Osztályok létrehozása

A konkrét életbeli objektum(csoportok) formai leírása lesz az osztály. Osztályokat kell létrehoznunk ahhoz, hogy majd létre tudjunk hozni a memóriában objektumokat. Osztályokkal olyan objektumokat formalizálunk, melyek azonos tulajdonságokkal és műveletekkel rendelkeznek, mint például az emberek. Sok különböző ember létezik, de mégis rendelkeznek közös tulajdonságokkal:

van név és kor tulajdonságuk, valamint mindenki tudja magáról, hogy férfi-e.

Készítsünk egy ilyen osztályt, melynek kulcsszava a class. Általában osztályokat külön fájlokba készítünk, melynek neve megegyezik a létrehozni kívánt osztállyal, kiegészítve a .java kiterjesztéssel.

Tehát készítsük el az Ember.java fájlt:

```
public class Ember {
}
```

Az embereknek van néhány közös tulajdonságuk, amelyet UML-ben tulajdonságnak, attribútumnak hívunk. Most a név, kor , valamint a férfi-e tulajdonságot szeretnénk a programunkban használni. Ezek legyenek String, int és boolean típusú változók.

```
public class Ember {
    String nev;
    int kor;
    boolean ferfi;
}
```

Elkészült az osztályunk, már csak a viselkedést kellene valahogy a forráskódban is reprezentálni. Erre metódusok lesznek a segítségünkre, melyeket az osztályba írunk bele, és ezek azt jelentik, hogy az adott osztályból létrejövő objektumok milyen viselkedéssel rendelkezhetnek. Például, az emberek tudnak köszönni, készítsük el a koszon metódust.

```
public class Ember {
    String nev;
    int kor;
    boolean ferfi;
    public void getKoszon(){ // kívülről elérhető metódus (interfész)
        String str=("Szia! " + nev + " vagyok és " + kor + " éves, mellesleg " + (ferfi ? "férfi." : "nő.");
        System.out.println(str);
    }
}
```

Az így elkészült osztályból objektumokat készítünk, ezt **példányosításnak** hívjuk. Minden egyes objektum pontosan egy osztály példánya, azonban egy osztályból számtalan objektumpéldány is létrehozható. Hozzuk létre Józsit most a main függvényben a new operátor segítségével.

```
public static void main(String[ ] args){
    Ember jozsi = new Ember();
    jozsi.getKoszon(); // elérhető metódus (interfész)
}
```

Fordítsuk, majd futtassuk le a programot.

Kimenet: `Szia! null vagyok és 0 éves, mellesleg nő.`

Nem éppen lett Józsi, még férfi sem igazán. Ennek az az oka, hogy az osztályban lévő adattagok nincsenek rendesen inicializálva, a típusokhoz tartozó *default* értéket vették fel.

Ahhoz, hogy ezt megfelelő módon megtehessük, konstruktort kell létrehoznunk.

Konstruktor

Ez a speciális függvény fogja létrehozni a példányokat (objektumokat) az osztályokból, beállítani az osztályban deklarált változókat a konkrét, adott objektumra jellemző értékekre.

Ilyen alapból is létezik (hiszen az előbb mi sem írtunk ilyet), de mi paramétereseket is megadhatunk, ami segítségével gyorsan és könnyen tudjuk az adott objektumot inicializálni. Megjegyzendő, hogy ha mi készítünk **bármilyen** konstruktort, akkor a fordító nem készít egy *default*, paraméter nélküli konstrukotrt (ahogy azt tette korábban).

- A konstruktor nevének meg kell egyeznie az osztály nevével
- Visszatérési értéke/típusa nincs (nem is lehet!)
- Új objektum létrejöttekor fut le

```
/*ez egy default konstruktor:
  alapból létezik, nem muszáj kiírni, hacsak nem szánunk neki speciális sze-
  repet.*/
public class Ember {
}
/*paraméteres konstruktor:
  nekünk kell létre kell hoznunk (ha szeretnénk paraméteres konstruktort)*/
public Ember (String nev, int kor, boolean ferfi) {
      this.nev = nev;
      this.kor = kor;
      this.ferfi = ferfi;
}
```

this kulcsszó - az objektum saját magára tud hivatkozni vele, ennek akkor van gyakorlati haszna például, amikor egy metódusban szereplő paraméter neve megegyezik az osztályban deklarált adattag nevével, akkor valahogy meg kell tudnunk különböztetni, hogy melyik az objektum tulajdonsága, és melyik a paraméter.

A paraméteres konstruktorral könnyen, egy sorban létrehozhatjuk és értéket is adhatunk a példányoknak.

Az így kiegészített osztály után hozzuk létre most már valóban Józsit.

```
public static void main(String[ ] args){
      Ember jozsi = new Ember("Józsi", 20, true);
      jozsi.getKoszon(); // elérhető metódus (interfész)
}
```

Kimenet: `Szia! Józsi vagyok és 20 éves, mellesleg férfi.`

Láthatóságok

A láthatóságok segítségével tudjuk szabályozni adattagok, metódusok elérését, ugyanis ezeket az objektumorientált paradigma értelmében korlátozni kell, kívülről csak és kizárólag ellenőrzött módon lehessen ezeket elérni, használni. Része az implementáció elrejtésének, azaz akár a programot, akár az osztályt anélkül használják kívülről a felhasználók, hogy pontosan ismernék annak működését.

Láthatóságok:

public - mindenhonnan látható

private - csak maga az osztály látja

nem írtunk ki semmit - "friendly" láthatóság, csomagon belül public, csomagon kívül private

protected - a csomag, az osztály és az azokból származtatott gyermekosztályok látják

Getter/Setter

Az adattagok értékeinek lekérésére (getter), valamint inicializálás utáni módosítására (setter) használjuk. Ezek összefüggenek azzal, hogy egy objektum adatait csak ellenőrzött módon lehet lekérni, illetve módosítani, ezeken a függvényeken keresztül. Általában csak simán lekérésre, és beállításra használjuk, de ide tehetünk mindenféle ellenőrzéseket.

Ezekkel tulajdonképpen elrejtjük a változókat és akár még módosíthatunk is az értékeken lekéréskor vagy megadáskor (mondjuk setternek megadhatjuk, hogy ne hagyja 0-ra állítani a változó értékét (például egy osztás nevezőjét))

Hagyományos elnevezése: get + adattag neve illetve set + adattag neve.

```
//getter
public String getNev() {
    return this.nev;
}
//setter
public void setNev(String nev) {
    this.nev = nev;
}
//setter
public void setKor(int kor) {
    if (kor > 0) {
        this.kor = kor;
    }else{
        System.err.println("A kor csak pozitív szám lehet!");
     }
}
```

Boolean értékek esetében a getter függvényt általában is + adattag neve formában szoktuk elnevezni.

```
public boolean isFerfi(){
    return this.ferfi;
}
```

A munkaterületen jobb klikk és Source > Generate Getters and Setters... menüpontban találjuk meg az ide kapcsolódó varázslót.