Programozási nyelvek II.: JAVA, 4. gyakorlat

2017. október 2-6.

A 4. gyakorlat tematikája

- Tömbök
- `A java.util.Arrays osztály`
- A String osztály
- A StringBuffer és a StringBuilder osztály

Tömbök

- Sok azonos típusú érték tárolására
- Hatékony elérés: indexelés
  - minden tömbelem egyenlő méretű helyet foglal el
  - a tömb elemei folytonosan helyezkednek el a memóriában, az elemek helyének kiszámítása ugyanazzal a formulával, pl. egydimenziós tömb esetén: `t[i] = t + i * sizeof(T)`, ahol a `t` a tömb, az `[i]` az indexelő operátor, amelyben `i` az elérendő elem indexe, a `sizeof()` operátor egy típus értékeinek méretét adja meg byte-ban, a `T` pedig a tömb elemeinek típusa
  - a tömb nevének hivatkozása a tömb kezdőcímét adja meg a memóriában
  - a tömböket 0–tól indexelik, hossz–1–ig
  - hibás index megadása: `ArrayIndexOutOfBoundsException` kivétel
- Időigényes a beszúrás és a törlés

Tömb létrehozása

- Minden T típushoz hozzárendelhető egy T[] típus, amely a T elemekből képzett tömböt jelenti
- Tömb típusú változó definíciója `int[] intArray;`, `char[] charArray;`, `String[] stringArray;`, vagy `int intArray[];`, `char charArray[];`, `String stringArray[];`
- A változó deklarációja nem hozza létre a tömböt
- A tömböket az objektumokhoz hasonló módon példányosítani kell, pl. `int[] t = new int[10];`, `int t[] = new int[10];` (a hossz megadása a tömb létrehozásakor, a hosszt megváltoztatni nem lehet)
- A példányosítás elmulasztásával: `NullPointerException` kivétel, pl.: `int[] s; int x = s[0];` Hibás!

Tömbök inicializálása

- `boolean[] barr1 = { true, false };`
- `boolean[] barr2 = new boolean[] { true, false };`
- `boolean[] barr3 = new boolean[2]; barr3[0] = true; barr3[1] = false;`

Többdimenziós tömbök

- tömbök tömbje
- deklaráció: `int[][] mx;`
- inicializálásnál az első dimenziót meg kell adni, vagyis `int[][] mx = new int[5][];`
- szabálytalan alakú tömb:
  - `int mdt[][];`
  - `mdt = new int[2][];`
  - `mdt[0] = new int[2];`
  - `mdt[0][0] = 7;`
  - `mdt[0][1] = 2;`
  - `mdt[1] = new int[3];`
  - `mdt[1][0] = 2;`
  - `mdt[1][1] = 4;`
  - `mdt[1][3] = 0;`

A `java.util.Arrays` osztály

- a tömbökhöz tartozó segédosztály
- számos, a tömbökkel kapcsolatos műveletet meg lehet találni a java.util.Arrays osztályban (pl. a szöveggé alakításának (toString()), bináris keresésnek (binarySearch()), vagy a tömb feltöltésének (fill()) műveletét)
- tömbök összehasonlítása: `equals`, nem pedig az == operátorral

For ciklus tömbökre

enhanced for loop

```
class EnhancedForDemo {
    public static void main(String[] args) {
        int[] numbers = {1,2,3,4,5,6,7,8,9,10};
        for (int item : numbers) {
            System.out.println("Az elem erteke: " + item);
        }
    }
}
```

Aliasing

- több referencián keresztül hivatkozunk ugyanarra az objektumra

```
Rectangle box1 = new Rectangle (0, 0, 100, 200);
Rectangle box2 = box1;
```

- tömbök esetében problémát okozhat, pl. tömbok megfordításánál

```
void reverse( int[] src, int[] dst ){
    for( int i=0, j=src.length-1;
            i<src.length; ++i, --j ){
        dst[j] = src[i];
    }
}

int[] t = {1,2,3,4,5};
int[] t = new int[5]{1,2,3,4,5};

reverse(t,t)
```

Aliasing – egy megoldás

```
void reverse( int[] src, int[] dst ){
    assert src != null;
    assert dst != null;
    assert src.length == dst.length;
    assert src != dst;
        // ezt is kossuk ki, hogy jol mukodjon
    for( int i=0, j=src.length-1;
        i<src.length; ++i, --j ){
        dst[j] = src[i];
    }
}
```

Kiszivárogtatás

```
public class Point {
    private final int x,y;
    public Point( int x, int y ){
        this.x = x;
        this.y = y;
    }
    public int getX(){ return x; }
    public int getY(){ return y; }
}
```

Kiszivárogtatás

```
public class Point {
    private final int[] coords;
    public Point( int x, int y ){
        coords = new int[]{x, y};
    }
    public int getX(){ return coords[0]; }
    public int getY(){ return coords[1]; }
    public int[] coords(){ return coords; }
}

Point p = new Point(1,1);
int[] c = p.coords();
```

Kiszivárogtatás

- a `coords()` metódus engedi kiszökni a `Point` belső állapotát
- amin keresztül direkt manipulálható kívülről a belső állapot
- sérti az OOP elveket (private)

Kiszivárogtatás – egy megoldás

```
public class Point {
    private final int[] coords;
    public Point( int x, int y ){
        coords = new int[]{x, y};
    }
    public int getX(){ return coords[0]; }
    public int getY(){ return coords[1]; }
    public int[] coords()
        { return new int[]{coords[0],coords[1]}; }
    // masolatot adunk vissza
}
```

A `String` osztály

- Unicode karakterek sorozata
- pl. `String s = "Szia!";`, `String s = new String("Szia!");`
- Egy String objektum tartalmát nem lehet módosítani, helyette új stringet kell létrehozni.
- pl. `s = "Szia?";`, `s = new String("Szia?");`
- Műveletei: `length`, `charAt`, `compareTo`, `concat`, `endsWith`, `replace`, `substring`, `trim`, `valueOf`, `indexOf`, `equalsIgnoreCase`, `toLowerCase`, `...`

A StringBuffer és a StringBuilder osztály

- Unicode karakterek sorozata
- A tartalmuk megváltoztatható anélkül, hogy új objektumot hozunk létre
- Műveletei: `append`, `insert`, `reverse`, `setCharAt`, `setLength`, ...

Feladat: Több-dimenziós tömb sorainak kiiratása

Készíts egy IntegerMatrix nevű osztályt a következő metódusokkal.

- Egy konstruktor, mely 3 paramétert vár:

  `int rowNum` A mátrix sorainak száma.
  `int colNum` A mátrix oszlopainak száma.
  `Integer[] linearData` Egy, a mátrix elemeit sorfolytonosan tároló tömb.

- Egy objektumszintű `toString` metódus, mely egyetlen karakterláncba felsorolja a mátrix elemeit. A karakterláncban az egy sorban szereplő elemeket a `,` karakterrel válaszd el! A sorokat a `;` karakterrel válaszd el!

- Például `linearData = {1,2,3,4,5,6}` esetén az `IntegerMatrix(2,3,linearData)` konstruktorhívás hatására a következő mátrix készül:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

  Ez esetben objektum `toString` metódusa a következő karakterlánccal tér vissza: `"1,2,3;4,5,6"`.

Feladat: Több-dimenziós tömb sorainak kiiratása

```
public class IntegerMatrixTest {
  public static void main(String[] args){
    Integer[] linearData = {1, 2, 3, 4, 5, 6};
    System.out.println(new IntegerMatrix(2,3, linearData));
  }
}
```

Feladat: Több-dimenziós tömb sorainak kiiratása

```
public class IntegerMatrix {
  private int rowNum;
  private int colNum;
  private Integer[][] data;

  public IntegerMatrix(int rowNum, int colNum,
                       Integer[] linearData){
    this.rowNum = rowNum;
    this.colNum = colNum;
    data = new Integer[rowNum][colNum];

    for(int i = 0; i < linearData.length; i++) {
      int row = (int) Math.floor(i / colNum);
      int col = i % colNum;
      data[row][col] = linearData[i];
    }
  }
  [...]
}
```

Feladat: Több-dimenziós tömb sorainak kiiratása

A `toString` metódus.

Probléma A szeparátor (`,` vagy `;`) az elemek közé kell, hogy kerüljön.

Ötlet Ez ugyanaz, mintha az első elem kivételével minden elem elé tennénk szeparátort. Dolgozzuk fel külön az első elemet.

Probléma Különböző szeparátort kell írnunk a sorok és az elemek közé.

Ötlet Használjunk beágyazott ciklusokat! A ciklustörzsben mindig a sorok és elemek elé konkatenáljuk a megfelelő szeparátort (kivéve az első sort és a sorok első elemeit).

Feladat: Több-dimenziós tömb sorainak kiiratása

```
public class IntegerMatrix {
  [...]
  public String toString(){
    String s = "";

    if(data.length < 1) return s;
    s += data[0].length < 1 ? "" : data[0][0];
    for(int j=1; j<data[0].length; j++)
      s += "," + data[0][j];

    for(int i=1; i<data.length; i++){
      s += ";";
      s += data[i].length < 1 ? "" : data[i][0];
      for(int j=1; j<data[i].length; j++)
        s += "," + data[i][j];
    }
    return s;
  }
}
```

Feladat: Több-dimenziós tömb sorainak kiiratása

A `toString` metódus.

Probléma Nehezen olvasható, ráadásul a feltétel-kiértékelések nem hatékonyak.

Ötlet Kezdetben a szeparátor legyen az üres szó. Az első sor vagy elem kiértékelése után írjuk felül a megfelelő szeparátorral. (Az értékadás hatékonyabb, mint a feltétel-kiértékelés.)

Feladat: Több-dimenziós tömb sorainak kiiratása

```
public class IntegerMatrix {
  [...]
  public String toString(){
    String s="",rowDelim="",colDelim="";

    for(Integer[] row : data){
      s += rowDelim;
      rowDelim = ";";
      for(Integer elem :row){
        s += colDelim;
        colDelim = ",";
        s += elem;
      }
      colDelim = "";
    }
    return s;
  }
}
```

Feladat: Több-dimenziós tömb sorainak kiiratása

A `toString` metódus.

Probléma A `String` immutábilis. Konkatenáláskor (+) a operandusok lemásolásával új `String` készül, melynek hossza az operandusok hosszának összege. A ciklusban való konkatenálások költsége így $O(n^2)$. Két egymásba ágyazott ciklus esetén már $O(n^3)$.

Ötlet Használjuk a `java.lang.StringBuilder` osztályt! Ennek `append` metódusát a hatékony konkatenálásra vezették be. `https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html`

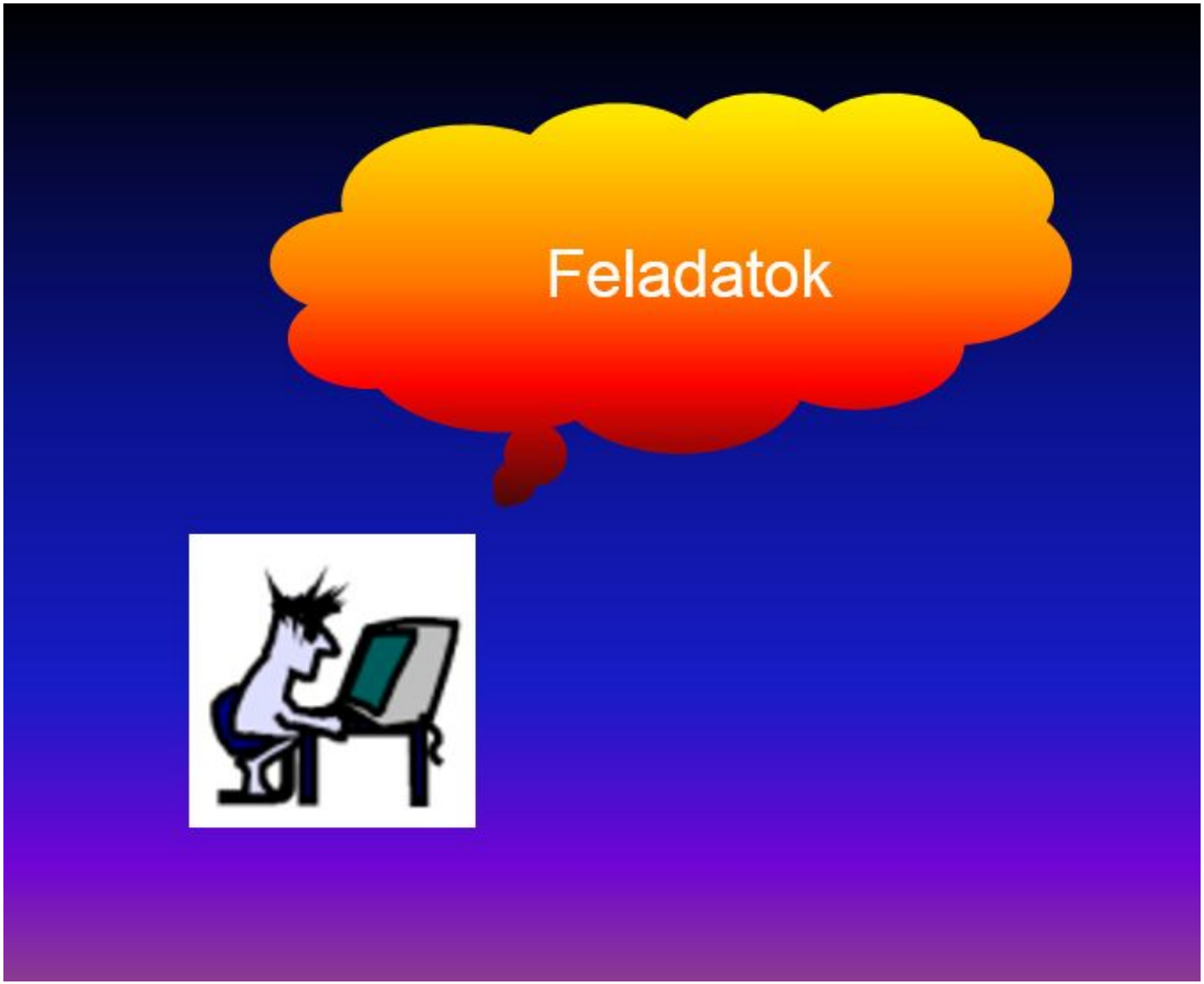
Feladat: Több-dimenziós tömb sorainak kiiratása

```
public class IntegerMatrix {
  [...]
  public String toString(){
    StringBuilder sb = new StringBuilder();
    String rowDelim="", colDelim="";

    for(Integer[] row : data){
      sb.append(rowDelim);
      rowDelim = ";";
      for(Integer elem : row){
        sb.append(colDelim);
        colDelim = ",";
        sb.append(elem);
      }
      colDelim = "";
    }
    return sb.toString();
  }
}
```

Java Platform SE 8 API

`http://docs.oracle.com/javase/8/docs/api/`


Feladatok

Hibajavítás (IntVector.java, IntVectorDemo.java)

Javítsuk ki a HIBÁS programo(ka)t!

- Készítsünk a util csomagon belül egy `IntVector` osztályt, amely egészek sorozatát ábrázolja!
- Legyen egy tömb adattagja, amely a sorozatot tárolja.
- Adjunk az osztályhoz egy konstruktort, amely egy egészekből álló tömböt vár paraméterül! (Segítség: ügyeljünk, hogy a belső állapotot ne szivárogtassuk ki!)
- Vegyünk fel egy `add()` metódust, mely a sorozat minden eleméhez hozzáad egy paraméterül kapott egész számot!
- Készítsünk egy `toString()` metódust is, mely felsorolja a számokat szóközzel elválasztva. Például: `[1 2 3]`

Vektor osztály (Vector.java, VectorTest.java)

Készítsünk egy `utils.Vector` osztályt (valós számokat tartalmazó tömb mint vektor segítségével), amelynek a következő műveletei vannak: két vektor skaláris szorzatának, összegének, különbségének, a vektor euklideszi normájának, vektor skalárral való szorzatának kiszámítása, valamint a vektor sztringként történő ábrázolása (`java.lang.StringBuilder`–t használjunk a szöveg előállításához). Készítsünk főprogramot is, amely teszteli ezen műveleteket!

Vektor osztály (VectorAL.java, VectorTestAL.java, VectorLL.java, VectorTestLL.java)

Készítsük el az előbbi osztálynak azon változatát, amelyben a vektort valós számokat tartalmazó

- tömbös lista (`java.util.ArrayList`),
- láncolt lista (`java.util.LinkedList`)

segítségével valósítja meg!

Számológép osztály (CalculatorVector.java, CalculatorVectorAL.java, CalculatorVectorLL.java)

Készítsünk számológépet és tegyük képessé vektorokon végezhető műveletek elvégzésére! A program három parancssori paramétert vár: az első és a második paraméterben számok vannak vesszővel elválasztva, a harmadik paraméter pedig egy szám (pl. `java CalculatorVector 2.0,3.0,4.0 3.4,5.6,1.2 2.0`). Ellenőrizzük, hogy megfelelő számú paramétert kaptunk–e! Ha igen, akkor feltehetjük, hogy a paraméterek valóban számok.

Mátrix osztály (Vector.java, Matrix.java, MatrixTest.java)

Készítsünk egy `basics.Matrix` osztályt (valós számokat tartalmazó kétdimenziós tömb mint mátrix segítségével), amelynek a következő műveletei vannak: $N \times N$ dimenziós egységmátrix létrehozása, $M \times N$ dimenziós véletlen mátrix létrehozása, mátrix transzponáltjának, két mátrix szorzatának, összegének, különbségének kiszámítása, mátrix–vektor szorzás (ehhez használjuk a létrehozott vektor osztályt), valamint a mátrix sztringként történő ábrázolása (`java.lang.StringBuilder`–t használjunk a szöveg előállításához). Készítsünk főprogramot (`MatrixTest.java`, amelyet rakjunk a main csomagba) is, amely teszteli ezen műveleteket!