Midterm 2013, verzia A Meno a priezvisko: _____
obsahuje 5 príkladov, spolu 29>25 bodov skupina: _____

1A) [8 bodov] Zistite, čo počíta nasledujúca rekurzívna funkcia **foo** pre **n>=0**. Hint: **foo(2013) = 6**.

```
static long foo(long n) {
     if (n > 0)
          return foo(n/10)+(n%10);
     else
          return 0;
}
```

[1 bod] Pre istotu to vyjadrite pár slovami a máte prvý bod:

[1 bod] Vypočítajte **foo(911775039)** = ?

[1 body] Dokážte, alebo vyvráťte nasledujúce tvrdenie: „pre ľubovoľné n >= 0 platí foo(n) <= n“:

☐ Tvrdenie platí a dôkaz je:

☐ Tvrdenie neplatí a kontra-príklad je:

Jadrom úlohy je ale ďalšia funkcia **goo**, ktorá používa **foo** a jedno statické konštatné pole **p**:

```
final static boolean[] p =
          new boolean[] { true,false,false,true,false,false,true,false,false,true };

static boolean goo(long n) {
     if (n < 10)
          return p[(int)n];
     else
          return goo(foo(n));
}
```

[2body] Vyjadrite pár (<=16) slovami, čo počíta rekurzívna funkcia **goo** pre nezáporné argumenty **n>=0**. Pre istotu a overenie vašej hypotézy Vám prezradíme, že **goo(2013)** je **true**. Správna odpoveď je krátka, napr. tvaru "*počet cifier čísla n faktoriál*". Nepopisuje ako rekurzia funguje, to nehodnotíme**:**

[1 bod] Vypočítajte **goo(911775039)** = ?

[2 body] Definujte nerekurzívnu, a čo najelegantnejšiu, verziu **goo** v Jave, ktorá navyše nepoužíva **foo**:

Midterm 2013, verzia B Meno a priezvisko: _____
obsahuje 5 príkladov, spolu 29>25 bodov skupina: _____

1A) [8 bodov] Zistite, čo počíta nasledujúca rekurzívna funkcia **foo** pre **n>=0**.
Hint: **foo(1989,0) = 27** a **foo(2013,7) = 6+7 = 13.**

```
static long foo(long n, long a) {
    if (n > 0)
        return foo(n/10, a+(n%10));
    else
        return a;
    }
```

[2 body] Pre istotu vyjadrite pár slovami, čo je **foo(n, a)** pre **n>=0**:

[1 bod] Vypočítajte **foo(918972645,0)** = ?

Jadrom úlohy je ale ďalšia funkcia **goo**, ktorá používa **foo** a jedno statické konštatné pole **r**:

```
final static boolean[] r =
     new boolean[]{false,true,true,false,true,true,false,true,true,false};

static boolean goo(long n) {
if (n < 10)
    return r[(int)n];
else
    return goo(foo(n, 0));
}
```

[2body] Vyjadrite pár (<=16) slovami, čo počíta rekurzívna funkcia **goo** pre nezáporné argumenty **n>=0**. Pre istotu a overenie vašej hypotézy Vám prezradíme, že **goo(2013)** je **false**. Správna odpoveď je krátka, napr. tvaru "*počet cifier čísla n faktoriál*". Nepopisuje ako rekurzia funguje, to nehodnotíme**:**

[1 bod] Vypočítajte **goo(918972645)** = ?

[2 body] Definujte nerekurzívnu, a čo najelegantnejšiu, verziu **goo** v Jave, ktorá navyše nepoužíva **foo**:

————— riešenia príkladu 1 píšte na tento list (aj z druhej strany) alebo si vypýtajte ďalší list

2) [6 bodov] Binárnu haldu **`HeapVector<E extends Comparable<E>>`** ideme reprezentovať pomocou kolekcie **`Vector<E>,`** čo je flexibilné pole: **`Vector<E> heap = new Vector<E>();`** o **`Vector`**-e vám stačí vedieť, že **`heap.size()`** je veľkosť, **`heap.elementAt(i)`** je indexovanie (t.j. **`heap[i]`**), a **`heap.set(i,value)`** je prepísanie hodnoty na indexe **`i`** (t.j. **`heap[i] = value`**).

V triede **`HeapVector<E extends Comparable<E>>`** definujte nasledujúce metódy:

- **[2 body]** **`public boolean isHeap()`** vráti **`true`**, ak obsah vektora **`heap`** je binárna halda, t.j. pre ľubovoľný index platí, že hodnota **`i`**-teho prvku (rodiča) nie je väčší ako hodnoty jeho synov s indexami **`2*i+1`** a **`2*i+2.`** Riešenie píšte na druhú stranu.
- **[2 body]** **`public void heapify(E elem)`**, ktorá pridá prvok **`elem`** do existujúcej haldy **`heap`** tak, aby ostala binárnou haldou. Tu je kostra, do ktorej môžete doplniť chýbajúci kód, resp. napíšte celú vlastnú metódu (na druhej strane):

```
public void heapify(E elem) {
   heap.add(elem);                  // pridanie elem na koniec vektora heap
   for (int r = heap.size()-1; r > 0;) { // aktuálna veľkosť vektora heap
      int parent = (r-1)/2;              // parent 1->0, 2->0, 3->1, 4->1,...
      int leftChild = 2 * parent + 1;    // leftChild  0->1, 1->3, 2->5, ...
      int rightChild = 2 * parent + 2;   // rightChild 0->2, 1->4, 2->6, ...
      int swap = parent;
// nájdi (väčšie) z detí parenta (left/rightChild) väčšie ako hodnota parenta
// na väčšie z nich bude ukazovať index swap, kód píš priamo sem dole:









      if (swap != parent) {// vymeň hodnoty na indexoch swap, parent, ak treba
                           // kód píš opäť sem dole:









      } else break;
   }
}
```

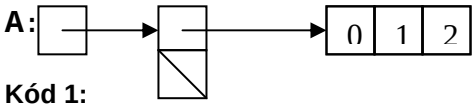
- **[2 body]** **`public BNode<E> toTree()`**, ktorá z binárnej haldy vo vektore **`heap`** vytvorí binárny strom – haldu - obsahujúci vrcholy typu **`BNode<E extends Comparable<E>>:`**

```
public class BNode<E extends Comparable<E>> {
    BNode<E> left;           // ľavý podstrom
    E key;                   // hodnota vrchola
    BNode<E> right;          // pravý podstrom
    public BNode(BNode<E> left, E key, BNode<E>right) {  // konštruktor
      this.left=left; this.key=key; this.right = right;
}
```

——————— riešenia príkladu 2 píšte na tento list alebo si vypýtajte ďalší list

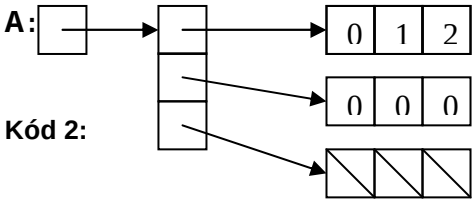
- **3) [5 bodov = 0.5 bodu za každú správnu odpoveď]** Ku každému obrázku napíšte kód, ktorý vytvorí znázornené pole v pamäti. Prečiarknutá bunka prestavuje hodnotu **null**. Ak taký kód neexistuje, napíšte NEEXISTUJE. Nedajte sa nachytať !

Obrázok 1:



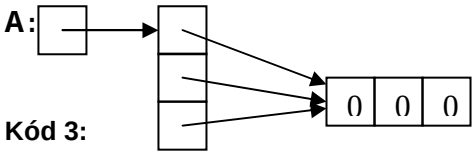
Kód 1:

Obrázok 2:



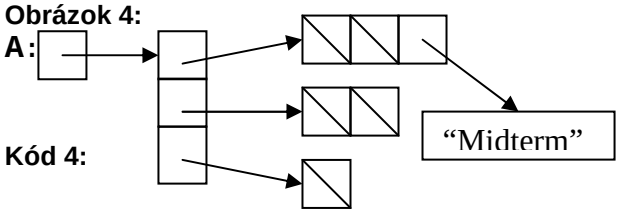
Kód 2:

Obrázok 3:



Kód 3:

Obrázok 4:



Kód 4:

V pravej časti strany **prečiarknite** riadky, ktoré sú:

- syntakticky nesprávne – kompilátor by na nich zahlásil chybu (tie **prečiarknite** a označte S),
- spôsobia chybu počas behu programu (tie označte R).

```
//---------- pole s null-mi
Integer[] pole1 =
       new Integer[]{1,2,null,null};
for(Integer p:pole1) {
  System.out.println(p+p);
}

//---------- priradenie poľa
Integer[] pole3 = { 0 };
Integer[] pole4 = pole3;

pole4 = new Integer[]{1,2};
System.out.println(pole3[1]);

pole3 = pole4;
pole4[1] = 99;

if (pole3[1] == 99)
  System.out.println(pole3[1]);


//---------- dvojrozmerné pole
Integer[][] pole2 = new Integer[3][];
pole2[0] = new Integer[]{1,2,3};
pole2[2] = new Integer[]{4,5,6};

// podtrieda HashSet
class MySet extends HashSet<String> {}

MySet ms = new HashSet<String>();
HashSet<String> hs = new MySet();

// pole podtried
MySet[] pole5 = new HashSet<String>[5];
HashSet<String>[] pole6 = new MySet[6];

// List podtried
ArrayList<MySet> al1 =
       new ArrayList<HashSet<String>>();
ArrayList<HashSet<String>> al2 =
       new ArrayList<MySet>();
```

4) [5 bodov] Nasledujúci kód sa spustí s argumentami predstavujúcimi jednosmerné cestovné poriadky:

```
java Cities BA-ZA-KE BA-NR-BB-KE PO-SK-BJ NR-ZA
```

```
public class Cities {
static TreeMap<String, TreeSet<String>> cudo=new TreeMap<String,
TreeSet<String>>();

public static void main(String[] args) {
   for (String arg : args) {
      String[] edges = arg.split("-");
      String vertex = null;
      for (String v : edges) {                        // sem pride graf…
         if (vertex != null) {
            TreeSet<String> set = cudo.get(vertex);
            if (set == null)
                  set = new TreeSet<String>();
            set.add(v);
            cudo.put(vertex, set);
         }
         vertex = v;
      }
   }
System.out.println(cudo);
```

výpis premennej *cudo* vypíše podľa Vás **[označte správnu možnosť, 2 body]**:

1. {BA=NR,BB=KE,BJ=SK,KE=BB,NR=BA,PO=SK,SK=BJ,ZA=BA}
2. {PO=[SK],SK=[BJ],ZA=[KE],BA=[ZA,NR],BB=[KE],NR=[ZA,BB]}
3. {BA=[ZA,NR],BB=[KE],NR=[ZA,BB],PO=[SK],SK=[BJ],ZA=[KE]}
4. {0=[NR,ZA],1=[KE,NR],2=[SK],3=[BB,ZA],4=[BA,BB],5=[SK],6=[BJ,PO],7=[BA,KE]}
5. {BA=[NR,ZA],BB=[KE],NR=[BB,ZA],PO=[SK],SK=[BJ],ZA=[KE]}
6. {BA=[NR,ZA],BB=[KE,NR],BJ=[SK],KE=[BB,ZA],NR=[BA,BB],PO=[SK],SK=[BJ,PO],ZA=[BA,KE]}
7. {BB=[NR],BJ=[SK],KE=[BB,ZA],NR=[BA],SK=[PO],ZA=[BA]}
8. niečo ine:

Napíšte kus kódu, ktorý nájde všetky mestá, odkiaľ nevedie žiaden spoj, v uvedenom príklade sú to KE a BJ **[3 body]**

——— riešenia príkladu 4 píšte na tento list, aj zozadu, alebo si vypýtajte ďalší list

5) [5 bodov] Multi-množina je sťaby množina, v ktorej sa môžu opakovať prvky viackrát. Nezáleží na ich poradí. Kým v množine sa prvok nachádza alebo nie, v multi-množine sa nachádza n-krát (n>0), alebo nenachádza. Navrhnite a zrealizujte implementáciu multi-množiny s nasledujúcim interface:

```
public interface MultiSet<E> {
public void add(E elem);          // pridanie prvku do multi-množiny [2 body]
public int count(E elem);         // počet výskytov prvku v multi-množine [1 bod]
public boolean subset(MultiSet<E> ms);  // this je obsiahnutá v multi-množine ms [2 b]
}
```

Kód, ktorý musí zbehnúť:

```
HashMultiSet<Integer> ms1 = new HashMultiSet<Integer>();
ms1.add(1);ms1.add(3);ms1.add(2);ms1.add(1);ms1.add(3);ms1.add(4);// ms1={1,3,2,1,3,4}
for(int i=0; i<5; i++)
  System.out.println(i+" je tam "+ms1.count(i)+".krat");
HashMultiSet<Integer> ms2 = new HashMultiSet<Integer>();
ms2.add(1); ms2.add(3); ms2.add(2); // ms2 = { 1,2,3 }
System.out.println(ms2.subset(ms1)); // true
ms2.add(2);     // ms2 = { 1,2,3,2 }
System.out.println(ms2.subset(ms1)); // false
```

? treba definovat submutiset ?

——— riešenia príkladu 5 píšte na tento list, aj zozadu, alebo si vypýtajte ďalší list