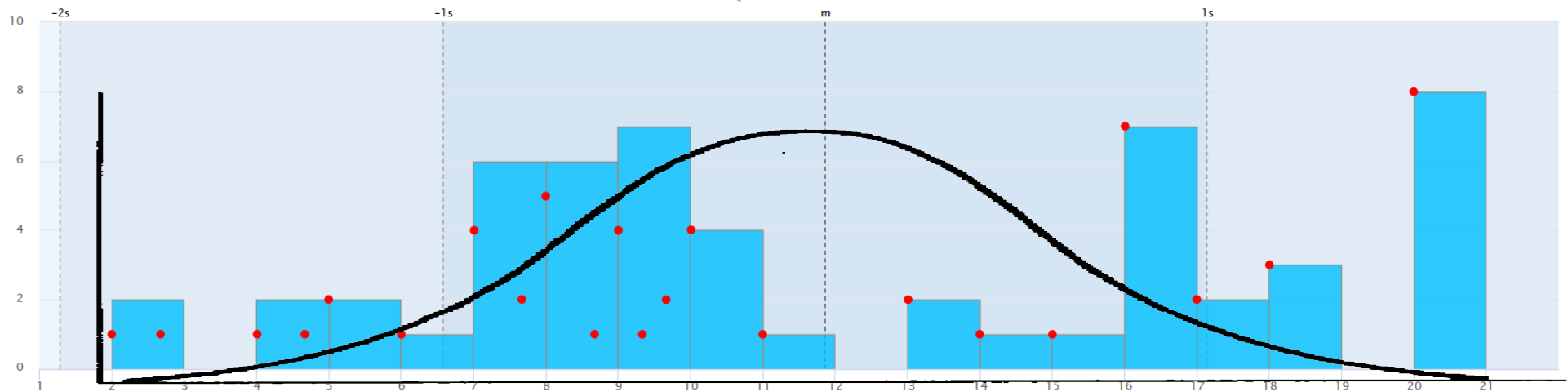


V teórii, je teória a prax to isté.
Ale v praxi nie...

Quad



- Priemer: 11.70 / 20
- Median: 10.00
- Počet: 54

Priebeh:

Dve nepatrné chyby v testoch (DOL),
žiadna evidentná chyba v zadaní
L.I.S.T. – žiadne vážnejšie technické problémy neboli

Top 10

Quad



Júlia	Gablíková	20
Lukáš	Gajdošech	20
Jozef	Kubík	20
Daniel	Kyselica	20
Soňa	Senkovičová	20
Erik	Szalay	20
Tomáš	Takács	20
Michal	Knor	18
Jakub	Soviš	18
Tomáš	Velich	18

Prémie



Lukáš	Gajdošech	27.76
Jozef	Kubík	24.66
Tomáš	Takács	19.64
Tamara	Savkova	18.14
Jakub	Kracina	17.64
Katarína	Dodoková	17.25
Marián	Bagyanszký	16.63
Daniel	Kyselica	16.16
Dávid	Šuba	15.56
Miroslav	Janočko	14.64

Top 10

Total



1.	Lukáš	Gajdošech	62.76
2.	Jozef	Kubík	59.66
3.	Tomáš	Takács	51.64
4.	Daniel	Kyselica	51.16
5.	Marián	Bagyanszky	46.13
6.	Tamara	Savkova	45.14
7.	Jakub	Kracina	45.14
8.	Dávid	Šuba	43.56
9.	Júlia	Gablíková	43.2
10.	Tomáš	Velich	42.64
11.	Soňa	Senkovičová	42.6
12.	Jakub	Soviš	40.6



Quad naj's

- najprekvapujúcejšie zistenie:
že v kruhu sú body, ktorých vzdialenosť od stredu je \leq polomer
- najneobvyklejšia otázka:
môžem si na „Piškvorky“ stiahnuť nejaké piškvorky z netu ?
- najdlhší debug:
vzdialenosť dvoch bodov v rovine (kde je chyba ?) :

```
Math.pow(Math.pow(x1-x2, 2) + Math.pow(y1-y2, 2), 1/2);
```



Perly

(definitely, not the best practice...)

Dobrý copy-paste nikdy nie je zlý..

```
public static boolean vStlpci(char[][] pole) {
    for (int i=0; i<pole.length-5;i++) {
        public static boolean naDiagonale1(char[][] pole) {
            for (int i=0; i<pole.length-5;i++) {
                public static boolean naDiagonale2(char[][] pole) {
                    for (int i=pole.length; i>4; i--) {
                        int j=0;
                        while (j<pole[i].length) {
                            if((pole[i][j]!='X')&(pole[i][j]!='O')){
                                }
                                else{
                                    if (pole[i-1].length>pole[i].length+1) {
                                        if (pole[i][j]==pole[i-1][j+1]) {
                                            if (pole[i-2].length>pole[i].length+2) {
                                                if (pole[i][j]==pole[i-2][j+2]) {
                                                    if (pole[i-3].length>pole[i].length+3) {
                                                        if (pole[i][j]==pole[i-3][j+3]) {
                                                            if (pole[i-4].length>pole[i].length+4) {
                                                                if (pole[i][j]==pole[i-4][j+4]) {
                                                                    return true;
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

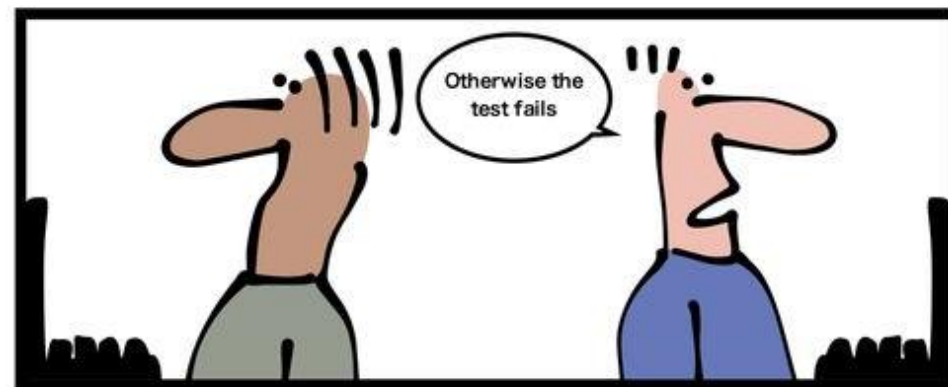
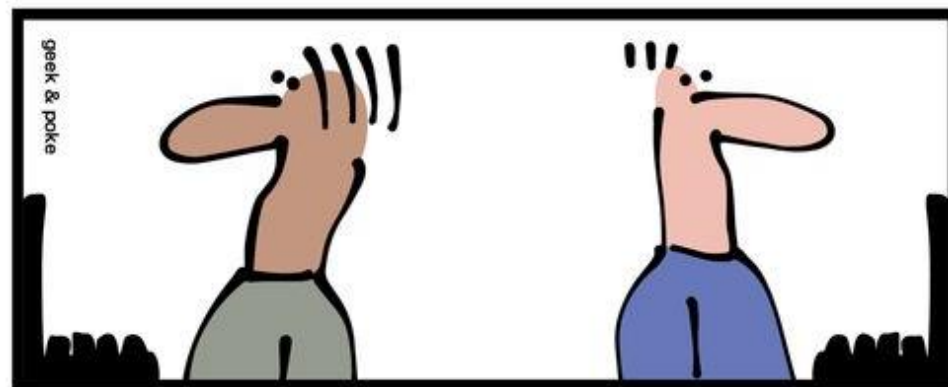
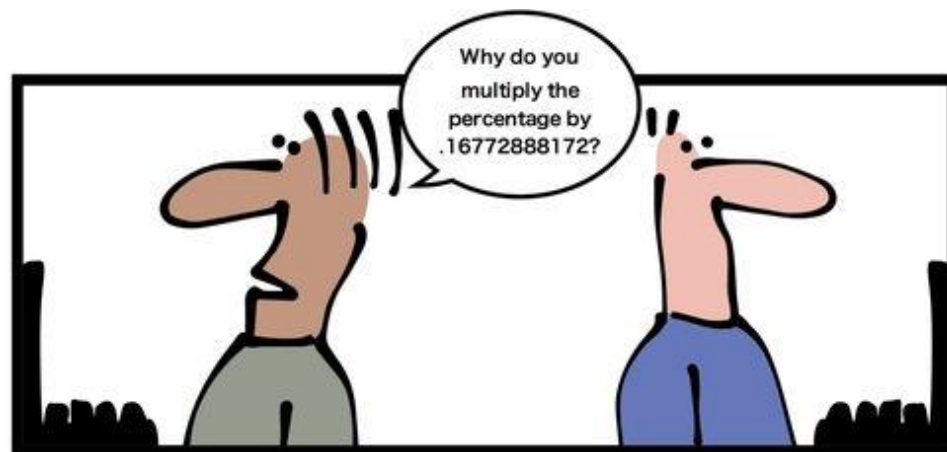
TDD

Test Driven Development

Ak sa vyskytnú prípady,
že kód obabráva test tým,
že kód je zjavne písaný tak,
aby (slabým) testom prešiel,
a to bez ohľadu na zadanie,

prestaneme zverejňovať
(celé) testy...

```
switch(input) {  
  case c1: return res1;  
  case c2: return res2;  
  ...  
}
```

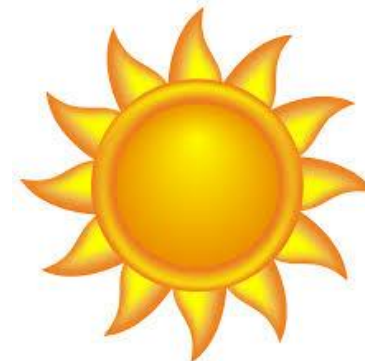


TDD



Čo bolo...

- prišla .jar



- Java 10 ([March 20th 2018](#))



Java 10



Implicitná typová deklarácia-typ premennej si domyslí s inicializačnej hodnoty

```
var a = 0;  
var h = new Hruska();  
var pole = new String[10];  
var list = new ArrayList<String>();  
var map = new HashMap<String, String>();  
class Hruska { ... }
```

Nič to nemení na fakte, že Java zostáva staticky typovaná
nikdy z toho nebude JavaScript, var-podobnosť je čisto náhodná...

Pre tých, čo chcú byť na hrane technológií (leading edge technology):

- ako s Java10 v IntelliJ:
<https://aruld.info/local-variable-type-inference-in-java-10/>
- Thursday, Apr 5, 4:00 PM – 5:00 PM CEST, **Live Webinar: Java 10 and IntelliJ IDEA**, <https://blog.jetbrains.com/idea/tag/java-10/>



Kovariancia „útočila“

```
{ // príklad z prednášky
    Stack50<Podtrieda> stA = new Stack50<Podtrieda>();
    stA.push(new Podtrieda());
    Stack50<Nadtrieda> stB = stA; // ak by to tak bolo, tak toto by išlo
                                // ale ono to v skutočnosti nejde...
                                // dôvod (nabúrame typovú kontrolu
    //stB.push(new Nadtrieda()); // ak by sme to dopustili, potom
}
```

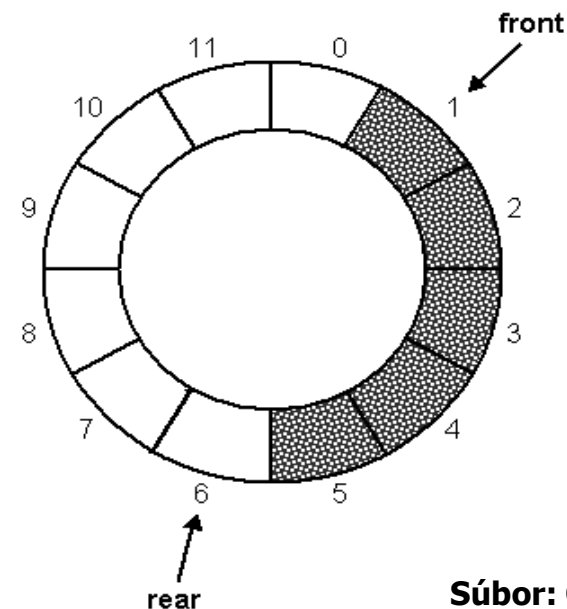
```
{ // otázka LukášaG. : skúsme to s poliami, ktoré sú kovariantné
    Podtrieda[] stA = new Podtrieda[]{ new Podtrieda(), null };
    Nadtrieda[] stB = stA;
    stB[1] = new Nadtrieda();
    System.out.println(stA[1]);
}
```

- kód je skompilovateľný, statická typová kontrola nenájde chybu
- ale počas behu nastane `java.lang.ArrayStoreException: Nadtrieda`
- aspoň že typová homogénnosť poľa je zachovaná/uchránená

Queue - interface



```
public interface QueueInterface<E> {  
    public int size();  
    public boolean isEmpty();  
    public E front() throws EmptyQueueException; // prvý  
    public void enqueue (E element);             // pridaj posledný  
    public E dequeue() throws EmptyQueueException; // zober prvý  
}
```

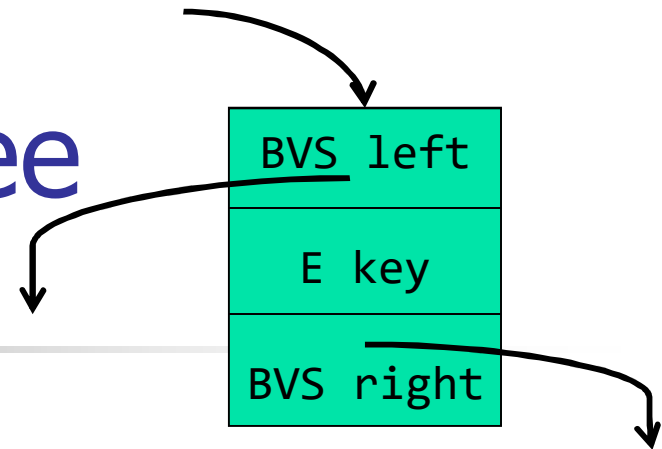


Súbor: QueueInterface.java

BVSTree

(Binárny vyhľadávací/vyvážený strom
býva Midterme)

Polymorfný/parametrizovateľný model:



```
public class BVSTree<E extends Comparable<E>> {
    BVSTree<E> left;
    E key;
    BVSTree<E> right;
    public BVSTree(E key) {                // konštruktor
        this.key = key;
        left = right = null;
    }
}
```

- Comparable (**Comparable<E>**) je interface predpisujúci jedinú metódu:
int compareTo(Object o), <E>int compareTo(E e)
- základné triedy implementujú interface Comparable (ak to dáva zmysel):
Integer, Long, ..., String, Date, ...
- pre iné triedy môžeme dodefinovať metódu **int compareTo()**

Súbory: **BinaryNode.java, BVSTree.java**

Interface Comparable

ak typ nie je primitívny musíme mu prezradiť, ako porovnávať hodnoty tohto typu

```
public class Zamestanec implements Comparable<Zamestanec> {
    private final String meno, priezvisko;
    public Zamestanec(String meno, String priezvp) { // konštruktor
        this.meno = meno; this.priezvisko = priezvp;
    }
    public int compareTo(Zamestanec n) {
        int lastCmp = priezvisko.compareTo(n.priezvisko);
        return (lastCmp != 0 ? lastCmp : meno.compareTo(n.meno));
    }
    // alternatíva
    public int compareTo(Object o) { // zistí, či je daného typu
        if (!(o instanceof Zamestanec)) return -9999;
        Zamestanec n = (Zamestanec)o; // bez cast-exception
        int lastCmp = priezvisko.compareTo(n.priezvisko);
        return (lastCmp != 0 ? lastCmp : meno.compareTo(n.meno));
    }
}

NULL_XAVER.compareTo(NULLOVÁ MATILDA) < 0
"NULLXAVER" > "NULLOVÁMATILDA"
```

Súbor: Zamestnanec.java



BVSTree

(insert)

```
public class BVSTree<E extends Comparable<E>> {  
    BVSNode<E> root;           // smerník na vrchol stromu  
  
    public BVSTree() {  
        root = null;  
    }  
    public void insert(E x) {  
        if (root == null)      // je prázdny ?  
            root = new BVSNode<E>(x);  
        else                   // vytvor jediný uzol  
            root.insert (x);    // inak vsuň do existujúceho stromu  
    }  
}
```

find bolo na cvičení

Súbory: BinaryNode.java, BVSNode.java



BVSTree

(insert)

```
public void insert (E k) {  
    if (k.compareTo(key) < 0)  
        if (left == null)  
            left = new BVSTreeNode<E>(k);  
        else  
            left.insert(k);  
    else if (k.compareTo(key) > 0)  
        if (right == null)  
            right = new BVSTreeNode<E>(k);  
        else  
            right.insert(k);  
}
```

BVSTree – zlé riešenie

(delete)

```
public void delete(E k) { root = root.delete(k); }
```

```
private BVSTree<E> delete(E k) {
```

```
    if (this == null)
```

```
        return null;
```

```
    if (k.compareTo(key) < 0)
```

```
        left = left.delete(k);
```

```
    else if (k.compareTo(key) > 0)
```

```
        right = right.delete(k);
```

```
    else // k == key
```

```
        this = null;
```

Pozor na konštrukcie:

- if (this == null)

- this = null,

pravdepodobne indikujú chybu

BVSTree

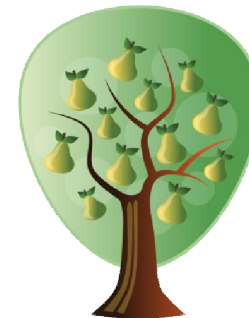
(delete)

```
public void delete(E k) { root = delete(k, root); } // de-selfikácia  
// root prestane byť this-self a stane sa argumentom
```

```
private BVSNode<E> delete(E k, BVSNode<E> t ) {  
    if (t == null )  
        return t;  
    if (k.compareTo(t.key) < 0) // element je v ľavom podstrome  
        t.left = delete(k, t.left); // delete v ľavom podstrome  
    else if(k.compareTo(t.key) > 0) // element je v pravom podstrome  
        t.right = delete(k, t.right); // delete v pravom podstrome  
    else if( t.left != null && // je to on,  
             t.right != null ) { // a má oboch synov  
        t.key = findMin(t.right); // nájdí min.pravého podstromu  
        t.right = delete(t.key, t.right); // rekurz.zmaž minimum  
    } else // pravého podstromu  
        t = (t.left!=null)?t.left:t.right; // ak nemá 2 synov, je to ľahké  
    return t;  
}
```


Klonovanie

(trieda Hruska)



```
interface Clonable {           // vlastná analógia clon(e)able
    public Object copy();      // z istého dôvodu úplne vo vlastnej réžii
}

public class Hruska implements Comparable<Hruska>, Clonable {
    static int allInstances = 0;    // počítadlo všetkých inštancií
    private int instanceIndex;      // koľkatá inštancia v poradí
    private int size;               // veľkosť hrušky

    public Hruska(int size) { this.size = size;
        instanceIndex = allInstances++;
        System.out.println("create Hruska " + instanceIndex);
    }

    public Hruska copy() {
        System.out.println("copy Hruska " + instanceIndex);
        return new Hruska(size);
    }

    public int compareTo(Hruska inaHruska) {
        return Integer.compare(this.size, inaHruska.size);
    }
}
```



Klonovanie

(trieda BVSTNode)

```
class BVSTNode<E extends Comparable<E> & Clonable> implements Clonable {
    BVSTNode<E> left, right; E key;

    static int allInstances = 0;           // počítadlo všetkých inštancií
    private int instanceIndex;           // koľkatá inštancia v poradí

    public BVSTNode(E theKey) { key = theKey; left = right = null;
        instanceIndex = allInstances++;
        System.out.println("create BVSTNode " + instanceIndex);
    }

    public BVSTNode<E> copy() {
        System.out.println("copy BVSTNode " + instanceIndex);
        BVSTNode<E> clone = new BVSTNode<E>(
            (key!=null)?(E)(key.copy()):null    );

        clone.left  = (left != null) ? left.copy():null;
        clone.right = (right != null) ? right.copy():null;
        return clone;
    }
}
```



Klonovanie

(trieda BVSTree)

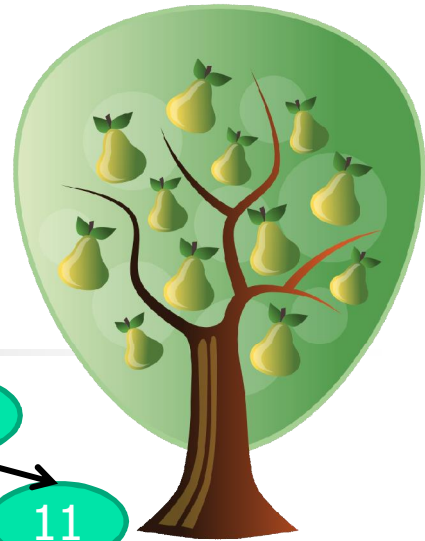
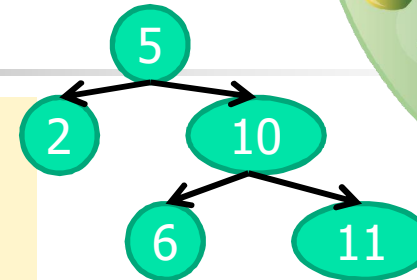
```
class BVSTree<E extends Comparable<E> & Clonable> implements Clonable {  
    BVSTree<E> root;  
    static int allInstances = 0;  
    private int instanceIndex;  
  
    public BVSTree () {  
        instanceIndex = allInstances++;  
        System.out.println("create BVSTree " + instanceIndex);  
        root = null;  
    }  
    public BVSTree<E> copy() {  
        System.out.println("copy BVSTree " + instanceIndex);  
        BVSTree<E> clone = new BVSTree<E>();  
        clone.root = (root != null)?root.copy():null;  
        return clone;  
    }  
}
```

Pear Tree Copy

(Klonovanie stromu hrušiek)

```
create BVSTree 0
create Hruska 0
create BVSTree 0
create Hruska 1
create BVSTree 1
create Hruska 2
create BVSTree 2
create Hruska 3
create BVSTree 3
create Hruska 4
create BVSTree 4
```

```
BVSTree<Hruska> s =
    new BVSTree<Hruska>();
Random r = new Random();
for(int i=0; i<5; i++)
    s.insert(new Hruska(r.nextInt(19)));
```



```
<key:som hruska 5:> - <left:som hruska 2>, <right:som hruska 10>
<key:som hruska 2:> - <x>, <x>
<key:som hruska 10:> - <left:som hruska 6>, <right:som hruska 11>
<key:som hruska 6:> - <x>, <x>
<key:som hruska 11:> - <x>, <x>
```

```
(BVSTree<Hruska>)
s.copy();
copy BVSTree 0
create BVSTree 1
copy BVSTree 0
copy Hruska 0
create Hruska 5
create BVSTree 5
copy BVSTree 3
copy Hruska 3
create Hruska 6
create BVSTree 6
```

```
copy BVSTree 1
copy Hruska 1
create Hruska 7
create BVSTree 7
copy BVSTree 4
copy Hruska 4
create Hruska 8
create BVSTree 8
copy BVSTree 2
copy Hruska 2
create Hruska 9
create BVSTree 9
```