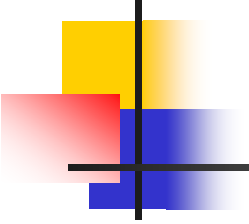





# Vo štvrtok Quadterm

Vždy píšete kód tak, ako by ten chlapík, čo ho po vás bude udržiavať, mal byť násilnícky psychopat, ktorý bude vedieť, kde bydlíte.

- internet bude, o.i. JAVA API, SO, prednášky, cvičenia, ..., USB
- fair hra, nulová tolerancia k GM, FB, SKP, ...
- isic pre vstup do H3/H6
- všetky bodovacie testy (zo serveru) sú zverejnené
- nie sú záťažové, ale jeden nikdy nevie...
- obsahujú viac testovacích vstupov ako zadanie...
- zostava obsahuje zdrojáky, DO KTORÝCH dopisujete vaše metódy
- stiahnite si ich (na vrchu zostavy hľadajte src.zip, alebo celyprojekt.zip)
- je dobre vedieť naimportovať celý projekt do workspace, a mám to...
- reťazce, polia, triedy-objekty-dedenie-abstraktná ... príkladov bolo dosť
- pozrite si String/StringBuffer/StringBuilder, Arrays, niečo sa môže hodiť
- všetky 3 príklady preriešili/pretestovali cvičiaci s limitom na 30 min.
- záver Q1 bude panika, submitovací-testovací ošial a LIST padne/nestíha
- CHRÁNIM SI ZDROJÁKY, NEOPUSTÍM H3/H6, kým posledná verzia nie je v LISTe (u cvičiaceho na USB, ak by apokalypsa), testovať sa dajú aj neskôr

- 
- Neberte si kľúčiky od veľkej miešačky, kým neviete robiť s malou lopatou
  - všetky príklady sú riešiteľné bez ArrayListu



- 
- Java by mohla být dobrým příkladem toho, jak má vypadat programovací jazyk.
  - Ale aplikace v Javě jsou dobrým příkladem toho, jak by aplikace neměly vypadat

```
if (condition == true) ...  
if (condition == false) ...  
if ((nasiel==false)==false)...
```

```
if (cond == false) return true  
else return false;
```

```
if(condition) {  
    foo();  
    return;  
} else { // zbytočné else  
    bar();  
}
```

```
"Value: " + a.toString()  
this.p.doit()
```

```
if(condition) ...  
if(!condition) ...  
if (nasiel) ...
```

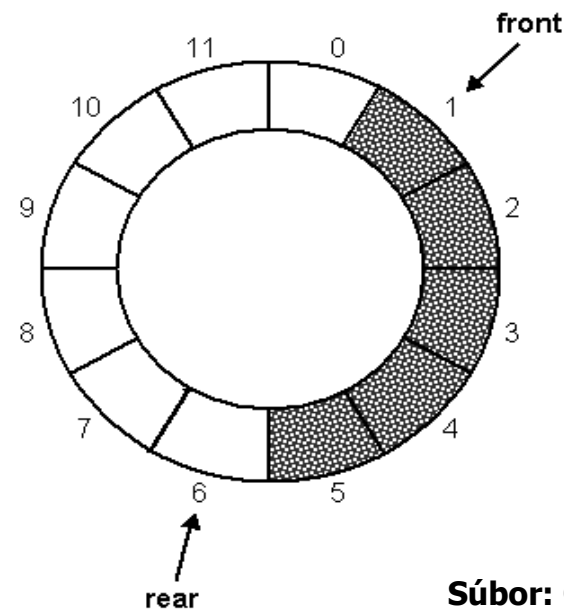
```
return !cond;
```

```
if(condition) {  
    foo();  
    return;  
}  
bar();
```

```
"Value: " + a  
p.doit()
```

# Queue - interface

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



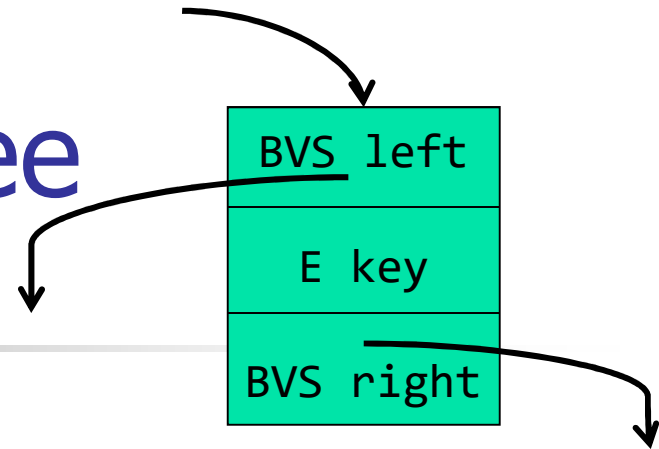
Súbor: QueueInterface.java



# BVSTree

(Binárny vyhľadávací/vyvážený strom  
býva Midterme)

Polymorfný/parametrizovateľný model:



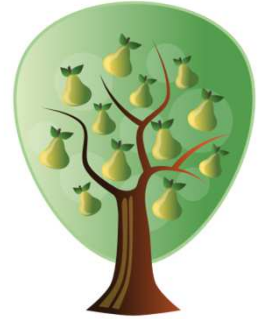
```
public class BVSTree<E extends Comparable<E>> {
    BVSTree<E> left;
    E key;
    BVSTree<E> right;
    public BVSTree(E key) {                // konštruktor
        this.key = key;
        left = right = null;
    }
}
```

- 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

# 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;  
    }  
}
```

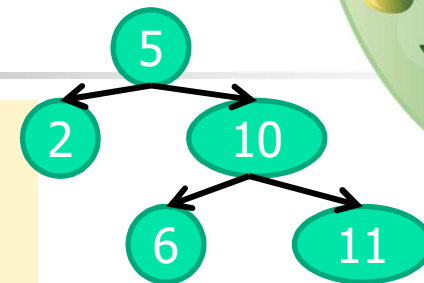


# Pear Tree Copy

(Klonovanie stromu hrušiek)

```
create BVSTree 0
create Hruska 0
create BVSTNode 0
create Hruska 1
create BVSTNode 1
create Hruska 2
create BVSTNode 2
create Hruska 3
create BVSTNode 3
create Hruska 4
create BVSTNode 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 BVSTNode 0
copy Hruska 0
create Hruska 5
create BVSTNode 5
copy BVSTNode 3
copy Hruska 3
create Hruska 6
create BVSTNode 6
```

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

# 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...