



Programozási technológia

Bevezetés

Dr. Szendrei Rudolf
ELTE Informatikai Kar
2018.

Információk

- ▶ Képzés
 - ▶ Programtervező Informatikus BSc, nappali tagozat, C szakirány
- ▶ Tárgykód:
 - ▶ IP-17cPROGT1EG
- ▶ Előfeltétel (erős):
 - ▶ Programozás tárgy (-2018)
 - ▶ Objektumelvű programozás (2019-)
- ▶ Kreditszám: 5
 - ▶ 2 óra előadás
 - ▶ 2 óra gyakorlat
 - ▶ 1 óra konzultáció
- ▶ Cél:
 - ▶ Eseményvezérelt alkalmazások készítése JAVA nyelven

Információk

Számonkérés

- Gyakorlati jegy alapján megszerezhető vizsgajegy (összevont számonkérés)
- 3 db beadandó feladat bemutatása dokumentációval, határidőre
 - Csak a kitűzött feladatot megoldó, önállóan megvalósított, hibátlanul működő, letesztelt program fogadható el.

Információk

Elérhetőségek:

- ▶ Honlap:
 - ▶ <https://swap.web.elte.hu/>
- ▶ E-mail:
 - ▶ swap@inf.elte.hu
- ▶ Személyesen:
 - ▶ Déli épület, 2.602

Objektumorientált tervezés

Ismétlés

- Nem a funkciókból, tevékenységekből indulunk ki, hanem az adatokból, a feladatban részt vevő elemekből.
- Ezeket azonosítjuk, csoportosítjuk, felderítjük a kapcsolataikat, felelősségeiket. Így jönnek létre az objektumok, illetve az osztályok.
- A rendszer funkcionalitását az egymással együttműködő objektumok összessége adja ki. Egy objektum csak egy jól meghatározott részért felelős.
- Az objektumok adatot tárolnak, ezek kezeléséért felelősek, de ezeket elrejthetik a külvilág elől. Szabványos módon lehet az objektumokkal kapcsolatba lépni.

Feladat

- Készítsünk egy alkalmazottak tárolására használható osztályt.
- Az alkalmazottakról a következőket tudjuk:
 - vezetéknév,
 - keresztnév,
 - beosztás,
 - fizetés.
- A szükséges lekérdező műveleteken kívül legyen lehetőség az alkalmazott fizetésének növelésére.

Megoldás

```
package company;

public class Employee {

    private String firstName, lastName;

    private String job;

    private int    salary;

    ...

}
```

- Változó láthatósága más osztályok felé:
 - `public` mindenki számára látható kívülről
 - `protected` csak a származtatott osztályaiban látható
 - `private` csak ebben az osztályban látható

Megoldás

```
package company;

public class Employee {

    private String firstName, lastName;

    private String job;

    private int salary;

    ...

}
```

- Osztály láthatósága más csomagban lévő osztályok számára:
 - `public` bárhol elérhető az osztály
 - `-` csak a saját csomagjában lévő osztályok számára látható (`package private`)

Megoldás

```
package company;

public class Employee {

    ...

    public String getFirstName () { return firstName; }
    public String getLastName () { return lastName; }
    public int    getSalary () { return salary; }
    public String getJob () { return job; }

    ...

}
```

- `boolean` getter metódus neve: `is` + változónév
- Egyéb getter metódus neve: `get` + változónév
- Setter metódus neve: `set` + változónév

Megoldás

```
public class Employee {  
    ...  
    public Employee(String firstName, String lastName,  
                    int salary, String job) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.salary = salary;  
        this.job = job;  
    }  
}
```

- Java-ban nincsenek pointerok, csak referenciák. Az objektum a `this`-el hivatkozhat önmagára.

Megoldás

```
public class Employee {  
    ...  
    public void raiseSalary(int percent) {  
        salary = (int) (salary * (1.0 + percent / 100.0));  
    }  
    ...  
}
```

- Ha egy típus konverziónál információ veszteség léphet fel, akkor ott mindig kényszerítenünk kell a konverziót.

Java projektek felépítése

- A Java projektek hierarchiája:
 - Csomagok
 - Osztályok, interfészek, felsorolások
- Más csomagban található kódot az `import` utasítással tudunk elérhetővé tenni
- Az osztály nevének meg kell egyeznie a tartalmazó fájl nevével (kivéve beágyazott osztály esetében).
- Minden általunk deklarált adattípus, adattag, illetve metódus előtt feltüntetjük annak láthatóságát.
- Az adattagokat általában elrejtjük a külvilág elől, és csak beállító/lekérdező (setter/getter) metódusokon keresztül engedjük azokat használni.

Feladat

- Készítsünk egy főprogramot (main), és használjuk fel az alkalmazottak osztályt úgy, hogy a konzolról beolvasott adatokkal példányosítjuk, majd pedig kiírjuk a tulajdonságait a konzolra

Megoldás

```
package company;
import java.util.Scanner;
public class EmployeeTester {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Vezetéknév: ");
        String firstName = sc.nextLine();
        System.out.print("Keresztnév: ");
        String lastName = sc.nextLine();
        System.out.print("Beosztás: ");
        String job = sc.nextLine();
        System.out.print("Fizetés: ");
        int salary = sc.nextInt();
        Employee e = new Employee(firstName, lastName, salary, job);
        ...
    }
}
```

Megoldás

- A program belépési pontja egy osztályának az alábbi statikus main metódusa lehet:

```
public static void main(String[] args) {...}
```

- A statikus metódusok az osztályhoz tartoznak, így csak az osztály statikus tagjait érhetik el!
- Java nyelvben:
 - Konzol bemenet: `System.in`
 - Konzol kimenet: `System.out`
- Kiírás konzolra: `print` metódus változataival
- Beolvasás konzolról: `Scanner` objektum megfelelő `next...()` metódusát meghívva.

Megoldás javítása

- A megoldás meglehetősen sok kódismétlést tartalmaz, amit érdemes függvényekbe szervezéssel csökkentenünk. Vezessük be az alábbi két metódust.

```
public static String readString(Scanner sc, String msg){  
    System.out.print(msg);  
    return sc.nextLine();  
}
```

```
public static int readInt(Scanner sc, String msg){  
    System.out.print(msg);  
    int i = sc.nextInt(); // Sor végi ENTER-t pufferben hagyja!  
    sc.nextLine();       // Sorvég jel eltávolítása pufferből  
    return i;  
}
```


Megoldás függvényekkel

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String firstName = readString(sc, "Vezetéknév: ");
    String lastName  = readString(sc, "Keresztnév: ");
    String job       = readString(sc, "Beosztás: ");
    int    salary    = readInt(sc,    "Fizetés: ");
    int    raise     = readInt(sc,    "Emelés: ");
    Employee e = new Employee(firstName, lastName,
                                salary, job);
    System.out.println(e.getFirstName() + " " +
                       e.getLastName()  + " beosztása: " +
                       e.getJob()       + ", fizetése: " +
                       e.getSalary());
    e.raiseSalary(raise);
    System.out.println("Emelés után:");
    System.out.println(e); // kiírás toString() használatával
}
```

toString

- Az objektumban tárolt információ szöveges megjelenítése összetett feladat, ezért nem akarjuk azt többszörösen elvégezni.
- A kód függvényé történő kiemelése most is hasznos, azonban ennek meg van Javában a megfelelő módja.
- Implementáljuk az alábbi függvényt a kiírandó objektum osztályában:

```
@Override
public String toString() {
    return firstName + " " + lastName +
        " beosztása: " + job +
        ", fizetése: " + salary;
}
```

toString

- Elemi adattípusok esetén azok értékét adja vissza
- Objektumok esetén alapértelmezetten azok memória referencia címét adja meg
- A metódust felüldefiniálva befolyásolhatjuk, hogy mit adjon vissza értékként. Konzolra való kiírásnál ez nagyban leegyszerűsíti a kódot.
- Az osztályban egy őssosztály metódusának felüldefiniálását az `@Override` annotációval jelezhetjük.
- Mi az **Employee** őssosztálya?!
 - Javában az `Object` minden osztály őssosztálya

Feladat

- Az előző feladat megoldását felhasználva hozzunk létre egy tárolót, amelybe beolvasunk a konzolról alkalmazottakat.
- Kérjük be, hogy milyen beosztású alkalmazottak fizetését akarjuk emelni és mennyivel, majd hajtsuk is végre az emelést.
- Írjuk ki az alkalmazottakról a konzolra a róluk ismert információkat.
- Mondjuk meg, hogy milyen beosztású és mennyit keres az, akinek a legtöbb a fizetése.

Megoldás

- ▶ Javában sok adatszerkezet áll rendelkezésünkre, amelyek közül választhatunk. Ezek közül a felhasználás/adattárolás módja szerint szokás választani: (bővebben később)
 - ▶ Indexelhető:
 - ▶ ArrayList, ArrayLinkedList, Vector, Stack...
 - ▶ Láncolt listás:
 - ▶ Queue, DeQueue, PriorityQueue, LinkedList...
 - ▶ Fa adatszerkezetű:
 - ▶ TreeSet, TreeMap...
 - ▶ Hasító függvényt alkalmazó:
 - ▶ HashSet, LinkedHashSet, HashTable, HashMap...
- ▶ Válasszuk most az ArrayList-et

Megoldás

- Az alkalmazottakat egy ciklusban olvassuk be, hogy eltároljuk őket.
- Az alkalmazott beolvasását akár külön metódusba is szervezhetjük.
- Beolvassuk a módosítással kapcsolatos adatokat.
- Egy ciklusban végigmegyünk minden alkalmazotton, és a megfelelő beosztásúaknál megemeljük a fizetést.
- Lefuttatjuk a maximum keresés programozási tételt az alkalmazottakon, hogy megtaláljuk a legjobban fizetett alkalmazottat, majd kiírjuk a beosztását és a fizetését.

Megoldás – Alkalmazott beolvasása

```
public static Employee readEmployee(Scanner sc){
    String firstName = readString(sc, "Vezetéknév: ");
    String lastName  = readString(sc, "Keresztnév: ");
    String job       = readString(sc, "Beosztás: ");
    int salary       = readInt(sc, "Fizetés: ");
    return new Employee(firstName, lastName,
                           salary, job);
}
```

Megoldás – Adatok beolvasása

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    ArrayList<Employee> employees = new ArrayList<>();  
  
    for (int i = 0; i < 3; i++) {  
        employees.add(readEmployee(sc));  
        System.out.println(employees.get(i));  
    }  
    ...  
}
```


Megoldás – Fizetésemelés

```
public static void main(String[] args) {  
    ...  
    int raise = readInt(sc, "Fizetésemelés: ");  
    String job = readString(sc, "Beosztás: ");  
  
    for (Employee e : employees) {  
        if (e.getJob().equals(job)) e.raiseSalary(raise);  
        System.out.println(e);  
    }  
    ...  
}
```

Objektumok összehasonlítása

- A beosztások összehasonlításához az `equals()` metódust használtuk, mivel objektumok esetében az egyenlőség operátor (`==`) az objektumok referenciáit hasonlítja össze!
- Stringek esetében az `equals()` és `equalsIgnoreCase()` metódusok használhatók a hagyományos string összehasonlításhoz.
- Saját osztályokon összehasonlítást definiálni az `equals()` és `hashCode()` metódusok implementálásával lehet (automatikusan generálható a legtöbb Java fejlesztő környezetben).

Megoldás – Maximum keresés

```
public static void main(String[] args) {  
    ...  
    Employee richMan = employees.get(0);  
    for (Employee e : employees)  
        if (e.getSalary() > richMan.getSalary())  
            richMan = e;  
  
    System.out.println("Legjobban fizetett alkalmazott:"  
        + richMan);  
}
```

A megoldás hiányosságai

- ▶ Ellenőriznünk kellene az adatok helyességét, és jelezni, ha a felhasználó pl. nem számot ad meg, amikor azt várunk.
- ▶ Hibakereséshez a programot többször is futtatnunk kell, és ilyenkor kézzel kell megadnunk mindig az adatokat.
- ▶ Az **Employee** osztályban nem implementáltuk az `equals` és `hashCode` metódusokat, így a hasítófüggvényt alkalmazó adatszerkezeteket még nem tudjuk velük helyesen használni.
- ▶ Ezek megoldásait majd a későbbiekben fogjuk elsajátítani.

JAVA fejlesztőeszközök és -környezetek

Gyakran használt eszközök

- JAVA Development Kit (JDK)
- JAVA Runtime Environment (JRE)
- JAVA dokumentáció
<https://docs.oracle.com/javase/8/docs/>

- NetBeans <http://netbeans.org/>
- Eclipse <http://www.eclipse.org/>
- IntelliJ IDEA <http://www.jetbrains.com/idea/>

Netbeans gyorsbillentyűk

- ▶ Futtatás: F6
- ▶ Aktuális fájl futtatása: Shift + F6
- ▶ Kódkiegészítő: Ctrl + szóköz
- ▶ Kódgenerálás: Alt + Insert
- ▶ Kód formázása: Alt + Shift + F
- ▶ Hibajavítási tippek: Alt + Enter
- ▶ Átnevezés: Ctrl + R
- ▶ Változó kiemelése kifejezésből: Alt + Shift + V
- ▶ Attribútum kiemelése kifejezésből: Alt + Shift + E
- ▶ Függvény készítése kódrészletből: Alt + Shift + M