] args) {
    System.out.println(najvacsie7());
}
}

```

```

public class Sedemciferne {
    public static int najvacsie7(){

        for (int i = 10000000; i >= 1000000; --i){
            if (rozneCifry(i)) return i;
        }

        return 0;
    }

    public static boolean rozneCifry(int n){
        int temp = n;
        int pole[] = {0,0,0,0,0,0,0,0};
        int i;
        int x =10;
        for (i=0;i<7;i++){
            if (temp % x == 0) return false;
            pole[(temp % x) -1 ]++;
            temp/=10;
        }
        if (max(pole) > 1) return false;

        for (i=0; i <9; i++){
            if (pole[i] == 1){
                if (n % (i+1) != 0) return false;
            }
        }
        return true;
    }

    public static int max(int pole[]){
        int x = pole[0];
        for (int i = 1; i < 9; i++)
            if (pole[i]>x) x = pole[i];
        return x;
    }

    public static void main(String[] args) {
        System.out.println(najvacsie7());
    }
}

```

Donald E. Knuth



(Pozor na chyby v tom kódu; já jsem pouze dokázal, že funguje, nezkoušel jsem ho ... Donald Knuth :)

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



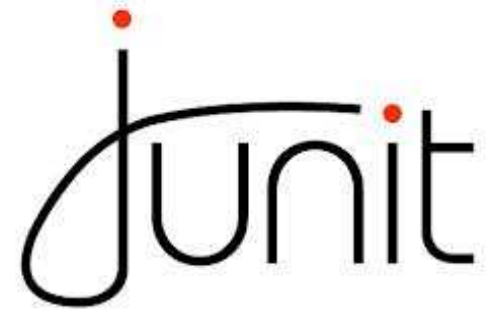
π

TEX

⑩ Release October 2016 is 3.14159265.
⑩ upon his death, the version of TeX shall be frozen at

Testy

(sú iné ako ste čakali)



- používame techniku štandardných java **junit4 testov**, viac v prednáške
- všeobecne: **dodržiavanie predpísaných mien súborov, tried a metód**
- keďže rekord submitov je > 20 , neprezeráme všetky, ale len posledný, preto d **posledného submit dajte všetky dobré riešenia celej zostavy !!!**
- do testov posielajte **.zip** obsahujúci ***.java bez package**
- na cvičení **bude** súťaž o najlepšieho testera (v písaní junit testov)
- testy **čiastočne zverejňujeme** (okrem autorského riešenia), zdrojáky čítame!
- ak bezradne/dlho ladím a test failuje, v testovanej metóde si **vypíšem vstupy**
- ak to nie je uvedené v zadaní, môžete sa **pýtať na rozsah vstupov**
- **15 s. je timeout** pre každého, ak to nie je inak junit testom upravené





Konverzie, reťazce, polia

dnes bude:

- **operátory** &&, &, ||, | , preťažovanie operátorov
- **konverzie** základných číselných typov
- **reťazce** (trieda String) a práca s nimi,
 - triedy StringBuilder, StringBuffer
 - **regulárne výrazy** - regexp
- **polia** (pohľad C++ programátora)
- **testovanie** (prvý JUnit test – asi na cvičení)

cvičenia:

- programy s poľami, testovanie a ladenie (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/>)



Logické operácie komutatívne?

(vyhodnocovanie zľava doprava - "short-circuit")

- je `a&&b` je to isté ako `b&&a`, resp. `a||b` je to isté ako `b||a` ?
- **v algebre áno, v programe nie:**

```
public static boolean loop() { for(;;); } // nekonečný cyklus  
&&
```

```
false && (7/0 > 0)           // false  
//(7/0 > 0) && false        // div by zero
```

```
||  
true || (7/0 > 0)           // true  
//(7/0 > 0) || true        // div by zero
```

```
&&  
false && loop()             // false  
//loop() && false          // zacyklí sa
```

```
||  
true || loop()             // true  
loop() || true             // zacyklí sa
```


Skutočné logické operácie

(nebolo v C++)

- && , || sú "**skrátеным**" súčinom (konjunkciou), súčtom (disjunkciou), vyhodnocujú sa zľava doprava, a len kým treba...
- & , | sú **plnohodnotným** súčinom, súčtom vyhodnocujú oba argumenty

&

```
//false & (7/0 > 0)           // div by zero
//(7/0 > 0) & false           // div by zero
```

|

```
//true | (7/0 > 0)           // div by zero
//(7/0 > 0) | true           // div by zero
```

&

```
//false & loop()             // zacyklí sa
//loop() & false             // zacyklí sa
```

|

```
//true | loop()             // zacyklí sa
//loop() || true            // zacyklí sa
```

Skrátený súčet, súčin

```
int i, j, k;  
i = 1; j = 2; k = 3;  
if (i == 1 || ++j == 2) k = 4;  
System.out.println("i = " + i + ", j = " + j + ", k = " + k);
```

toto sa nevyhodnotí, lebo $i == 1$ a $true ||$ hocičo je $true...$

$i = 1, j = 2, k = 4$

```
i = 1; j = 2; k = 3;  
if (i == 1 & ++j == 2) k = 4;  
System.out.println("i = " + i + ", j = " + j + ", k = " + k);
```

teraz sa to vyhodnotí

$i = 1, j = 3, k = 4$

```
i = 1; j = 2; k = 3;  
if (i == 2 && ++j == 3) k = 4;  
System.out.println("i = " + i + ", j = " + j + ", k = " + k);
```

toto sa nevyhodnotí, lebo $i != 2$

$i = 1, j = 2, k = 3$

```
i = 1; j = 2; k = 3;  
if (i == 2 & ++j == 3) k = 4;  
System.out.println("i = " + i + ", j = " + j + ", k = " + k);
```

teraz sa to vyhodnotí, aj keď $i != 2$

$i = 1, j = 3, k = 3$

Pret'azovanie operátorov

- v Java existujú pret'ažené operátory, napr +, *, ...
 - + :: int + int -> int
 - + :: float + float -> float
- ale aj &, |:
 - | :: int | int -> int
 - | :: boolean | boolean -> boolean
 - & :: int & int -> int
 - & :: boolean & boolean -> boolean
- v Java používateľ nemôže definovať pret'ažené operátory... ☺
- ale môže napísať

```
int a = 7, b = 0;
```

```
...
```

```
a == 0 | b == 0
```

```
(a | b) == 0
```

```
// true, lebo b == 0
```

```
// false, lebo a | b == 7
```

```
// bitový súčet dvoch intov
```



JAVA	C
■ -1	■ -1
■ !int	■ 43
■ int int	■ 1
■ !int	■ 1

A screenshot of a "jshell - Shortcut" window. The background is black, and the text is white. It shows four commands being entered at the prompt "jshell>".

```
jshell> ~0x2B  
$5 ==> -44
```



```
jshell> 0x2b  
$6 ==> 43
```



```
jshell> Integer.toBinaryString(-44)  
$7 ==> "111111111111111111111111111111111010100"
```



```
jshell> Integer.toBinaryString(43)  
$8 ==> "101011"
```



```
jshell> Integer.toBinaryString(43|-44)  
$9 ==> "1111111111111111111111111111111111111111"
```



```
jshell>
```

The window has standard OS controls (minimize, maximize, close) in the top right corner.

Zvrhlosti

(iné v C++ a Java)

Kód, ktorý vyvoláva
pochybnosť/nejednoznačnosť,
nie je (v praxi) dobrý kód !!!

```
#include <stdio.h>
void main(int argc, char *argv) {
    int a = 0;
    int b = (a++) + (a);
    printf("%d\n",b);
    int i = 0;
    i = i++;
    printf("%d\n",i);
}
```

```
RACDB 1 root@orcl1:~$ gcc x.c
```

```
RACDB 1 root@orcl1:~$ ./a.out
```

```
0
```

```
1
```

```
3 public class Zvrhlosti {
4     public static void main(String[] args) {
5         int a = 0;
6         int b = (a++) + (a);
7         System.out.println(b);
8         int i = 0;
9         i = i++;
10        System.out.println(i);
11    }
```

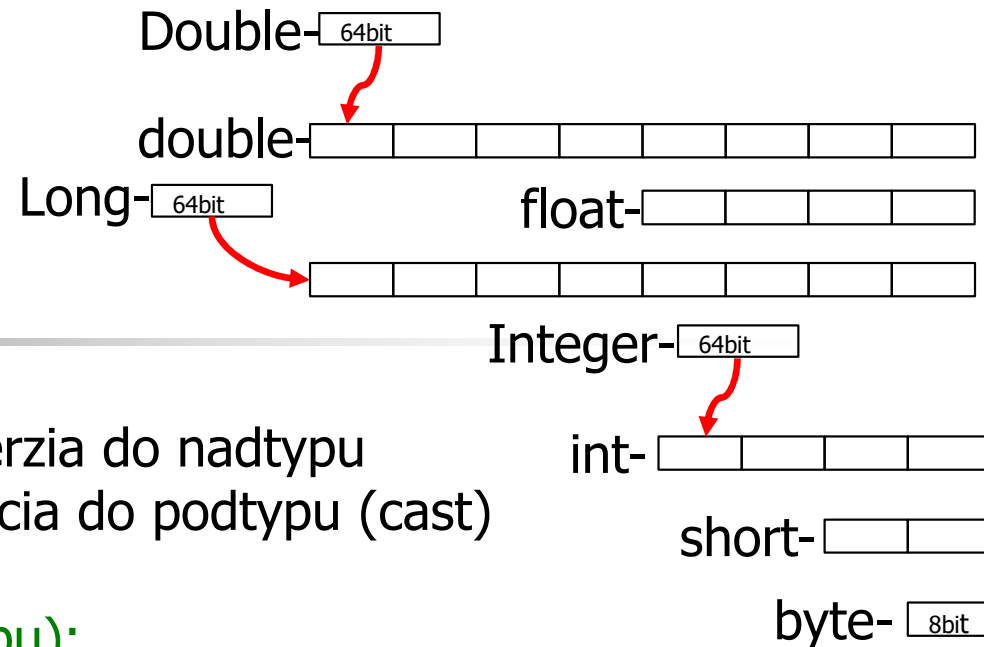
Správne miesto, kde sa (takto)
vyblbnúť, sú prémie v škole, nie
tímová programátorská práca...

Problems Javadoc Declaration Search Console Error Log History Git Staging

<terminated> Zvrhlosti [Java Application] C:\java8_0_91\bin\javaw.exe (28. 2. 2017, 21:49:39)

```
1
0
```

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  
b = (byte) 255;           // b = -1  
byte bb = 126;            // bb = 126  
bb += 3;                  // bb = -127  
bb = -126;                // bb = -126  
bb += -5;                 // bb = 125
```

```
jshell - Shortcut  
jshell> byte b = (byte)255;  
b ==> -1  
  
jshell> byte bb = 126; bb+=3;  
bb ==> 126  
$15 ==> -127  
  
jshell> byte bb = -126; bb+=-5;  
bb ==> -126  
$17 ==> 125
```

(300-256)

(255-256)

(126+3-256)

(-126-5+256)

Konverzie z/do String

String -> int

Integer.valueOf("123") → Integer
Integer.parseInt("123") → Int

Integer.parseInt("1010", 2) //10

String -> double

Double.valueOf("3.1415")
Double.parseDouble("3.1415")

String -> Boolean

Boolean.valueOf("true")
Boolean.parseBoolean("false")

int -> String

String.valueOf(123) → String
Integer.toString(123, 10) → String
"" + **123** je String tiež

Integer.toString(31) //11111
Integer.toString(15) // 17
Integer.toString(255) // ff

double -> String

String.valueOf(Math.PI)
Double.toString(Math.PI)

Boolean -> String

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



Ret'azce – metódy

```
String s1 = new String("Hello");
String s2 = "World";
-- zret'azenie, indexovanie, vyhľadavanie
String s3 = s1 + ' ' + s2 + "!";
s1.charAt(0) == 'H'
s1.indexOf("ll") == 2
s1.substring(1, 3).equals("el")
-- cyklus s indexom
for (int i = 0; i<s3.length(); i++) {
    s3.charAt(i)
}
-- cyklus bez indexu
char[] charArray = s3.toCharArray();
for (char ch : charArray) {
    ch
}
```

```
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
```



Ret'azce – porovnávanie

"" != null

Prázdny reťazec nie je neinicializovaný reťazec

== porovnáva pointre a nie obsahy reťazcov

väčšinou je to chyba vo vašom programe

ale

.equals(), .compareTo, .equalsIgnoreCase()

porovnávajú skutočné reťazce

s.equals("java") môže padnúť, ak s=null

"java".equals(s)

NIKDY NEPADNE na Null Pointer Exception - NPE

Reťazce (trieda String)

nikdy neporovnávame reťazce `s1==s2`, ale `s1.equals(s2)`

```
String ja = "Ja";  
String[] p = new String[] {"Jana", "Anna", "Mama"};  
for(int i = 0; i < p.length; i++)  
    System.out.println(p[i]);
```

Jana\nAnna\nMama\n

```
String janaString = p[0];  
System.out.println(janaString == "Jana");  
System.out.println(janaString == "Ja" + "na");  
System.out.println(janaString == ja + "na");  
System.out.println(janaString.equals(ja + "na"));  
System.out.println((ja + "na").equals(janaString));
```

true

true

false

true

true

```
System.out.println( (janaString.charAt(0) == 'J') );  
for(int i = 0; i < janaString.length(); i++)  
    System.out.print(janaString.charAt(i));
```

true

Jana

```
char[] poleCharov = janaString.toCharArray();  
for(char ch : poleCharov) System.out.print(ch);
```

Jana
Súbor: Retazce.java



Polia

(aby ste sa nenadreli, používajte metódy `java.util.Arrays`)

```
import java.util.Arrays;           // veľmi šikovná trieda na prácu s poľami
```

```
String[] p = new String[] {"Jana", "Anna", "Mama"};  
System.out.println(Arrays.binarySearch(p, "Jana"));
```

-3 - nenašiel

```
Arrays.sort(p);
```

```
for(String s : p)
```

```
    System.out.println(s);
```

Anna\nJana\nMama

```
System.out.println(Arrays.binarySearch(p, "Jana"));
```

1 – našiel na indexe 1

```
System.out.println(Arrays.binarySearch(p, "baa"));
```

-4 - nenašiel

```
System.out.println(Arrays.toString(p));
```

[Anna, Jana, Mama]

```
Arrays.fill(p, "cc");
```

[cc, cc, cc]

```
String[] r = Arrays.copyOf(p, 3);
```

[cc, cc, cc]

```
p[0] = "zmena";
```

[zmena, cc, cc]

```
System.out.println(r[0]);
```

cc

```
System.out.println(p[0]);
```

zmena

Súbor: [Retazce.java](#)



Kvíz - 1

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

```
System.out.println(s1 == s2);    true
```

```
System.out.println(s1 == s3);    false
```

```
System.out.println(s1.length()); java.lang.NullPointerException
```

```
System.out.println(s2.length()); java.lang.NullPointerException
```

```
System.out.println(s3.length()); 0
```

(NPE)



Kvíz - 2

```
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

```
s1.  
  substring(int beginIndex, int endIndex) String  
  substring(int beginIndex) String  
  matches(String regex) boolean  
  length() int  
  split(String regex) String[]  
  split(String regex, int limit) String[]  
  toCharArray() char[]  
  charAt(int index) char  
  contains(CharSequence s) boolean  
  startsWith(String prefix) boolean  
  startsWith(String prefix, int toffset) boolean  
  trim() String
```

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
```

-- zret'azenie volania metódu

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

- používajte kontextový help
- naučte sa používať [JDK API](#), search
- prestavu o existujúcich metódach

```
Character.get
```

```
  getNumericValue(int codePoint)
```



Ret'azce – 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  
for(char chr = 'a'; chr <= 'z'; chr++) { ... }
```

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



StringBuffer/StringBuilder

(o tragédii na quadterme 2016)

```
long start = System.currentTimeMillis();
String s = "";
for(int i = 0; i<1000000; i++) s += "a";
System.out.println("elapsed time:"+(System.currentTimeMillis()-start)/1000);
```

elapsed time: 698 s.

- pri prireťazení hoc aj jedného znaku sa naalokuje nový reťazec, prekopíruje, ...
- preto dostaneme kvadratickú zložitosť s kvadratickým garbage ☹
- ak to v rámci testu na staručkom L.I.S.T.e odpálilo 60 študentov pri vrcholiacom quadterme, katastrofa bola jasná ☹ ☹ ☹
- no a chyba je v príklade, teste, alebo v programátoroch ? (nepříjemná hádka, 2016)
- ako je to v Pythone ?

```
long start = System.currentTimeMillis();
StringBuffer ss = new StringBuffer();
for(int i = 0; i<1000000; i++) ss.append("a");
String s = ss.toString();
System.out.println("elapsed time: " + (System.currentTimeMillis()-start)/1000);
```

elapsed time: 0 s. !!!



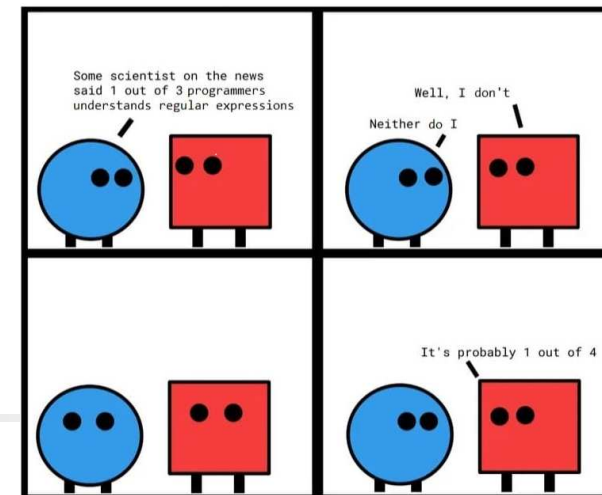
String in Java

(StringBuilder, StringBuffer)

- String v Jave nie je `char*` v C++
- String je **nemenný** (immutable) – ak chcete modifikovať reťazec, musíte ho vytvoriť (naalokovať) znova, to stojí čas aj pamäť
- StringBuffer/StringBuilder sú modifikovateľné reprezentácie reťazcov vytvorené na heape
napr. `StringBuffer.setCharAt(int index, char ch)`
- implementácia StringBuffer je najbližšie tomu, čo boli reťazce v Pythone
- StringBuffer je thread safe
- StringBuilder nie je thread safe, preto je trochu rýchlejší

Regulárne výrazy

poznáte z linuxu



[abc]	a, b, c
[^abc]	okrem a, b, c
[a-zA-Z]	a..z,A..Z (interval)
[a-d[m-p]]	[a-dm-p] (zjednotenie)
[a-z&&[def]]	d, e, f (prienik)
[a-z&&[^bc]]	[ad-z] (rozdiel')
[a-z&&[^m-p]]	[a-lq-z] (rozdiel')
19 20	19 alebo 20 (alebo)

X?	raz či vôbec X
X*	viackrát X
X+	aspoň raz X
X{n}	n krát X
X{n,}	aspoň n krát X
X{n,m}	n až m krát X

.	ľub.znak
[0-9]	
\\d	
\\D	
\\s	
\\S	
\\w	
\\W	
^	zač.riadku
\$	koniec riadku
\\b	hranica slova
\\A	zač.vstupu
\\Z	koniec vstupu

[- +] ? ([0 - 9] * \ . [0 - 9] + | [0 - 9] +)

(1 9 | 2 0) \\ d \\ d [- / .] (0 [1 - 9] | 1 [0 1 2]) [- / .] (0 [1 - 9] | [1 2] [0 - 9] | 3 [0 1])

^ . * \\ b (o n e | t w o | t h r e e) \\ b . * \$

Príklady reg.výrazov



`string.matches(regex)` vráti **true**

Regex:

`"java"`

`".*java.*"`

`"^java.*"`

`"[a-z0-9]+"`

`"[^python]+"`

`"[A-F[0-9]]+"`

`"[A-Q&&[K-Z]]+"`

`"[A-Z&&[^F-H]]+"`

`"\\d{3}"`

`"(\\d)\\1\\d"`

`"(\\d\\d\\d)\\1"`

`"[A-Z][a-zA-Z]*"`

`"\\d{3}\\s\\d{2}"`

`"09\\d{2}\\s\\d{3}\\s\\d{3}"`

`"[0]\\d+[/-]\\d+"`

`"(muz|zena)"`

`"[a-zA-Z0-9_]+[@][a-zA-Z0-9_].+"`

`"(19|20)\\d\\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|1[2][0-9]|3[01])"`

`string->>true:`

`"java"`

`"python java kotlin"`

`"java python kotlin"`

`"java9"`

`"java9"`

`"FF00FF"`

`"KLM"`

`"KLM"`

`"158"`

`"558"`

`"567567"`

`"Peter"`

`"821 06"`

`"0905 819 123"`

`"02/2517293"`

`"muz"`

`"jano.mrkva@.com"`

`"(19|20)\\d\\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|1[2][0-9]|3[01])"`

`"1982-12-26"`

`"2021-02-29"` ☹

false

`string->>false:`

`"_java "`

`"python ja_va kotlin"`

`"_java python kotlin"`

`""`

`"kotlin"`

`"ff00ff"`

`"SWISS"`

`"FRANCE"`

`"1234"`

`"123"`

`"567576"`

`"peter"`

`"82106"`

`"4212 777 333"`

`"02 2517293"`

`"dieta"`

`"jano/mrkva@com"`

`"(19|20)\\d\\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|1[2][0-9]|3[01])"`

`"1982-12-26"`

`"26.12.1982"`

// totálna zhoda
 // niekde v reťazci
 // na začiatku riadku
 // malé písmena, cifry, >=1x
 // okrem p,y,t,h,o,n
 // A-F zjednotenie 0-9
 // A-Q prienik K-Z
 // A-Q rozdiel K-Z
 // tri cifry
 // xxd - rovnaké prvé 2cifry
 // xyzxyz–dve rovnaké trojice
 // meno začína veľkým písm.
 // PSČ
 // mobil
 // pevná
 // pohlavie
 // ≈email
 // yyyy-mm-dd



Grupy

Grupy v regexpe zatvoríme do (...)

```
String regexp = "(\\d{4})(\\d{2})(\\d{2})/(\\d{4}|\\d{3})";
Pattern pattern = Pattern.compile(regexp);
Matcher matcher = pattern.matcher("20200225/123");
if (matcher.matches()) {
    if (matcher.groupCount() > 0) {
        for (int g = 0; g <= matcher.groupCount(); g++) {
            System.out.println("group " + g + " is " + matcher.group(g));
        }
    }
}
```

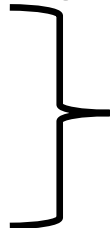
group 0 is 20200225/123

group 1 is 2020

group 2 is 02

group 3 is 25

group 4 is 123



- grupa 0 je vždy celý reťazec

matcher.groupCount() podgrúp

```
for (int g = 0; g <= matcher.groupCount(); g++)
```

Billion-dollar mistake



Tony Hoare

String (či akýkoľvek objekt) môže mať hodnotu **null** má na Svedomí Tony Hoare (okrem iných vecí: QuickSort, CSP,...) , ktorý tak riešil nedokonalosť typového systému Algol→Pascal→C→C++→Java

Sú na tom aj horšie: Javascript má dve hodnoty pre nedefinovanú hodnotu **null** a **undefined**, total chaos...

Moderné jazyky (Scala, Swift, Kotlin) to riešia typmi String (never null), String? (nullable).

Aj v Jave existuje Optional, málokto pozná a používa (asi lebo prišlo až v Jave 8)

2009, he apologised for inventing the [null reference](#):^[20]

I call it my billion-dollar mistake.

It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language ([ALGOLW](#)). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.



Bard's first public mistake cost Google \$100 billion

An underwhelming launch event and an error in an ad put Google's chatbot on the backfoot compared to Microsoft's ChatGPT



<https://qz.com/google-bard-rival-microsoft-chatgpt-ad-error-1850092796>

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;
```

```
        // int[] poleInt;
```

```
        // poleInt = new int[MAX];
```

```
        int[] poleInt = new int[MAX];
```

```
        var poleInt = new int[MAX];
```

```
        for (int i = 0; i < poleInt.length; i++) {
```

```
            poleInt[i] = i + 1;
```

```
            System.out.print(poleInt[i] + " ");
```

```
        } // for
```

```
    } // main
```

```
} // class
```

// konštanta – veľkosť poľa

// definícia poľa

// vytvorenie poľa

// definícia poľa s vytvorením

// definícia poľa s vytvorením

// i < MAX

// inicializácia poľa

typ elementu poľa



Dobré rady

(kuchárka začiatčníka)

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

- ak je len trochu možné, vytvorte/alokujte pole ZÁROVEŇ s jeho deklaráciou. Predpokladá to, že v mieste deklarácie poľa poznáte jeho veľkosť. Ušetríte 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  
var prvocisla = new int[]{2, 3, 5, 7, 11, 13, 19};
```
- 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 Java toto `int[] P` je deklarácia a toto `P = new int[100]` alokácia
- v Java aj C++ pole inicializujete podobne `int P[] = { 1, 2, 3, 4 },`
`int[] P = { 1, 2, 3, 4 }`
- v Java sa vytvorené pole inicializuje hodnotami
 - 0 pre číselné typy, `'\u0000'` pre char, false pre boolean, null iné
- v Java sa nedá indexovať za hranice poľa, kontroluje hranice
- pole je referenčný typ v Java aj C++
- `pole1 = pole2;` je priradením referencií nie kopírovanie polí
- ak potrebujeme kopírovať poľa:
 - C++: `void*memcpy(void *dest, void *source, size_t num)`
 - Java: `System.arraycopy(src, srcPos, dst, dstPos, count)`
 - Java: `Arrays.copyOf(src, len)`
- ak potrebujeme vypísať poľa:
 - `Arrays.deepToString(src)`

```
jshell - Shortcut
jshell> int[][] a = {{1,2,3},{11,22},{33}, null};
a ==> int[4][] { int[3] { 1, 2, 3 }, int[2] { 11, 22 }, int[1] { 33 }, null }

jshell> Arrays.deepToString(a)
$21 ==> "[[1, 2, 3], [11, 22], [33], null]"

jshell> _
```

Polia dvojrozmerné

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

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

```
public class Dvojite {
```

```
    public static void main(String[] args) {
```

```
        var int[][] a = new int[4][];
```

// hlavné pole

```
        for (var int i = 0; i < a.length; i++) {
```

```
            a[i] = new int[i + 1];
```

// podpole

```
            for (var int j = 0; j < a[i].length; j++) {
```

```
                a[i][j] = i * 10 + j;
```

```
                System.out.print(a[i][j] + " ");
```

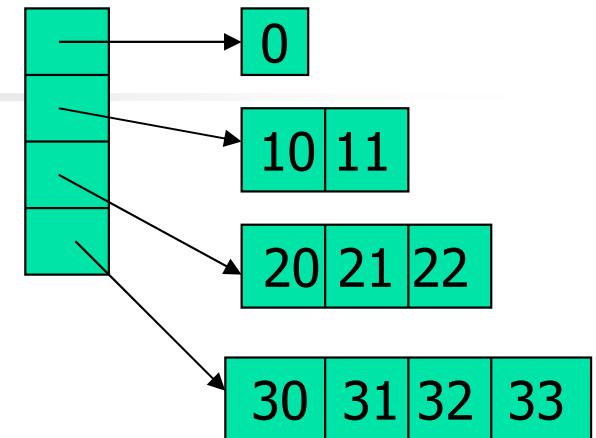
```
            } // for
```

```
            System.out.println();
```

```
        } // for
```

```
    } // main
```

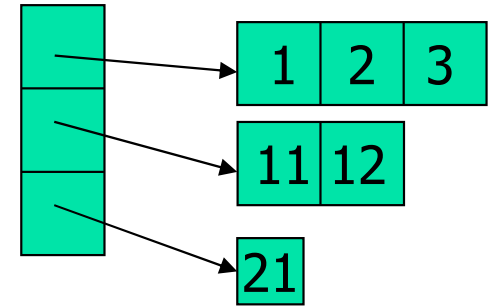
```
} // class
```



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

Inicializácia poľa

(jagged array – štrbavé pole)



- inicializácia dvojrozmerného poľa

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

```
a[0][0] = 1; a[0][1] = 2; a[0][2] = 3; a[0].length == 3  
a[1][0] = 11; a[1][1] = 12; a[1].length == 2  
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]  
f[0]?[1] = new int[8]
```

```
.... Chyba - nemôžem vytvoriť "maticu", ktorej  
.... druhý rozmer nepoznám ale tretí poznám
```

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

Polia a cykly

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

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

Daň za Billion Dollar Mistake:

Skôr než napíšem

`a.blabla`

otestujem, či

`if (a != null) {`

`a.blabla`

`}`



Ako na pole

zretaz(*null*); **NPE**
zretaz(*new* String[]{*null*}); **O.K.**
spocitaj(*new* String[]{*null*}); **NPE O.K.**
spocitaj(*new* String[]{*new* String()});
O.K. *SIMPLY EXPLAINED*

- Ľubovoľné pole daného typu, napr. String[]

```
public class AkoNaPole {  
    public static String zretaz(String[] a) {  
        StringBuffer sb = new StringBuffer();  
        if (a != null)  
            for(int i = 0; i < a.length; i++) sb.append(a[i]);  
        return sb.toString();  
    }  
  
    public static int spocitaj(String[] a) {  
        int vysl = 0;  
        if (a != null)  
            for(int i = 0; i < a.length; i++)  
                if (a[i] != null)  
                    vysl += a[i].length();  
  
        return vysl;  
    }  
}
```



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



Bubble sort

Bubble sort je bezpochyby najobľúbenejší triediaci algoritmus medzi študentami.

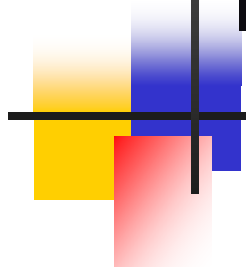
•ale aj ten možno poukázať, vid' Chyba1, Chyba2, Chyba3, ...

```
public class BubbleSort {  
  
    public static void main(String[] args) {  
        int[] a = {4,5,2,12,1,2,3};  
        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;  
                } // if  
            } // for  
        } // for  
        for (int elem:a)  
            System.out.println(elem);  
    }  
}
```

// cyklus for-to-do
// cyklus for-downto-do
// cyklus for-each-element

1
2
2
3
4
5
12





Sú collections lepšie ?

(a môžeme ich už používať?)

```
public static void bubbleSortuj(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);  
            }  
        }  
    }  
}  
  
public static void bubbleSortuj(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;  
            }  
        }  
    }  
}
```

10⁵ - elapsed time:33 s. – 2.35x pomalšie

10⁴ - elapsed time:141 milis.

10⁵ - elapsed time:14s.

10⁶- elapsed time: ???



Je Python lepší ?

```
import random
import datetime
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
alist = random.sample(range(1000000000), 10000)
start = datetime.datetime.now()
bubbleSort(alist)
now = datetime.datetime.now()
print(alist)
print(now-start)
```

10⁴ - elapsed time:15 s. - 106x pomalšie...
10⁵ - elapsed time: 28m05s - 120x pomalšie...

Verzia ADŠ (<http://struct.input.sk/07.html#triedenia>):

```
def bubble_sort(pole):
    for i in range(1, len(pole)):
        for j in range(len(pole)-i):
            if pole[j] > pole[j+1]:
                pole[j], pole[j+1] = pole[j+1], pole[j]
```

<http://interactivepython.org/courselib/static/pythononds/SortSearch/TheBubbleSort.html>

QuickSort

	ADŠ quicksort	.sort() builtin
10^6	7.81s	0.88s
10^6	102.3s	16.2s



v Jave:

10^5 - elapsed time: 23ms. (608x)
 10^6 - elapsed time: 120ms
 10^7 - elapsed time: 1.2s
 10^8 - elapsed time: 12.31s
 10^9 - elapsed time: 123.8s



builtin, Arrays.sort:

10^5 - elapsed time: 39ms.
 10^6 - elapsed time: 129ms
 10^7 - elapsed time: 1.02s
 10^8 - elapsed time: 10.5s
 10^9 - elapsed time: 101.5s



<http://www.vogella.com/tutorials/JavaAlgorithmsQuicksort/article.html>

Súbor: QuickSort.java, QuickSortTest.java

Kockatí trpaslíci



```
static TreeSet<Integer> delitele(int n) {
    int m = n;
    TreeSet<Integer> res = new TreeSet<Integer>();
    for (int a = 1; (long)a*a <= m; a++) {
        if (m % a == 0) {
            res.add(a);                // menší z delitel'ov
            res.add(n/a);              // větší z delitel'ov, asi...
        }
    }
    return res;
}

public static int pocetMoznosti(int n) {
    int count = 0;
    for (Integer a : delitele(n)) {
        for (Integer b : delitele(n/a)) {
            if (b < a) continue;
            ...
        }
    }
}
```

Kockatí trpaslíci



```
static Integer[] delitele(int n) {  
    int m = n;  
    Integer[] res = new Integer[10]; // prvý odhad na počet deliteľov ☺  
    int count = 0;  
    for (int a = 1; (long)a*a <= m; a++) { // nie po n, ani n/2, ale odmocninu m  
        if (m % a == 0) {  
            if (count+2 > res.length) { // hmmm.... je ich viac, tak reallocate  
                Integer[] newres = new Integer[res.length*2]; // reallocate na 2*väčšie  
                System.arraycopy(res, 0, newres, 0, count); // prekopírovanie starého obsahu  
                res = newres; // zahodenie starého obsahu  
            }  
            res[count++] = a; // pridanie menšieho deliteľa  
            if (a != n/a) // čo ak sú rovnaké...  
                res[count++] = n/a; // pridaj, len ak sú rôzne  
        }  
    }  
    res = Arrays.copyOf(res, count); // orež výsledné pole  
    Arrays.sort(res); // treba to utriediť ???  
    return res;  
}
```



Náhodné číslo náhodné pole

```
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]);
```

```
        }
```

```
    }
```

```
}
```

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



Pole ako (vstupný) argument

```
public class Sort {  
    public static void bubbleSortuj(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;  
                } // if  
            } // for  
        } // for  
    }  
    public static void vypis(int[] a) {  
        for (int i:a) System.out.println(i + ",");  
        System.out.println();  
    }  
    public static void main(String[] args) {  
        int[] poleInt = {4,5,2,12,1,2,3};  
        bubbleSortuj(poleInt);  
        vypis(poleInt);  
    }  
}
```

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

// int[7] a – je chyba, lebo
// int[7] nie je typ poľa



Pole ako výstupná hodnota

```
public static int[] generuj(int velkost) {  
    int[] retValue = new int[velkost];  
    Random rand = new Random();  
    for(int i=0; i<velkost; i++)  
        retValue[i] = rand.nextInt(100);  
    return retValue;  
}  
  
public static void main(String[] args) {  
    int[] poleInt = generuj(20);  
}
```

// generuj pole danej veľkosti
// deklaruj a vytvor lokálne pole
// naplň lokálne pole
// vráť referenciu na pole ako
// výsledok funkcie

// pri volaní funkcie si definujeme
// premennu, do ktorej uložíme
// referenciu na vytvorené pole

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 bubbleSortuj

here)' with a checkbox for 'Generate comments' which is unchecked. At the bottom, the 'Class under test' is set to 'Sort' with a 'Browse...' button. At the very bottom of the dialog are buttons for '?', '< Back', 'Next >', 'Finish', and 'Cancel'."/>

New JUnit Test Case

Unit Test Case

The use of the default package is discouraged.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: 02_java Browse...

Package: (default) Browse...

Name: SortTest

Superclass: java.lang.Object Browse...

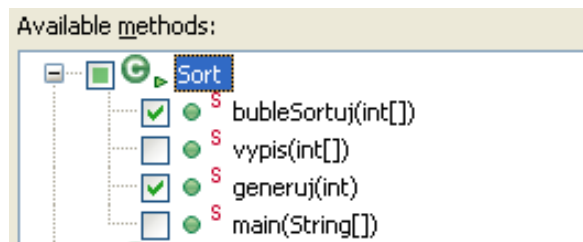
Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Class under test: Sort Browse...

? < Back Next > Finish Cancel



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



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)
assertEquals("test n", n, 100)
assertEquals("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 ďalších

@Anotácie:

@Test

@Before

@After

@Ignore

... a ďalšie

```
@Test(expected=IndexOutOfBoundsException.class) public
void testBubbleSortuj() {
```

```
// toto nebude dobrý test, lebo
// ignoruje nesprávne indexovanie
```

Time for a Break



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 dvojnásobnej veľkosti
                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();
        }
    }
}
```



Nečitateľné úmyselne

System:

```
System.arraycopy(pole, 0, novePole, 0, pole.length);
```

Arrays:

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


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

užitočné statické metódy na prácu s poľami

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

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

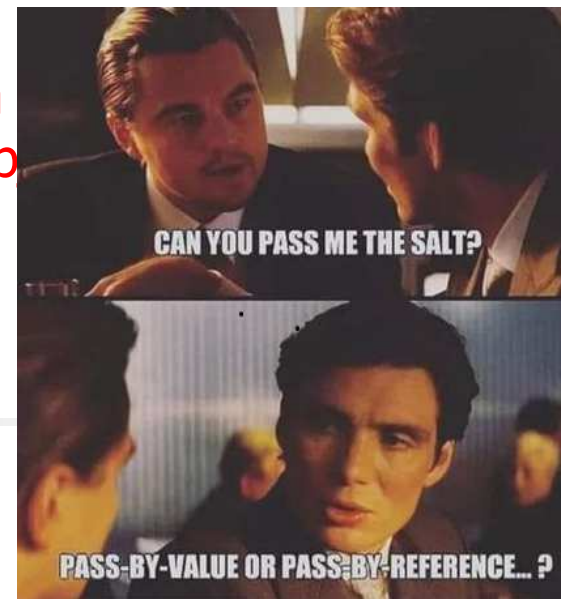
String[] s = {"janko", "marienka", "jozko", "mracik"};
String[] s_copy = new String[4];
System.arraycopy(s, 0, s_copy, 0, s.length);           // kópia poľa
Arrays.sort(s);                                          // triedenie poľa
for(String elem:s) System.out.print(elem+",");          // janko,jozko,marienka,mracik,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

Predávanie argumentov

Základné typy sa
ostatné (polia, ob



```
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
```



Statické metódy

doposiaľ sme až na pár skrytých prípadoch 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 píšú (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;
```

```
        cisla[0] = 1;
```

```
        cisla[1] = 2;
```

```
    public static void main(String[] args) {
```

```
        for (int i=1; i<Prvocisla.cisla.length; i++)
```

```
            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++;
```

```
        }
```

```
    }
```



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); // fajšmekerská 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

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);  
    {  
        // int i = 7;  
        // long i = 7;  
        int j = 8;  
        System.out.println(j);  
    }  
    // System.out.println(j);  
}  
static void tlac2() {  
    int i = 6; int q;  
    System.out.println(i);  
    // for (int i = 1; i < 5; i++)  
        System.out.println(i);  
}
```

// vnorený blok
// chyba - dvojnásobná deklarácia
// chyba - dvojnásobná deklarácia

// koniec vnoreného bloku
// chyba - j nie je viditeľná

// chyba, i už je definovaná

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 bubbleSortuj

here)' with a checkbox for 'Generate comments' which is unchecked. At the bottom, the 'Class under test' is set to 'Sort' with a 'Browse...' button. At the very bottom of the dialog are buttons for '?', '< Back', 'Next >', 'Finish', and 'Cancel'."/>

New JUnit Test Case

Unit Test Case

The use of the default package is discouraged.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: 02_java Browse...

Package: (default) Browse...

Name: SortTest

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Class under test: Sort Browse...

? < Back Next > Finish Cancel



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



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)
assertEquals("test n", n, 100)
assertEquals("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 ďalších

@Anotácie:

@Test

@Before

@After

@Ignore

... a ďalšie

```
@Test(expected=IndexOutOfBoundsException.class) public
void testBubbleSortuj() {
```

```
// toto nebude dobrý test, lebo
// ignoruje nesprávne indexovanie
```

Ako zadat' argumenty

