

# Programozási nyelvek II.: JAVA

## 6. gyakorlat

2017. október 16-20.

# Az 6. gyakorlat tematikája

- Túlterhelés - összefoglalás
- statikus adattagok és metódusok - összefoglalás
- default konstruktor -összefoglalás
- Filefeldolgozás - példa
- Enum

- egy osztályban több művelet ugyanazzal a névvel, de különböző paraméterezéssel
- különböző számú formális paraméter, pl. ArrayList: `void add( elem )`  
`void add( elem, index )`
- különböző deklarált típusú (különböző "statikus" típusú) paraméterek

# Egy sokszor látott példa

```
System.out.println(int);  
System.out.println(double);  
System.out.println(String);
```

# Túlterhelés - Konstruktor túlterhelése

```
public class Point {
    private int x, y;
    public Point( int x, int y ){
        this.x = x;
        this.y = y;
    }
    public Point(){
        x = 0;  y = 0;    // amugy is 0
    }
}

public class Point {
    private int x, y;
    public Point( int x, int y ){
        this.x = x;
        this.y = y;
    }
    public Point(){
        this(0,0);
    }
}
```

- ❶ paraméter nélküli
- ❷ automatikusan generált - amennyiben nincs konstruktor megadva
- ❸ alapértelmezett értékek:
  - ❶ Primitív típusoknál *0* / *'u0000'* / *false*
  - ❷ Referencia: *null*

# Default konstruktor példa 1.

```
public class Circle {  
    private int x, y;  
    private double radius;  
    public static void main(String[] args){  
        Circle c = new Circle(); // x:0; y:0, radius: 0.0  
    }  
}
```

```
public class Circle {  
    private int x = 0, y = 0;  
    private double radius = 1.0;    // inicializáló kifejezés  
    public static void main(String[] args){  
        Circle c = new Circle(); // x:0; y:0, radius: 1.0  
    }  
}
```

- 1 Addig léteznek, amíg a programozó nem definiál saját konstruktor.
- 2 Az alábbi kód már nem fordul le:

```
public class Circle {  
    private int x, y;  
    private double radius;  
    public Circle(int x,int, int r){/*...*/}  
    public static void main(String[] args){  
        //compiler error:  
        //constructor Circle in class Circle cannot be applied  
        to given types;  
        Circle c = new Circle();  
    }  
}
```



- ➊ Más néven osztályszintű tagok, azaz az egész osztályhoz, nem annak egy adott példányához tartoznak;
- ➋ A statikus változók az osztály betöltésekor(első használatakor) jönnek létre és inicializálódnak
- ➌ A statikus metódusokra ezekből következően vonatkozik két egyszerű szabály:
  - ➊ Csak statikus adattagokra hivatkozhatnak
  - ➋ Csak statikus metódusokra hivatkozhatnak

# Példa statikus adattagra

```
public class Circle {  
    public static int counter = 0;  
    private int x, y;  
    private double radius;  
    public Circle(){ counter++; /*...*/}  
    public static void main(String[] args){  
        System.out.println(Circle.counter);  
        Circle c1 = new Circle();  
        System.out.println(Circle.counter);  
        Circle c2= new Circle();  
        System.out.println(Circle.counter);  
    }  
}
```

Kimenet:

0  
1  
2

# Példas statikus metódusokra

```
public class Circle {  
    public static int counter = 0;  
    private int x, y;    private double radius;  
    private static void incrementCounter(){counter++;}  
    public Circle(String sx, String sy, String sradius){  
        incrementCounter();  
        x = Integer.parseInt(sx);  
    }  
    public static void main(String[] args){  
        incrementCounter();  
        System.out.println(Circle.counter);  
        Circle c1 = new Circle("1","2","3");  
        System.out.println(Circle.counter);  
    }  
}
```

Kimenet:

1  
2

# Statikus tagok - némi konvenció

- 1 konvenció szerint a statikus tagokhoz a jobb olvashatóság érdekében az osztály nevével, nem az objektumreferencián keresztül hivatkozunk (bár ezt a nyelv megengedi..)

```
public class Main{
    public static void main(String[] args){
        Circle.incrementCounter(); // -> OK
        System.out.println(Circle.counter); //->OK
        Circle c1 = new Circle("1","2","3");
        c1.incrementCounter(); // ->NEM OK !!
        System.out.println(c1.counter); // ->NEM OK !!
    }
}

class Circle {
    public static int counter = 0;
    /*...*/
    public static void incrementCounter(){counter++;}
}
```

# Enum - Felsorolási típus

- 1 A java enum típus egy speciális típus ami lehetővé teszi olyan változók használatát, melyek csak valamely előre konstans definiált értéket vehetnek fel
- 2 Az egyik általánosan használt példa a hét napjait reprezentáló osztály:
- 3 *valueOf(String)* - statikus metódus, a paraméterként kapott névvel egyező enumobjektummal tér vissza (ha nincs: *null*)

```
public enum Day {  
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
    THURSDAY, FRIDAY, SATURDAY  
}  
//...  
Day d1 = Day.MONDAY;  
Day d2 = Day.valueOf("MONDAY");
```

- 4 Általánosságban elmondható, hogy tanácsos minden olyan esetben enumokat használni, amikor konstansok egy meghatározott halmazával dolgozunk, vagyis ahol egy változó által felvehető minden lehetséges értéket ismerünk.

# Enum - Példa használatra 1.

```
public class EnumTest {
    Day day;
    public EnumTest(Day day) {    this.day = day; }
    public void tellItLikeItIs() {
        switch (day) {
            case MONDAY:
                System.out.println("Mondays are bad.");
                break;
            case FRIDAY:
                System.out.println("Fridays are better.");
                break;
            case SATURDAY: case SUNDAY:
                System.out.println("Weekends are best.");
                break;
            default:
                System.out.println("Midweek days are so-so.");
                break;
        }
    }
    //....
}
```

## Enum - Példa használatra 2.

```
//....  
public static void main(String[] args) {  
    EnumTest firstDay = new EnumTest(Day.MONDAY);  
    firstDay.tellItLikeItIs();  
    EnumTest thirdDay = new EnumTest(Day.WEDNESDAY);  
    thirdDay.tellItLikeItIs();  
    EnumTest fifthDay = new EnumTest(Day.FRIDAY);  
    fifthDay.tellItLikeItIs();  
    EnumTest sixthDay = new EnumTest(Day.SATURDAY);  
    sixthDay.tellItLikeItIs();  
    EnumTest seventhDay = new EnumTest(Day.SUNDAY);  
    seventhDay.tellItLikeItIs();  
}  
}
```

# Enum mint osztály - kicsit haladóbb példa

- 1 A Javában az enumok speciális osztályt jelenítenek meg, és mint Java osztályok tartalmazhatnak metódusokat és adattagokat is.
- 2 Emellett az enumnal definiált osztályokhoz a fordító egy sor metódust generál nekünk, pl: *values()* statikus metódus, mely visszaadja az összes enumértéket, vagy *ordinal()* amely az adott enum érték az osztálydefinícióban való helyzetét adja meg
- 3 Hogy miért is jók nekünk ezek, ezt illusztrálnánk a következő bolygós példával.
- 4 Szeretnénk létrehozni egy bolygókat felsoroló típust, mely tartalmaz bizonyos, az adott bolygót jellemző értékeket, mint például bolygó tömege és sugara, valamint ezen alaptulajdonságokból származó lekérdezéseket írni, mint hogy mennyi a gravitáció a planéta felületén. Ahhoz hogy ezt ki tudjuk számolni szükségünk lesz az általános gravitációs állandóra is (ez minden bolygó esetén ugyanaz). Végül az egészből szeretnénk írni egy tesztprogramot, mely kiszámolja valakinek a súlyát az összes bolygón.



# Enum mint osztály - kicsit haladóbb példa (folyt.)

- 1 A vonatkozó információk eltárolására létrehozhatunk példányváltozókat, illetve egy osztályváltozót a gravitációs állandónak.
- 2 Ezen példányváltozók értékei általában az adott objektum tulajdonságait írják le, ezért nem szeretnénk változtatni őket, és ezeket a létrehozás pillanatában tudjuk, így érdemes lehet konstruktorral beállítani őket:

```
public enum Planet {  
    VENUS    (4.869e+24, 6.0518e6),  
    EARTH    (5.976e+24, 6.37814e6),  
    //...  
    private final double mass;    // in kilograms  
    private final double radius; // in meters  
    Planet(double mass, double radius)  
        { this.mass = mass; this.radius = radius;}  
    public static final double G = 6.67300E-11;  
    double surfaceGravity()  
        { return G * mass / (radius * radius);    }  
    //...
```

# Enum mint osztály - kicsit haladóbb példa (folyt.)

- ❶ a *main* metódus a *values()* és *ordinal()* használatával

```
//..  
double surfaceWeight(double otherMass) {  
    return otherMass * surfaceGravity();  
}  
  
public static void main(String[] args) {  
    if (args.length != 1) {  
        System.exit(-1);  
    }  
    double earthWeight = Double.parseDouble(args[0]);  
    double mass = earthWeight/EARTH.surfaceGravity();  
    for (Planet p : Planet.values())  
        System.out.printf("Your weight on %s  
        (the %d th planet) is %f%n",  
        p, p.ordinal()+1,p.surfaceWeight(mass));  
}
```

# Feladatok



Keressük meg a hibákat a következő programban! **LETÖLT**

Feladat:

- A program parancssori argumentumként (`args[0]`) egy könyvtár- vagy fájlnevet vár.
- Ezután
  - 1 standard input-ról (`stdin`) beolvas egy szót.
  - 2 bejárja rekurzívan az `args[0]`-beli könyvtárat\* és minden fájl tartalmát kiírja az `output.txt`-be.
  - 3 bejárja rekurzívan az `args[0]`-beli könyvtárat\* és eldönti, hogy van-e olyan fájl, mely tartalmazza az `stdin`-en beolvasott szót. Az eredményt kiírja az `output.txt`-be

\*Ha `args[0]` egy fájl, akkor értelemszerűen csak erre az egy fájlra végzi el a fenti műveleteket.

## Megvalósítás:

- ❶ A Finder osztály néhány fájlműveletet tartalmaz:
  - ❶ a konstruktor egy fájlnevet vár, amellyel példányosít egy PrintWriter objektumot melynek segítségével kiírja az eredményeit
  - ❷ a findText túlterhelt metódusok egy paraméterként adott szöveget keresnek rekurzívan, a szintén paraméterként kapott nevű fájlban (rekurzív - ha könyvtárral találkozik belenéz az abban található fájllokba)
  - ❸ a printFile túlterhelt metódusok egy paraméterként adott File tartalmát írja a PrintWriter objektumba, szintén rekurzívan. (rekurzív - ha könyvtárral találkozik belenéz az abban található fájllokba)
- ❷ A Main.java példányosítja a Finder osztályt, majd a paraméterként kapott fájl/könyvtár tartalmát átírja egy output.txt állományba első lépésben, másodikban pedig végignézi azt hogy a standard inputról beolvasott szöveg megtalálható-e benne.

# Feladat 1 (Color.java)

Készítsünk egy egyszerű Color felsoroló típust, mely a következő értékeket tárolhatja: RED, GREEN, BLUE.

## Feladat 2 (Auto.java)

- 1 Írjunk egy Auto osztályt, mely a következő tagokat tárolja objektumonként:
  - 1 rendszám (String)
  - 2 szín (Color)
  - 3 maximális sebesség (int)
- 2 Az osztályban legyen számláló, mely tárolja, hogy hány objektumot hoztunk eddig létre .
- 3 Írjunk az osztályhoz egy konstruktort, mely ezt a három értéket várja paraméterként, és a megfelelőket beállítja az új objektumon.
- 4 Írjunk egy paraméter nélküli konstruktort is mely 'AAA-000', 'BLUE' és '120' értékekkel hoz létre objektumot
- 5 Írjunk egy osztályszintű összehasonlító metódust, mely két autó objektumot vár, is igazzal tért vissza, ha az első gyorsabb mint a második

## Feladat 3 (Main.java)

- 1 Helyezzük a Color osztályt az auto.utils csomagba, az Autót pedig az auto csomagba!
- 2 Hozzunk létre egy input.txt fájlt, melyben autók adatai vannak soronként megadva, vesszővel elválasztva. Pl:  
"ABC-123","RED","100"
- 3 Írjunk egy Main osztályt (a csomagokon kívül), amely a tesztprogramunkat fogja tartalmazni!
- 4 A Main osztály main metódusában olvassuk be az input fájl sorait, a létrehozott objektumokat pedig tároljuk el egy ArrayListbe!
- 5 segítség: használhatjuk a java.util.Scannert, illetve az Enum.valueOf(String) metódust