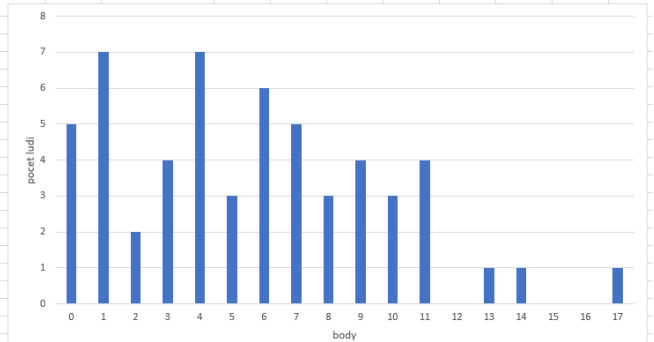


Quadterm 1

priemer	5,625
median	5,5
std	3,966658



- AVG: 6.6, MEDIAN: 6.4, TOTAL: 17 (15) - katastrofa ☹
 - možná oprava Quad1/Mid/Quad2 ako 1.skúškový termín (nepočíta sa...)
- Priebeh:
 - žiadna faktická chyba v testoch, ani evidentná chyba v zadaní
 - L.I.S.T./capek – vydržal, žiadne vážnejšie technické problémy neboli
- „nie najšťastnejšie“ – 1.príklad Morse zrejme nebol najľahší, ale bol prvý...
- Prekvapilo:
 - STRESS FAKTOR
 - dekodujVsetky() vyriešil 1, pokusili sa 2 – backtracking je asi tu tajomný
 - Gaussove prvé tri body „zadarmo“
 - „ako naimportovať projekt“ a nezabíjať minúty prerábaním testov
 - „akom mám z testu poznať, čo mi to vrátilo a čo mi to malo vrátiť“
 - „ako tie testy v Netbeanse...“
- rozhodli sme sa čítať vaše kódy a dobodovať
 - rozpracované riešenie, ktoré neprešlo testom
 - odhadnúť, ako ďaleko ste od cieľa boli (často znamenalo vám to odladiť)
- Midterm: v pridelenom termíne: utorok, 4.4. 18:00, A/B (?)
 - písomný test, vzory za posledné roky sú zverejnené



Top 10

Priezvisko	Prémia ▼
Polakovič	26.75
Šafárik	16.5
Ferienčík	16.5
Priebera	14.5
Mériová	14
Halasi	12.7
Vávrová	12.5
Sláma	12.45
Furmánek	11.5
Bočinec	10.75

...	Meno	Priezvisko	úloh Quadterm ▼
1.	Zoltan	Onody	17
2.	Miroslav	Šafárik	14
3.	Michal	Singer	13.5
4.	Viktória	Šimšíková	11.5
5.	Tomáš	Bočinec	11.4
6.	Radovan	Balog	11
7.	Matej	Kopčík	11
8.	Ján Filip	Kotora	10.7
9.	Kristína	Karafová	10.65
10.	Patrícia	Marmanová	10.5

...	Meno	Priezvisko	Spolu ▼
1.	Adam	Polakovič	51.95
2.	Miroslav	Šafárik	43.6
3.	Viktória	Šimšíková	36.1
4.	Miroslav	Ferienčík	35.85
5.	Michal	Singer	35.1
6.	Tomáš	Bočinec	34.65
7.	Patrícia	Marmanová	33.35
8.	Patrik	Priebera	33.3
9.	Kristína	Karafová	33.25
10.	Patrik	Faşang	32.15

Perly

(definitely, not the best practice...)

```
for(...){  
    if ( Arrays.asList(letters).indexOf(ss) >= 0 ) {  
        char c = (char)Arrays.asList(letters).indexOf(ss);
```

```
for(int j = 65; j < 91; j++)    hovorí    for(int j = 'A'; j < '['; j++)  
for(int j = 'A'; j <= 'Z'; j++)
```





Ako na MID

- Motto: „prekvapiť študenta znamená, že nakoniec budem prekvapený“
- MIDTERM sa píše na papier,
- zakázané sú elektronické pomôcky,
- kódy sa čítajú a nekompilujú,
- na syntaxi záleží len potiaľ, ak už sa nedá vôbec pochopiť, čo autor* myslel
- ak porovnávate reťazce `if (s1==s2)`, tak to nie je syntax ale sémantika...
- knihy, poznámky sú povolené, ale nevyrúbte veľa stromov...
- Pravidlo: Čo som doteraz nečítal, určite mi nepomôže
- Pripravím sa sám:

*toto chcela byť funkcia,
ktorá nájde v reťazci ...

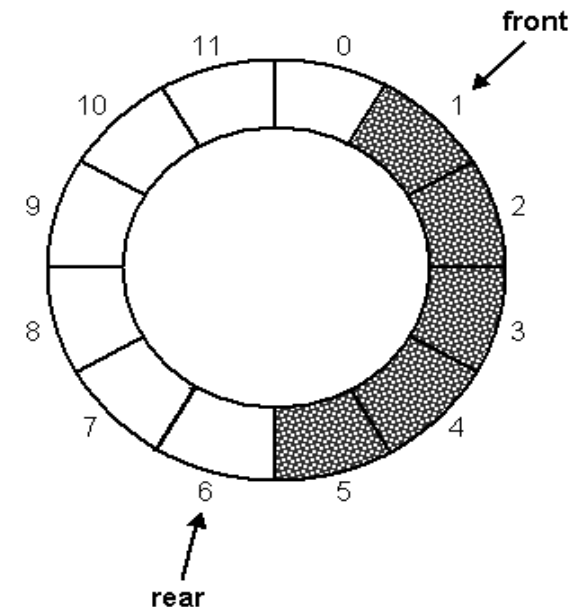
[4.4.] Midterm [2007, 2008_A, 2008_B, 2009_A, 2009_B, 2011, 2012, 2014, 2015, 2016]

- prídem na konzultáciu:
 - k dispo: štv-pia-po
 - to predpokladá, že viem, čo neviem
 - ešte lepšie, ak viem, že neviem niečo, čo bude na MID (pozriem si to skôr)
 - ak neviem, čo viem,
... zrejme to bude blízke stretnutie tretieho druhu

Queue - interface

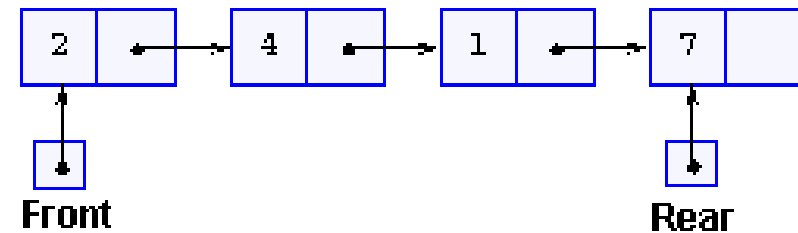
```
public interface QueueInterface<E> {  
    public int size();  
    public boolean isEmpty();  
    public E front() throws EmptyQueueException;  
    public void enqueue (E element);  
    public E dequeue() throws EmptyQueueException;  
}
```

// prvý
// pridaj posledný
// zober prvý



Súbor: QueueInterface.java

Queue



Reprezentácia:

```
public void enqueue(E elem) {
    Node<E> node = new Node<E>();
    node.setElement(elem);
    node.setNext(null);
    if (size == 0) // prvý prvok prázdneho frontu
        front = node;
    else
        rear.setNext(node);
    rear = node;
    size++;
}
```

```
Node<E> front=null; // prvý
Node<E> rear=null; // posledný
int size = 0; // veľkosť
```

```
public E dequeue() throws EmptyQueueException {
    if (size == 0)
        throw new
            EmptyQueueException("Queue is empty.");
    E tmp = front.getElement();
    front = front.getNext();
    size--;
    if (size == 0) // bol to posledný prvok frontu
        rear = null;
    return tmp;
}
```

Súbor: Queue.java

Balík

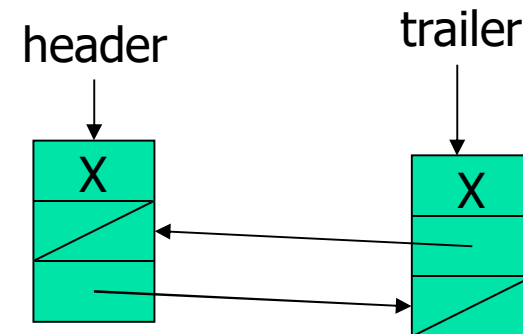
(Sentinel nodes)

```
public class Deque<E> implements DequeInterface<E> {
```

```
    protected DLNode<E> header, trailer; // reprezentácia balíka dvomi  
    protected int size;                  // pointerami na zač. a koniec
```

```
    public Deque() { // konštruktor  
        header = new DLNode<E>();  
        trailer = new DLNode<E>();  
        header.setNext(trailer);  
        trailer.setPrev(header);  
        size = 0;  
    }
```

```
    public E getFirst() throws Exception {  
        if (isEmpty()) throw new Exception("Deque is empty.");  
        return header.getNext().getElement();  
    }
```

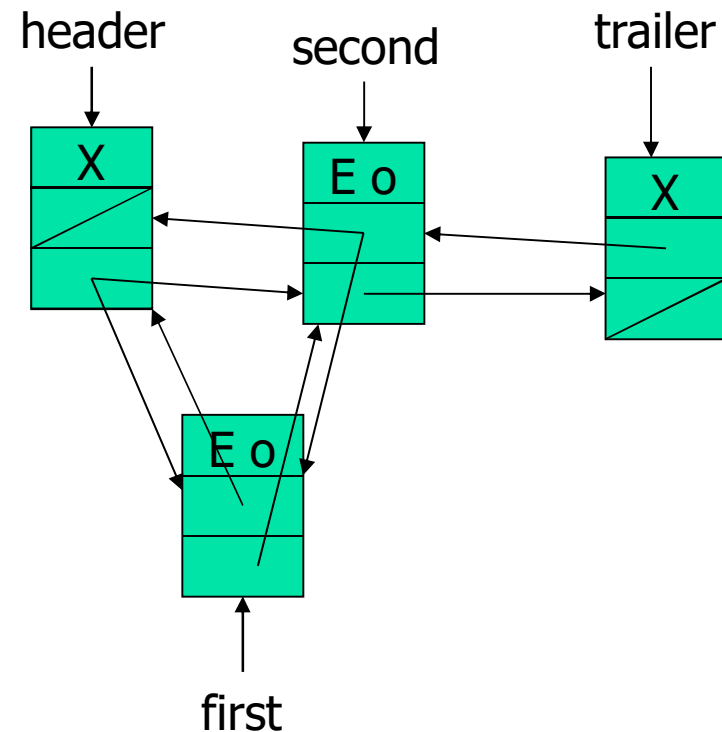


Súbor: DequeInterface.java

Balík – implementácia

(addFirst)

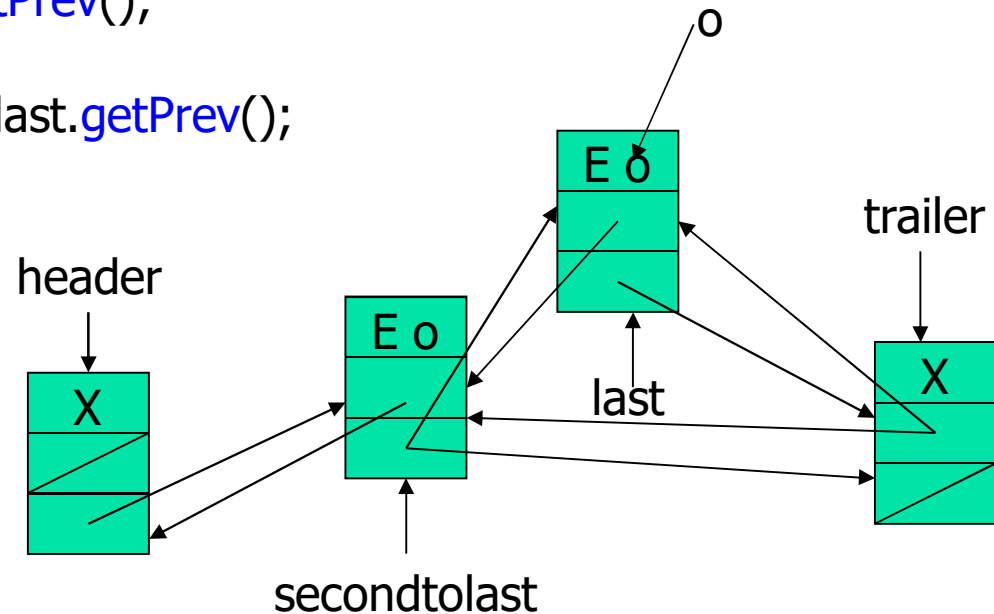
```
public void addFirst(E o) {  
    DLNode<E> second = header.getNext();  
    DLNode<E> first = new DLNode<E>(o, header, second);  
    second.setPrev(first);  
    header.setNext(first);  
    size++;  
}
```



Balík – implementácia

(removeLast)

```
public E removeLast() throws Exception {  
    if (isEmpty())  
        throw new Exception("Deque is empty.");  
    DLNode<E> last = trailer.getPrev();  
    E o = last.getElement();  
    DLNode<E> secondtolast = last.getPrev();  
    trailer.setPrev(secondtolast);  
    secondtolast.setNext(trailer);  
    size--;  
    return o;  
}
```



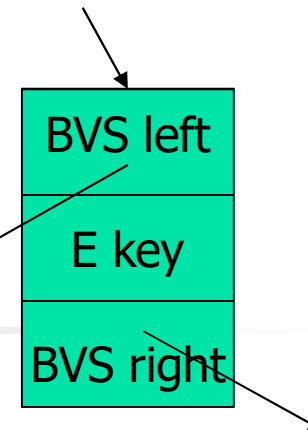
BVSTree

(Binárny vyhľadávací/vyvážený strom
býva Midterme)

parametrizovateľný model:

```
public class BVSTree<E extends Comparable<E>> {  
    BVSTree left;  
    E key;  
    BVSTree 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()





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 priezvisko) { // konštruktor  
        this.meno = meno; this.priezvisko = priezvisko;  
    }  
    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) {  
        if (!(o instanceof Zamestanec)) return -9999; // zistí, či je daného typu  
        Zamestanec n = (Zamestanec)o; // bez cast-exception  
        int lastCmp = priezvisko.compareTo(n.priezvisko);  
        return (lastCmp != 0 ? lastCmp : meno.compareTo(n.meno));  
    }  
}
```

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) {  
        root = (root == null)? // je prázdny ?  
            new BVSNode<E>(x): // vytvor  
                                // jediný uzol  
        root.insert(x); // inak vsuň do  
                        // existujúceho stromu  
    }  
  
    public BVSNode<E> insert (E k) {  
        if (k.compareTo(key) < 0)  
            if (left == null)  
                left = new BVSNode<E>(k);  
            else  
                left = left.insert(k);  
        else  
            if (right == null)  
                right = new BVSNode<E>(k);  
            else  
                right = right.insert(k);  
        return this;  
    }  
}
```

BVSTree – zlé riešenie

(delete)

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

```
private BVSTreeNode<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)

Pozor na konštrukcie:

- this = null,
 - if (this == null)
- pravdepodobne indikujú chybu

```
public void delete(E k) { root = delete(k, root); } // de-selfikácia
// root prestal byť this-self a stal sa argumentom
private BVSTree<E> delete(E k, BVSTree<E> t ) {
    if (t == null )
        return t;
    if (k.compareTo(t.key) < 0)
        t.left = delete(k, t.left);
    else if(k.compareTo(t.key) > 0)
        t.right = delete(k, t.right);
    else if( t.left != null && t.right != null ) {
        t.key = findMin(t.right).key;
        t.right = delete(t.key, t.right);
    } else
        t = (t.left != null) ? t.left : t.right;
    return t;
}
```

// element je v ľavom podstromu
// delete v ľavom podstromu
// element je v pravom podstromu
// delete v pravom podstromu
// je to on, a má oboch synov
// nájdi min.pravého podstromu
// rekurz.zmaž minimum
// pravého podstromu
// ak nemá 2 synov, je to ľahké



Klonovanie

(z istého dôvodu úplne vo vlastnej réžii)

```
interface Clonable { // vlastná analógia clon(e)able
    public Object copy();
}

public class Hruska implements Comparable<Hruska>, Clonable {
    static int allInstances = 0;
    private int instanceIndex;
    private int size;
    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 new Integer(this.size).compareTo(inaHruska.size);
    }
}
```



Klonovanie

(z istého dôvodu úplne vo vlastnej réžii)

```
class BVSTNode<E extends Comparable<E> & Clonable> implements Clonable {
    BVSTNode<E> left, right; E key;
    static int allInstances = 0;
    private int instanceIndex;

    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

(z istého dôvodu úplne vo vlastnej réžii)

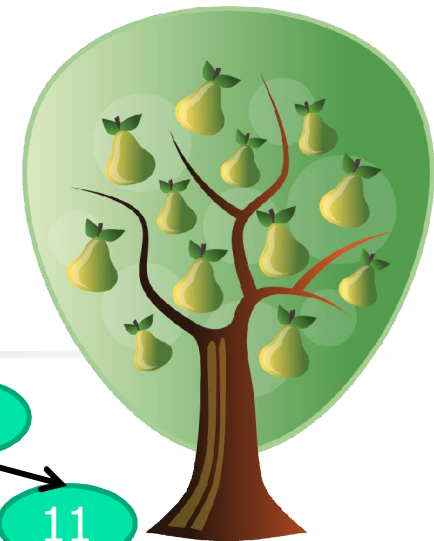
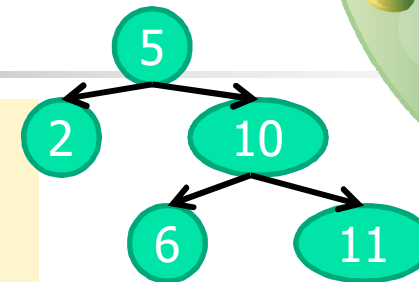
```
class BVSTree<E extends Comparable<E> & Clonable> implements Clonable {  
    BVSTree<E> root;  
    static int allInstances = 0;  
    private int instanceIndex;  
  
    public BVSTree () {  
        instanceIndex = allInstances++;  
        System.out.println("create BVSTree " + instanceIndex);  
        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

```
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>
```

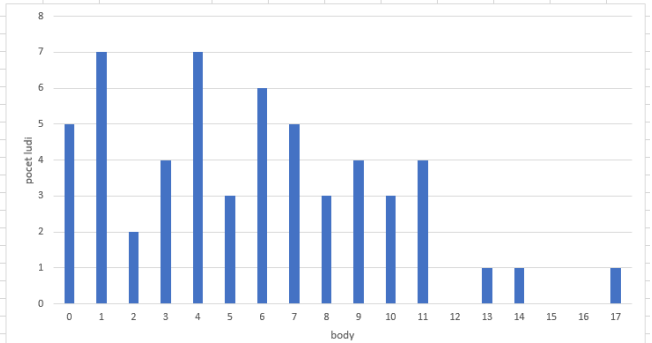


```
(BVSTree<Hruska>)
    s.copy();
copy BVSTree 0
create BVSTree 1
copy BVSTNode 0
copy Hruska 0
create Hruska 5
create BVSTNode 5
copy BVSTNode 3
copy Hruska 3
create Hruska 6
create BVSTNode 6
```

```
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
```

Quadterm 1

priemer	5,625
median	5,5
std	3,966658



- AVG: 6.6, MEDIAN: 6.4, TOTAL: 17 (15)-podpriemer/neúspech/katastrofa ☹
 - možná oprava Quad1/Mid/Quad2 ako 1.skúškový termín (nepočíta sa...)
- Priebeh:
 - žiadna faktická chyba v testoch, ani evidentná chyba v zadaní
 - L.I.S.T./capek – vydržal, žiadne vážnejšie technické problémy neboli
- „nie najšťastnejšie“ – 1.príklad Morse zrejme nebol najľahší, ale bol prvý ☹
- Prekvapilo:
 - STRESS FAKTOR
 - dekodujVsetky() vyriešil 1, pokusili sa 2 – backtracking je asi tu tajomný...
 - Gaussove prvé tri body „zadarmo“
 - „ako naimportovať projekt“ a nezabíjať minúty prerábaním testov
 - „akom mám z testu poznať, čo mi to vrátilo a čo mi to malo vrátiť“
 - „ako tie testy v Netbeanse...“
- rozhodli sme sa čítať vaše kódy a dobodovať
 - rozpracované riešenie, ktoré neprešlo testom
 - odhadnúť, ako ďaleko ste od cieľa boli (často znamenalo vám to odladiť)
- Midterm: v pridelenom termíne: utorok, 4.4. 18:00, A/B (?)
 - písomný test, vzory za posledné roky sú zverejnené



Top 10

Priezvisko	Prémia ▼
Polakovič	26.75
Šafárik	16.5
Ferienčík	16.5
Priebera	14.5
Mériová	14
Halasi	12.7
Vávrová	12.5
Sláma	12.45
Furmánek	11.5
Bočinec	10.75

...	Meno	Priezvisko	úloh Quadterm ▼
1.	Zoltan	Onody	17
2.	Miroslav	Šafárik	14
3.	Michal	Singer	13.5
4.	Viktória	Šimšíková	11.5
5.	Tomáš	Bočinec	11.4
6.	Radovan	Balog	11
7.	Matej	Kopčík	11
8.	Ján Filip	Kotora	10.7
9.	Kristína	Karafová	10.65
10.	Patrícia	Marmanová	10.5

...	Meno	Priezvisko	Spolu ▼
1.	Adam	Polakovič	51.95
2.	Miroslav	Šafárik	43.6
3.	Viktória	Šimšíková	36.1
4.	Miroslav	Ferienčík	35.85
5.	Michal	Singer	35.1
6.	Tomáš	Bočinec	34.65
7.	Patrícia	Marmanová	33.35
8.	Patrik	Priebera	33.3
9.	Kristína	Karafová	33.25
10.	Patrik	Faşang	32.15

Perly

(definitely, not the best practice...)

```
for(...){  
    if ( Arrays.asList(letters).indexOf(ss) >= 0 ) {  
        char c = (char)Arrays.asList(letters).indexOf(ss);
```

```
for(int j = 65; j < 91; j++)    hovorí    for(int j = 'A'; j < '['; j++)  
for(int j = 'A'; j <= 'Z'; j++)
```





Ako na MID

- Motto: „prekvapiť študenta znamená, že nakoniec budem prekvapený“
- MIDTERM sa píše na papier,
- zakázané sú elektronické pomôcky,
- kódy sa čítajú a nekompilujú,
- na syntaxi záleží len potiaľ, ak už sa nedá vôbec pochopiť, čo autor* myslel
- ak porovnávate reťazce `if (s1==s2)`, tak to nie je syntax ale sémantika...
- [vlastné] knihy, poznámky sú povolené, ale nevyrúbte veľa stromov...
- Pravidlo: *Čo som doteraz nečítal, určite mi nepomôže*
- Pripravím sa sám:

*toto chcela byť funkcia,
ktorá nájde v reťazci ...

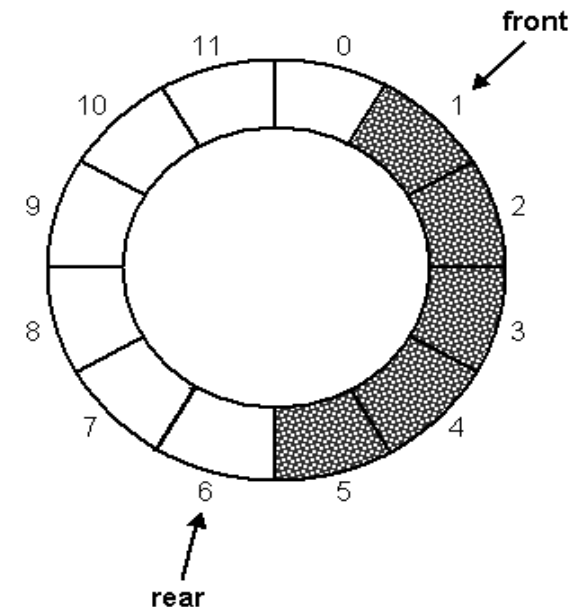
[4.4.] Midterm [2007, 2008_A, 2008_B, 2009_A, 2009_B, 2011, 2012, 2014, 2015, 2016]

- prídem na konzultáciu:
 - k dispo: štv-pia-po
 - to predpokladá, že viem, čo neviem
 - ešte lepšie, ak viem, že neviem niečo, čo bude na MID (pozriem si to skôr)
 - ak neviem, čo viem,
... zrejme to bude blízke stretnutie tretieho druhu

Queue - interface

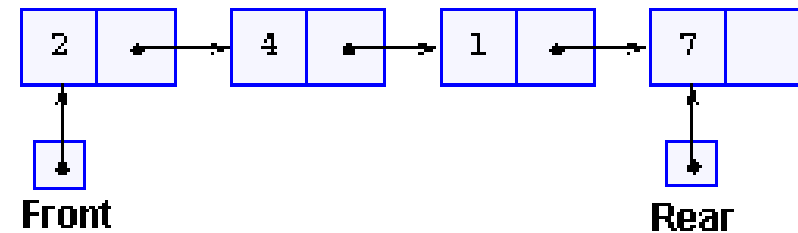
```
public interface QueueInterface<E> {  
    public int size();  
    public boolean isEmpty();  
    public E front() throws EmptyQueueException;  
    public void enqueue (E element);  
    public E dequeue() throws EmptyQueueException;  
}
```

// prvý
// pridaj posledný
// zober prvý



Súbor: QueueInterface.java

Queue



Reprezentácia:

```
public void enqueue(E elem) {
    Node<E> node = new Node<E>();
    node.setElement(elem);
    node.setNext(null);
    if (size == 0) // prvý prvok prázdneho frontu
        front = node;
    else
        rear.setNext(node);
    rear = node;
    size++;
}
```

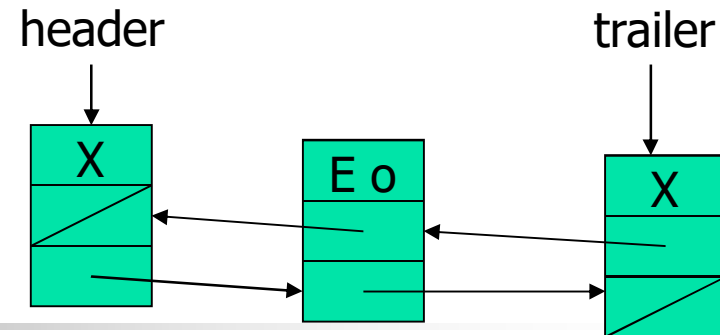
```
Node<E> front=null; // prvý
Node<E> rear=null; // posledný
int size = 0; // veľkosť
```

```
public E dequeue() throws EmptyQueueException {
    if (size == 0)
        throw new
            EmptyQueueException("Queue is empty.");
    E tmp = front.getElement();
    front = front.getNext();
    size--;
    if (size == 0) // bol to posledný prvok frontu
        rear = null;
    return tmp;
}
```

Súbor: Queue.java

Balík

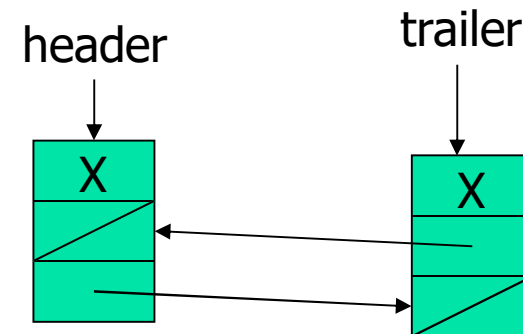
(Sentinel nodes)



```
public class Deque<E> implements DequeInterface<E> {
```

```
    protected DLNode<E> header, trailer; // reprezentácia balíka dvomi
    protected int size;                  // pointerami na zač. a koniec
```

```
    public Deque() { // konštruktor
        header = new DLNode<E>();
        trailer = new DLNode<E>();
        header.setNext(trailer);
        trailer.setPrev(header);
        size = 0;
    }
```



```
    public E getFirst() throws Exception {
        if (isEmpty()) throw new Exception("Deque is empty.");
        return header.getNext().getElement();
    }
```

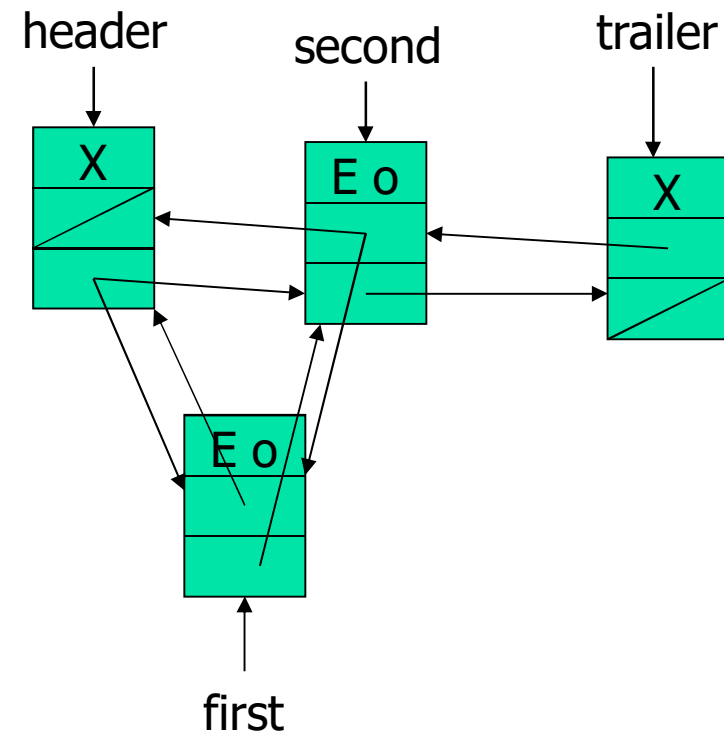
Súbor: DequeInterface.java

Balík – implementácia

(addFirst)

```
public void addFirst(E o) {  
    DLNode<E> second = header.getNext();  
    DLNode<E> first = new DLNode<E>(o, header, second);  
    second.setPrev(first);  
    header.setNext(first);  
    size++;  
}
```

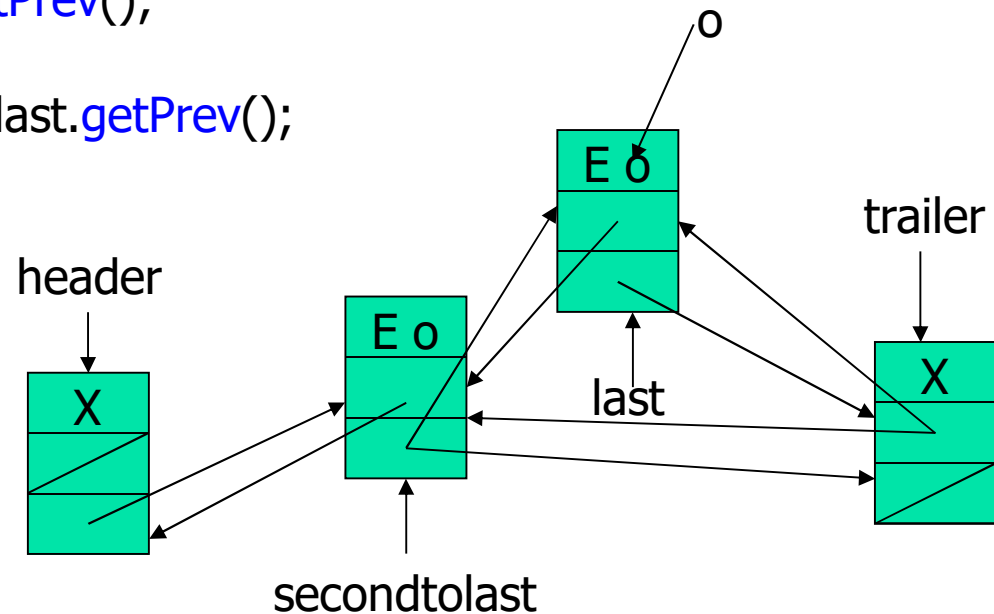
Zmysel použitia sentinel nodes je v tom, že nemusím testovať špeciálne prípady, ako napr. prázdny zoznam/balík



Balík – implementácia

(removeLast)

```
public E removeLast() throws Exception {  
    if (isEmpty())  
        throw new Exception("Deque is empty.");  
    DLNode<E> last = trailer.getPrev();  
    E o = last.getElement();  
    DLNode<E> secondtolast = last.getPrev();  
    trailer.setPrev(secondtolast);  
    secondtolast.setNext(trailer);  
    size--;  
    return o;  
}
```



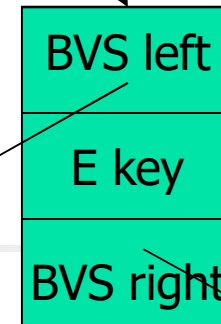
BVSTree

(Binárny vyhľadávací/vyvážený strom
býva Midterme)

parametrizovateľný model:

```
public class BVSTree<E extends Comparable<E>> {  
    BVSTree left;  
    E key;  
    BVSTree 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()





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 priezvisko) { // konštruktor  
        this.meno = meno; this.priezvisko = priezvisko;  
    }  
    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) {  
        if (!(o instanceof Zamestanec)) return -9999; // zistí, či je daného typu  
        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

find je cvičenie

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) {  
        root = (root == null)? // je prázdny ?  
            new BVSNode<E>(x): // vytvor  
                                // jediný uzol  
        root.insert(x); // inak vsuň do  
                        // existujúceho stromu  
    }
```

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

Súbory: **BinaryNode.java**, **BVSNode.java**

BVSTree – zlé riešenie

(delete)

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

```
private BVSTreeNode<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)

Pozor na konštrukcie:

- `this = null`,
- `if (this == null)`

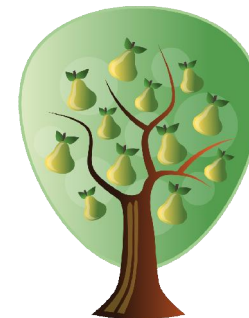
pravdepodobne indikujú chybu

```
public void delete(E k) { root = delete(k, root); } // de-selfikácia
// root prestal byť this-self a stal sa argumentom
private BVSTree<E> delete(E k, BVSTree<E> t ) {
    if (t == null )
        return t;
    if (k.compareTo(t.key) < 0)
        t.left = delete(k, t.left);
    else if(k.compareTo(t.key) > 0)
        t.right = delete(k, t.right);
    else if( t.left != null && t.right != null ) {
        t.key = findMin(t.right).key;
        t.right = delete(t.key, t.right);
    } else
        t = (t.left != null) ? t.left : t.right;
    return t;
}
```

// element je v ľavom podstromi
// delete v ľavom podstromi
// element je v pravom podstromi
// delete v pravom podstromi
// je to on, a má oboch synov
// nájdi min.pravého podstromu
// rekurz.zmaž minimum
// pravého podstromu
// ak nemá 2 synov, je to ľahké

Klonovanie

(z istého dôvodu úplne vo vlastnej réžii)



```
interface Clonable { // vlastná analógia clon(e)able
    public Object copy();
}

public class Hruska implements Comparable<Hruska>, Clonable {
    static int allInstances = 0;
    private int instanceIndex;
    private int size;
    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 new Integer(this.size).compareTo(inaHruska.size);
    }
}
```



Klonovanie

(z istého dôvodu úplne vo vlastnej réžii)

```
class BVSTNode<E extends Comparable<E> & Clonable> implements Clonable {
    BVSTNode<E> left, right; E key;
    static int allInstances = 0;
    private int instanceIndex;

    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

(z istého dôvodu úplne vo vlastnej réžii)

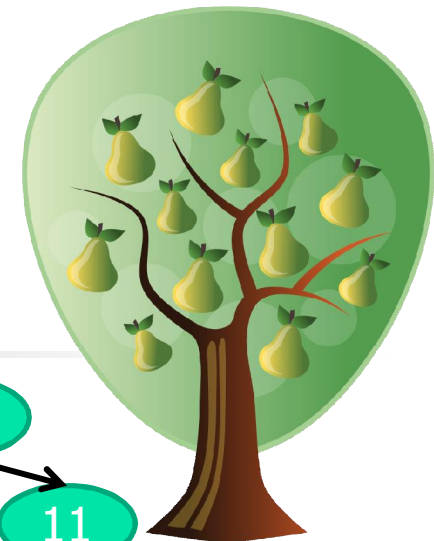
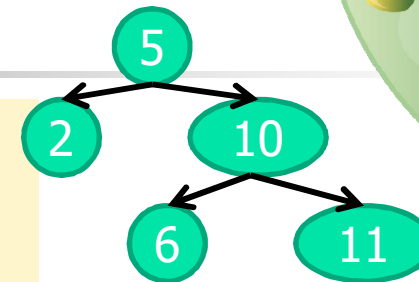
```
class BVSTree<E extends Comparable<E> & Clonable> implements Clonable {  
    BVSTree<E> root;  
    static int allInstances = 0;  
    private int instanceIndex;  
  
    public BVSTree () {  
        instanceIndex = allInstances++;  
        System.out.println("create BVSTree " + instanceIndex);  
        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

```
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
```