1 Osztályok

A magyarázatokat érdemes az alább található forráskóddal együtt olvasni.

1.1 Adattagok (2. - 3. sor)

Osztályoknak lehetnek adattagjaik, ezek olyan változók, melyek egy ilyen típusú objektumhoz tartoznak. Ilyen lehet például a komplex számoknál a valós és képzetes rész.

Egy adattag lehet

- *private*, azaz csak az osztály által elérhetõ
- *public*, mindenki számára elérhetõ
- *protected*, a leszármazottak által elérhetõ
- *package*, azon package-en belül elérhetõ, melyben az osztály található

Elõször adjuk meg egy adattag elérési szintjét, pl, hogy *private*, aztán a típusát pl *int*, végül a nevét.

1.2 Konstruktorok (5. - 18. sor)

Ahhoz, hogy létre tudjunk hozni egy objektumot kell írnunk az osztályba konstruktort. Egy konstruktor hasonlóan néz ki mint egy metódus, de nincs visszatérési értéke és a neve mindig az osztály nevével egyezik meg. A paraméterek bármik lehetnek.

Például az elsõ konstruktorban (5. - 8. sor) egy egészet és egy stringet használunk, hogy létrehozzuk az objektumot. A második konstruktorban ami említésre méltó, hogy nem okoz gondot, ha a paraméternek ugyanaz a neve, mint az adattagnak, viszont ekkor használnunk kell a *this* kulcsszót, hogy az adott objektumra utaljunk és annak az adattagjait változtassuk, *this* nélkül a paramétert érjük el.

1.3 Metódusok (20. - 30. sor)

A metódusokra úgy érdemes gondolni, mint mûveletekre amiket az objektumokon tudunk végrehajtani. Az elsõ példa (20. - 22. sor) kap egy számot, majd visszaadja az adott objektum *egeszAdattag* adattagjával vett szorzatot ezzel a kapott számmal.

A második egy példa *private* metódusra, amit csak az osztályon belül lehet elérni. Végül a harmadiknak nincs visszatérési értéke és megváltoztatja az objektumot amin meghívjuk, használva az elõzõ *private* metódust.

```
1  public class MyClass {
2          private int egeszAdattag;
3          private String stringAdattag;
4
5          public MyClass(int egesz, String string) {
6                  egeszAdattag = egesz;
7                  stringAdattag = string;
8          }
9
```

```
10          public MyClass(int egeszAdattag) {
11                  this.egeszAdattag = egeszAdattag;
12                  this.stringAdattag = "";
13          }
14
15          public MyClass() {
16                  egeszAdattag = 0;
17                  stringAdattag = "";
18          }
19
20          public int peldaMetodus(int szam) {
21                  return szam * egeszAdattag;
22          }
23
24          private int peldaPrivateMetodus() {
25                  return egeszAdattag * 2;
26          }
27
28          public void peldaValtoztatoMetodus() {
29                  egeszAdattag = peldaPrivateMetodus();
30          }
31  }
```

2 Objektumok

Itt nem fogom részletesen leírni a változók és objektumok közötti relációt, ezt megtaláljátok a megfelelõ elõadásban.

2.1 Létrehozás (3. - 5. sor)

A *new* kulcsszóval tudunk létrehozni objektumokat, ekkor a típus (osztály) valamelyik konstruktorát fogjuk használni. A példában mindegyik konstruktort kipróbáltam.

2.2 Objektumok használata (7. - 9. sor)

Az objektumok elérhetõ adattagjait és metódusait a . (pont) operátorral érjük el. Amint látjátok a *private* metódust direkt kikommenteztem, jelezve, hogy azt nem fogjuk tudni meghívni.

```
1   public class Main {
2           public static void main(String[] args) {
3                   MyClass pelda1 = new MyClass(5, "kutya");
4                   MyClass pelda2 = new MyClass(3);
5                   MyClass pelda3 = new MyClass();
6
7                   int eredmeny = pelda1.peldaMetodus(4);
8                   //pelda2.peldaPrivateMetodus();
```

```
9                   pelda3.peldaValtoztatoMetodus();
10          }
11  }
```

3 Statikus dolgok

Ha valami statikus, legyen az metódus vagy adattag, akkor az osztályhoz tartozik és nem az objektumhoz.

Egy *static* adattag például számolhatja, hogy hányszor volt meghívva az adott osztály konstruktora (2. és 8. sor). Vegyük észre, hogy a változónak adtam kezdõértéket is. Egy *static* metódus pedig a *static* adattagokon végezhet mûveleteket, viszont vigyázzunk, mert nem *static* változókon nem tud mûveletet végezni (mert nincs objektum aminek az adattagján végezze).

A használati példában (19. - 25. sor) létrehozok egy ilyen objektumot, ekkor az osztály *szamlalo* adattagja 1-re változik. Láthatjátok, hogy a statikus adattagokat el lehet érni egy ilyen típusú objektumon és közvetlenül az osztályon keresztül is (21. és 23. sor). A statikus metódusokra ugyanez igaz (25. és 27. sor).

```
1   public class StaticPelda {
2           public static int szamlalo = 0;
3
4           private int adattag;
5
6           public StaticPelda() {
7                   adattag = 0;
8                   szamlalo++;
9           }
10
11          public static void novel(int szam) {
12                  //adattag += szam;
13                  szamlalo += szam;
14          }
15  }
16
17  public class Main {
18          public static void main(String[] args) {
19                  StaticPelda pelda = new StaticPelda();
20
21                  int szam = pelda.szamlalo;
22
23                  int szam2 = StaticPelda.szamlalo;
24
25                  StaticPelda.novel(5);
26
27                  pelda.novel(4);
28          }
29  }
```

4 Öröklés

Egy osztály örökölhet egy másik osztályból, ekkor a leszármazott eléri az õse *protected* és gyengébb rejtettségû adattagjait és metódusait. Valamint az õsosztállyal hivatkozhatunk a leszármazottra.

4.1 Adattagok

A példában az *OsPelda* egy osztály (1. - 25. sor), melynek leszármazottja a *LeszarmazottPelda* (27. - 51. sor), az õsnek van 3 adattagja, egy *public*, egy *protected* és egy *private* (2. - 4. sor), ezek közül a leszármazott csak az elsõ kettõt fogja tudni közvetlenül elérni, ami *private* azt nem.

A leszármazottnak közvetlenül csak egy adattagja van (28. sor), viszont az õsön keresztül van még 3 (ami közül 1-et nem tud elérni).

4.2 Konstruktorok

A leszármazott elsõ konstruktora meghívja az õse konstruktorát a *super* kulcsszó segítségével, ennek az lesz az eredménye, hogy az õsosztály 3 adattagja megkapja az adott *a, b, c* értékeket. Utána pedig beállítjuk a leszármazott adattagjának az értékét is. Amint láthatjátok így be tudtuk állítani azt az adattagját is az õsnek, amit közvetlenül nem érünk el a leszármazottból (*osPrivate*).

A másik konstruktora a leszármazottnak mutatja, hogy az *osPrivate* adattagot így fapadosan nem tudjuk beállítani, mert nincs hozzáférésünk.

Nezzünk rá a *Main-re* (53. - 74. sor), itt simán létrehozunk 3 objektumot. Viszont az 57. sorban valami furcsa dolog történik, a változó *OsPelda* típusú, viszont az objektum amit hozzárendelünk *LeszarmazottPelda* típusú. Ilyet lehet csinálni, viszont ekkor vigyázni kell, mert ezen a változón (*opLp*) keresztül csak azokat a metódusait és adattagjait érjük el közvetlenük az objektumnak, amik már az õsosztályban is megvoltak.

4.3 Metódusok

Az õsosztálynak két metódusa volt *szumma* és *prod*, ezeket is örökli a leszármazott. Viszont a leszármazott felülírja (*Override*) a *szumma*-t, méghozzá úgy, hogy legyen az õs által számolt szumma (*super*-en keresztül elérhetjük az õsosztály metódusait, amikhez van hozzáférésünk) és a leszármazott új adattagja. A *prod*-ot nem írtuk felül, viszont írtunk egy új metódust *kul* névvel.

Nézzünk rá a *Main*-re, az 59.-60. sorban láthatjuk, hogy az õszosztály úgy viselkedik ahogy elvárnánk. Valamint a leszármazott is a 62.-64. sorban, igaz a *prod*-ot nem írtuk felül így az nem számolja bele a leszármazott új adattagját, de a *szumma* igen. Végül az *opLp*-n keresztül elérjük a *szumma* és *prod* metódusokat, de a *kul*-t nem, annak ellenére, hogy ennek az objektumnak van ilyen metódusa, de a változónak amiben tárolva van nincs. De amin meglepõdhetünk, hogy ha megnéznénk a *szumma* eredményét, akkor azt látnánk, hogy a leszármazott *szumma* metódusával számolt. Ez a nagy varázslat az öröklésben, hogy ha *Override*-olva van egy metódus, akkor is azt fogja használni, hogyha az õsosztállyal van hivatkozva az objektum.

4.4 Tömbözés

Az elõzõ azt is jelenti, hogy ha van egy tömbünk mely õsosztály tömb, attól még tárolhatunk benne leszármazott objektumokat. Sõt amikor az utána levõ *for* ciklusban (75.-77. sor) szummázzuk ezt

a két objektumot, akkor a megfelelõ szummákat használja (azaz a 2. elemnél a *LeszarmazottPelda szumma* metódusát fogja használni).

```
1  public class OsPelda {
2          public int osPublic;
3          protected int osProtected;
4          private int osPrivate;
5
6          public OsPelda(int a, int b, int c) {
7                  osPublic = a;
8                  osProtected = b;
9                  osPrivate = c;
10         }
11
12         public OsPelda() {
13                 osPublic = 0;
14                 osProtected = 0;
15                 osPrivate = 0;
16         }
17
18         public int szumma() {
19                 return osPublic + osProtected + osPrivate;
20         }
21
22         public int prod() {
23                 return osPublic * osProtected * osPrivate;
24         }
25 }
26
27 public class LeszarmazottPelda extends OsPelda {
28         public int leszarmazottAdat;
29
30         public LeszarmazottPelda(int a, int b, int c, int d) {
31                 super(a, b, c);
32                 leszarmazottAdat = d;
33         }
34
35         public LeszarmazottPelda(int a, int b, int d) {
36                 osPublic = a;
37                 osProtected = b;
38                 //osPrivate = c;
39                 leszarmazottAdat = d;
40         }
41
42         @Override
43         public int szumma() {
```

```
44                  //return osPublic + osProtected + osPrivate + leszarmazottAdat;
45                  return super.szumma() + leszarmazottAdat;
46          }
47
48          public int kul() {
49                  return super.szumma() - leszarmazottAdat;
50          }
51  }
52
53  public class Main {
54          public static void main(String[] args) {
55                  OsPelda op = new OsPelda(3, 2, 7);
56                  LeszarmazottPelda lp = new LeszarmazottPelda(5, 2, 7);
57                  OsPelda opLp = new LeszarmazottPelda(3, 9, 1, 4);
58
59                  int a = op.szumma();
60                  int b = op.prod();
61
62                  int c = lp.szumma();
63                  int d = lp.prod();
64                  int e = lp.kul();
65
66                  int f = opLp.szumma();
67                  int g = opLp.prod();
68                  //int h = opLp.kul();
69
70                  OsPelda[] t = new OsPelda[2];
71                  t[0] = new OsPelda(4, 2, 1);
72                  t[1] = new LeszarmazottPelda(7, 4, 9, 1);
73
74                  int osszeg = 0;
75                  for(OsPelda os : t) {
76                          osszeg += os.szumma();
77                  }
78          }
79  }
```

5 Interface

Az *interface*-ekre gondoljatuk úgy mint egy tulajdonságra, amivel fel lehet ruházni az osztályokat.

Egy *interface* csak azt tartalmazza, hogy milyen metódusokat kell megvalósítani, ha egy osztály implementálja ezt az *interface*-t. Ezeket úgy jelezzük, hogy a metódus fejét írjuk csak le (elérhetőség, visszatérési érték, függvény neve, paraméterek). Pl 1. - 7. sor.

Egy osztály több *interface*-t is implementálhat ahogy a 9. sorban láthatjátok. Valamint az implementált *interface*-ek metódusait *Override*-al cimkézzük.

A használatukból (47. - 74. sor) az olvasható ki, hogy egy objektumra hivatkozhatunk az interface típusával, de ekkor csak azokat a metódusokat érjük el, amiket implementál (59. - 60. sor). Valamint ilyen típusú tömböt is létrehozhatunk (62. sor).

```
1  public interface Szummazhato {
2          public int szumma();
3  }
4
5  public interface Prodolhato {
6          public int prod();
7  }
8
9  public class Osztaly1 implements Szummazhato, Prodolhato {
10         private int adat1;
11         private int adat2;
12
13         public Osztaly1(int a, int b) {
14                 adat1 = a;
15                 adat2 = b;
16         }
17
18         @Override
19         public int szumma() {
20                 return adat1 + adat2;
21         }
22
23         @Override
24         public int prod() {
25                 return adat1 * adat2;
26         }
27
28         public int ketszeres() {
29                 return adat1 * 2;
30         }
31 }
32
33 public class Osztaly2 implements Szummazhato {
34         private int adat;
35
36         public Osztaly2(int a) {
37                 adat = a;
38         }
39
40         @Override
41         public int szumma() {
42                 return adat;
```

```
43          }
44  }
45
46  public class Main {
47          public static void main(String[] args) {
48                  Osztaly1 o1 = new Osztaly1(2, 3);
49
50                  int s = o1.szumma();
51                  int p = o1.prod();
52                  int k = o1.ketszeres();
53
54                  Szummazhato sz1 = new Osztaly1(4, 1);
55                  Szummazhato sz2 = new Osztaly2(5);
56
57                  int a = sz1.szumma();
58                  int b = sz2.szumma();
59                  //int c = sz1.prod();
60                  //int d = sz1.ketszeres();
61
62                  Szummazhato[] t1 = new Szummazhato[2];
63                  t1[0] = new Osztaly1(4, 5);
64                  t1[1] = new Osztaly2(5);
65
66                  int szum = 0;
67                  for(Szummazhato sz : t1) {
68                          szum += sz.szumma();
69                  }
70
71                  Prodolhato[] t2 = new Prodolhato[2];
72                  t2[0] = new Osztaly1(2, 3);
73                  //t2[1] = new Osztaly2(5);
74          }
75  }
```