Java programozási nyelv 2007-2008/ősz
2. óra

Osztályok, objektumok

Osztályok felépítése
Mezők, metódusok, módosítók
JavaBeans
Példányosítás
Öröklés

legradi.gabor@nik.bmf.hu szenasi.sandor@nik.bmf.hu

Osztályok, objektumok

Témakörök

- Osztályok felépítése
- Mezők, módosítók
- Metódusok, módosítók
- JavaBeans
- Példányosítás
- Öröklés
- Interfészek

Osztályok felépítése

- Java-ban az osztályok belső szerkezete három, jól elkülöníthető részre osztható
  - mező(k): az objektum aktuális állapotát tárolják
  - metódus(ok): valamilyen művelet végeznek el, ez lehet mezők értékének lekérdezése vagy megváltoztatása
  - konstruktor(ok): az objektum létrehozásakor végzi a mezők beállítását
- Megjegyzés: ezeken felül lehetnek még úgynevezett „inicializáló blokkok” is
- Megjegyzés: más nyelvekkel ellentétben itt nincs destruktor. A már nem használt objektumpéldányok memóriából történő eltávolítását, és a memória felszabadítását a JVM végzi el.

  Ennek idejét nem ismerjük, a szemétgyűjtő metódus csak annyit garantál, hogy felszabadítás előtt meghívja az *Object* ősosztálytól örökölt void finalize() metódust

Osztályok felépítése (2.)

- Java-ban az osztályok belső szerkezete három részre osztható (példa):

```
/* demo.java */
public class Demo {
   /* meződefiníciók */
   int mező1;
   int mező2;

   /* metódusdefiníció */
   public void incMezo1(){
       mezo1 = mezo1 +1;
   }

   /* konstruktordefiníció */
   public Demo() {
       mezo1 = 0;
   }
}
```

Osztályszintű módosítók

- Hozzáférést szabályozó módosítók
  - **public**: az osztály tetszőleges más osztály számára elérhető. Ilyen publikus osztály csak azonos nevű, .java kiterjesztésű fájlban helyezkedhet el
  - jelzés nélküli: az osztály az őt tartalmazó csomagon belüli osztályok számára érhető csak el
- Egyéb módosítók
  - **abstract**: Az osztályból objektum nem példányosítható, csak a leszármazottak számára érdekes. Ezzel kikényszeríthető az öröklés.
  - **final**: Az osztályból nincs lehetőség leszármazottak készítésére, a fordító hibát jelez minden hasonló próbálkozásnál. Ezzel meggátolható az öröklés.
    - Védelmi okokból (pl. String osztály)
    - Tervezési okokból

Osztályok, objektumok

Témakörök

- Osztályok felépítése
- Mezők, módosítók
- Metódusok, módosítók
- JavaBeans
- Példányosítás
- Öröklés
- Interfészek

Mezők, módosítók

- A mezők jellemzői
  - Definiáláskor meg kell adni a mezőben tárolandó adat típusát
  - Nevet kell kapjon a mező, mivel ezen a néven lehet rá hivatkozni később
  - A definícióban van lehetőség kezdőérték beállításra is
  - Kezdőértéket még konstruktorban vagy inicializáló blokkban is meg lehet adni
  - Kezdőérték megadás hiányában a futtató rendszer ad kezdőértéket a mezőnek, azaz olyan helyzet nem fordulhat elő, hogy nem inicializált egy objektummező. Ez nagyban hozzájárul a Java programok biztonságos működéséhez.
    Alapértelmezett kezdőértékek:
    - Számok esetén 0
    - Logikai érték esetén false
    - Referenciák esetén null

Mezők, módosítók (2.)

- Az objektumban definiált mezőket minősített formában lehet elérni, a minősítést a pont operátorral („.”) lehet elérni:

```
public class Proba{
    public int mezo1 = 12;
}
```

A programunkban szerepel az alábbi:

```
/* létrehozzuk az objektumpéldányt */
Proba p = new Proba();
/* kiiratjuk az objektum mezőjének értékét */
System.out.println(p.mezo1);
/* módosítjuk az objektum mezőjének értékét */
p.mezo1 = 13;
/* kiiratjuk az objektum mezőjének új értékét */
System.out.println(p.mezo1);
```

Mezők, módosítók (3.)

- Az objektumban definiált mezőket elláthatjuk úgynevezett módosítókkal is (viselkedés/hozzáférés szabályozás)
- Hozzáférést szabályozó módosítók
  - **public**: a mező írható/olvasható bármelyik másik objektumpéldányból,
  - **private**: a mező közvetlenül csak az adott osztályból, vagy csak metódusokon keresztül (get/set metódusok) érhető el,
  - **protected**: a mező csak a leszármazott osztályokban látható, illetve az osztályt tartalmazó csomagon belülről
  - Jelzés nélküli mező : csak a saját névteréből (csomagjából)

	osztály	csomag	leszármazott	egyéb
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
jelzés nélkül	✓	✓	✗	✗
private	✓	✗	✗	✗

Mezők, módosítók (3.)

- További módosítók
  - **final**: a mező értéke nem változtatható, konstans
  - **static**: a mező ún. osztályváltozó, az értéke nem az objektum példányban, hanem az osztálydefinícióban tárolódik, további tulajdonsága az is, hogy pontosan egy létezik belőle,
  - **transient**: az ún. szerializált objektumok lemezre történő kiírásakor ezt a mezőt tilos kiírni, a beolvasásnál tilos ennek a mezőnek az értékét beolvasni, az ilyen mező implicit kezdőértéket kap,
  - **volatile**: a futtatórendszer garantálja, hogy ez a változó a memóriában marad, s onnan azonnal kiolvasható. Több szál párhuzamos futtatása során az ezzel a kulcsszóval megjelölt mezőkről a szálak nem készíthetnek másolatokat a regiszterekben, ezzel szinkronizációs hibákat okozva

    (az operációs rendszer – ha a program futtatások során kevésnek bizonyul a memória, az épp „nem használt” memóriaterületeket a lemezre írhatja - swappelés)

Feladat

2.1. feladat: Készítsen objektumosztályt, amely hallgatók adatainak tárolására szolgál!
Az osztály neve legyen *Hallgato*, és tartalmazza a hallgató nevét, egy adott félévben kapott eredményeit (legfeljebb 10 elemű tömbben tárolva)!

Működésének kipróbálásához készítsen főprogramot tartalmazó osztályt *Proba* néven!

Osztályok, objektumok

Témakörök

- Osztályok felépítése
- Mezők, módosítók
- Metódusok, módosítók
- JavaBeans
- Példányosítás
- Öröklés
- Interfészek

Metódusok, módosítók

- A metódusok jellemzői
  - Minden metódus: függvény
  - Definiáláskor meg kell adni a visszaadott érték típusát
  - Visszaadott érték hiányában – ekkor eljárásról beszélünk – a definícióban a void kulcsszót kell megadni
  - A metódusok n-szer futhatnak le a használat során (n értéke természetesen lehet akár zérus is)
  - Ha van visszaadott érték, akkor kötelezően kell paraméteres `return` utasításnak is szerepelnie a metódusban (több is szerepelhet), és a vezérlésnek mindenképpen el kell érnie
  - Eljárásnál nem szükséges kiírni a `return` utasítást, a fordító automatikusan elhelyez egyet a metódus utolsó végrehajtott utasítása és a blokkzáró zárójel közé
  - Ugyanazon a néven, de más paraméterezéssel (paraméterek típusa, darabszáma) lehet egyszerre több metódust is definiálni (metódusok túlterhelése), ez külön kulcsszót nem igényel

Metódusok, módosítók (2.)

- A metódusok jellemzői (folytatás)
  - Minden metódus neve után meg kell adni a kerek zárójelpárt (mind a deklarálásnál, mind pedig a metódushívásnál), ez nem hagyható el paraméter nélküli metódusok esetén sem
  - private minősítésű metódust más típusú objektumból kívülről nem lehet elérni, csak az adott, vagy azonos típusú objektumok metódusai tudnak hozzáférni, meghívni
  - A metódus, az őt tartalmazó objektumra a `this` pszeudó-változóval tud hivatkozni
  - Java-ban minden metódus virtuális, másfajta viselkedés nincsen, ezért ezt sem szükséges külön kulcsszóval jelölni (kivéve persze a statikus metódusokat, ahol nincs értelme a virtualitásnak)
  - A metódusok más nyelvekhez hasonlóan tartalmazhatnak lokális változókat, illetve fogadhatnak paramétereket. Paraméterek esetén csak értékszerinti paraméterátadás értelmezhető.

Metódusok, módosítók (3.)

- A metódusok módosítói
  - **public, private, protected**, jelzés nélküli: hozzáférést szabályozzák, hasonló tartalmúak, mint a mezők esetében
  - **final**: a metódus a leszármazottakban nem definiálható felül
  - **static**: a metódus statikus, tehát nem objektumpéldányhoz, hanem az osztályhoz kötődik. Így értelemszerűen csak a statikus mezők érhetők el belőle
    Statikus metódusok meghívása történhet az osztály nevén keresztül, illetve egy adott típusú objektumon keresztül is
  - **native**: a metódust nem Java nyelven valósították meg
  - **abstract**: olyan metódus, amelynek csak a definíciója létezik, konkrét megvalósítása nem, így kikényszeríti az öröklést
  - **synchronized**: a metódust egy időben maximum egy objektum hívhatja meg, így biztosítható a közösen használt erőforrások biztonságos kezelése. Általában párhuzamos működésű (pl. többszálú) alkalmazás esetén van rá szükség

Metódusok, módosítók (4.)

- Kitüntetett szerepű a main metódus, amely maga a főprogram. E metódus elindítása jelenti a Java programunk elindulását

```
public class Hello{
    public static void main(String[] args){
      System.out.println("Helló Világ!");
    }
}
```

- A main metódus mindig paraméteres, ez a paraméter String tömb, amely a parancssorban a program neve után szereplő argumentumokat tartalmazza. A főprogram befejeződése egyben a Java program futásának befejezését is jelenti. Ha a programból idő előtt kell kilépni, akkor a System.exit() utasítást használjuk

Metódusok, módosítók (5.)

- A konstruktorok (speciális metódusok) jellemzői:
  - Neve egyezik az őt tartalmazó osztály nevével
  - Nincs jelzett visszaadott értéke, még void sem
  - A publikus, paraméter nélküli, üres törzsű konstruktort implicit konstruktornak nevezzük
  - Ha nem írunk implicit konstruktort, és nincs paraméteres változat sem, akkor a fordító létrehoz egyet
  - A konstuktorok pontosan egyszer futnak le a használat során
  - Ugyanazon a néven, de más paraméterezéssel (paraméterek típusa, darabszáma) lehet egyszerre több konstruktort is definiálni (konstruktorok túlterhelése)
    Ezek közül a példányosításkor az osztály neve után írt paraméterlistával tudunk választani
  - Java nyelven nincs lehetőség osztályszintű (statikus) konstruktorok megadására. Helyette használhatók statikus inicializáló blokkok, amikkel hasonló funkciót érhetünk el

Feladat

2.2. feladat: A *Hallgato* osztályt egészítse ki olyan metódusokkal, amelyek az osztályzatokat tartalmazó tömb elemeinek értékét tudják módosítani!

2.2.a. feladat: Legyen Javító metódus, amely növeli a jegy értékét, valamint legyen Rontó metódus is, amely csökkenti a jegyet! Ezek a metódusok paraméterként azt a pozíciót kapják, amely tömbelemet javítani vagy rontani kell! Ügyeljen arra, hogy a jegyet tartalmazó tömb elemei csak az 1 és 5 között lehetnek!

2.2.b. feladat: Legyen olyan javító és rontó metódus, amely két paramétert kap, amely paraméterek rendre a módosítandó tömbelem indexét,valamint a módosítás mértékét tartalmazzák!

2.2.c. feladat: Módosítsa az adatmezőket úgy, hogy csak metódusok segítségevel tudjon az értékükhöz hozzáférni, módosítani!

Feladat

2.3. feladat: A *Hallgato* osztályt egészítse ki olyan konstruktorral, amely paraméterként azt a tömböt tartalmazza, melyben a hallgató jegyei vannak, és a konstruktor ezekre ez értékekre állítsa be a jegy mező értékét!

2.4. feladat: („negatív tanár”) A *Hallgato* osztály implicit konstruktorát módosítsa úgy, hogy a jegy mezők értékét induláskor elégtelenre állítja!

Metódusok, módosítók (7.)

- Inicializáló blokkok
  - Nagy tömegű adat (pl. tömb), kezdeti értékének beállítására
  - A végrehajtás sorrendjét a forráskódbeli helyzet szabja meg
  - Lehet jelzés nélküli (példányszintű) vagy `static` (osztályszintű) minősítővel ellátott. Előbbi lefut minden példány létrehozásakor, utóbbi pedig az osztály betöltésekor (nem tudjuk megjósolni mikor, de az biztos, hogy az első példány létrehozása előtt)
  - Végrehajtásuk során később definiált változóra (értelemszerűen) nem hivatkozhatnak
- Példa (példányszintű inicializáló blokk)

```
public int[20] tomb; /* meződefiníció */
{   /* inicializáló blokk */
    tomb[0]=1;
    tomb[1]=1;
    for(int i=2; i<(tomb.length()-2),i++)
        tomb[i]=tomb[i-1]+tomb[i-2];
}
```

Osztályok, objektumok

Témakörök

- Osztályok felépítése
- Mezők, módosítók
- Metódusok, módosítók
- JavaBeans
- Példányosítás
- Öröklés
- Interfészek

Tulajdonságok

- A Java nyelvben nincsenek tulajdonságok, helyettük a mezőket kezelő metódusok elnevezésére ad konvenciót:
  Szám/szöveg mezők:
  - Kiolvasás: `<típus> get<név>()`
  - Módosítás: `void set<név>(<típus> value)`

  Logikai mezők:
  - Kiolvasás: `<típus> is<név>()`
  - Módosítás: `void set<név>(<típus> value)`

  Indexelt mezők:
  - Kiolvasás: `<típus> get<név>(int index)`
  - Módosítás: `void set<név>(int index, <típus> value)`
- Hasonló ajánlások találhatók az események kezelésére
- Az ezeket betartó osztályokat nevezzük Bean-eknek. Ez javítja a kód emberi/gépi értelmezését (ezeket a komponenseket a fejlesztőeszközök is tudják kezelni)

JavaBeans

- JavaBeans: technológia az előregyártott és újrafelhasználható komponensek készítésének és használatának támogatására. Az ilyen komponenseket általában vizuális fejlesztőeszközben is lehet manipulálni
- Fajtái:
  - Komponensek: „hordozott elemek” (pl. gomb)
  - Konténerek: „hordozó elemek” (pl. ablak) (maguk is komponensek)
- Előny: támogatják a hierarchikus szoftverfejlesztést
- Komponens interfészek: segítségükkel komponensek feltárhatók és elemezhetők több szinten is támogatva (pl. *java.beans* és *java.lang.reflect* csomagok)
- „drag and drop” technológia alkalmazása

JavaBeans (2.)

- Tulajdonságok (properties)
  - Komponens megjelenése
  - Komponens viselkedése
- Tulajdonságlista: property sheet
- Tulajdonságok elérése: accessor metódusok
  - Lekérdező
  - Módosító
- Perzisztencia (állandóság, „örökéletűség”): objektum-szerializáció nyelvi szinten támogatva. Ennek köszönhetően a Bean-ek illetve az őket tartalmazó konténerek bármikor elmenthetők/visszatölthetők (természetesen platformtól függetlenül)
- „drag and drop” technológia alkalmazása: az esemény kezeléséhez szükséges kódot a fejlesztőeszköz generálja

JavaBeans (3.)

- Eseménykezelés alapjai
  - esemény forrása (forrás)
  - eseményobjektum
  - eseményfigyelő (nyelő)
- Tulajdonságok változása
  - Kötött: A Bean egy *PropertyChangeSupport* objektum segítségével egyszerűen felruházható azzal, hogy az egyes tulajdonságainak változásakor értesítsen (*PropertyChangeEvent) PropertyChangeListener-t megvalósító objektumokat*
  - Vétózható: Arra is van lehetőség, hogy a tulajdonság változása ne csak utólagos értesítéssel járjon, hanem a Bean a változás előtt erről értesítsen (*PropertyChangeEvent*) tetszőleges *VetoAbleChangeListener*-t megvalósító objektumokat. Ez eltérő kezelést igényel, mert az állapot csak az állapotjelzés után változik, és bármelyik így értesített objektum „megvétózhatja” a változást (*PropertyVetoException* kivétel dobással)

Osztályok, objektumok

Témakörök

- Osztályok felépítése
- Mezők, módosítók
- Metódusok, módosítók
- JavaBeans
- Példányosítás
- Öröklés
- Interfészek

Példányosítás

- Példányosítás: az a folyamat, amely során az osztálydefinícióból konkrét példányt hozunk létre

```
new osztálynév(konstruktor_paraméterek)
```

- A példányosítás folyamán fut le a konstruktor és az esetleges inicializáló blokkok is
- Pédául egy String típusú objektum létrehozása:

```
String szoveg = new String(”Hahó Világ!”);
```

vagy

```
String masikszoveg = ”Halló Világ!”;.
```

- Ezek után a változó nevével már tudunk a referencia által jelzett objektumra hivatkozni

```
System.out.println(szoveg);
System.out.println(masikszoveg);
```

Példányosítás (2.)

- A példányosítás során a `new` operátor lefoglalja a szükséges memóriát, létrehozza az objektumot, majd elindítja az inicializálását. A művelet visszatérési értéke egy referencia az új objektumra.
- Az így kapott referenciát eltárolhatjuk egy változóban, vagy akár azonnal továbbíthatjuk paraméterként stb.

```
Konyv k = new Konyv("Java");
```

vagy

```
bela.Olvas(new Konyv("Java"));
```

- A referenciához tartozó memóriaterületet – ha az objektumra már nem hivatkoznak – a JVM felszabadítja (a szabad memóriákat tartalmazó listához csatolja). A memória átcsatolása automatikus. Ha meg szeretnénk gyorsítani e folyamatot, akkor a System.gc() hívással utasítható a szemétgyűjtő az azonnali működésre.

Osztályok, objektumok

Témakörök

- Osztályok felépítése
- Mezők, módosítók
- Metódusok, módosítók
- JavaBeans
- Példányosítás
- Öröklés
- Interfészek

Öröklés

- A Java nyelv az egyszeres öröklést támogatja, azaz az utódosztályok pontosan egy ős osztályból örökölhetnek mezőket és metódusokat

  Megjegyzés: többszörös örökléshez hasonló viselkedést lehet elérni interfészek alkalmazásával
- Az öröklési fa legfelső szintjén az *Object* (*java.lang.Object*) osztály található, mely osztály az alábbi metódusokkal rendelkezik:
  - protected Object clone()
  - boolean equals(Object obj)
  - void finalize()
  - Class<?> getClass()
  - int hashCode()
  - void notify()
  - void notifyAll()
  - String toString()
  - void wait()

Öröklés (2.)

- Az öröklés jelzésere az utódosztályban az `extends` kulcsszó szolgál:

```
public class Utodosztaly extends Ososztaly{
}
```

- Az utód az ős minden tulajdonságát és metódusát automatikusan megörökli
- Módosítók szerepe az öröklés során
  - **public**: bármelyik osztályból elérhető
  - **private**: nem elérhető, csak az osztályon belülről
  - **protected**: elérhető, de csak az utódosztályban, vagy a csomagban
  - jelzés nélküli: csak azonos csomagban definiált osztályból érhető el

Öröklés (3.)

- A leszármazott osztályban az ős mezőinek, illetve metódusainak elérésére szolgál a `super` kulcsszó

  `super.mező` vagy `super.Metódus()`
- Konstruktor esetén a `super(paraméterek)` formában hívható meg az ős konstruktora (a leszármazott konstruktor kódjának tetszőleges részén)
- Konstruktorok esetén az öröklés szabályai hasonlóak a C# esetén tanultakhoz:
  - Kötelező meghívni az ős konstruktorát, bár erre különleges jelölés nem áll rendelkezésre. A konstruktoron belül a többi metódushoz hasonlóan a `super` kulcsszóval tehető meg
  - Ha ez nem történik meg, akkor a fordító automatikusan megpróbálja meghívni az ős implicit konstruktorát
  - Példányosításkor tehát az öröklési hierarchiában lévő osztályok konstruktorai sorra meghívódnak (legfelső őstől kezdve lefelé)

Feladat

2.5. feladat: Készítsen osztályt, amely könyvtári tagság nyilvántartására alkalmazható! Származtassa a *Kolcsonzo* osztályt a korábban kidolgozott *Hallgato* osztályból!

Egészítse ki az új osztályt olyan metódusokkal, amelyek a hallgató rossz jegyei esetén nem engedélyezik a kölcsönzést! Amennyiben szükséges, minimális és jól megindokolt módon változtasson a *Hallgato* osztály tulajdonságain is!

Osztályok, objektumok

Témakörök

- Osztályok felépítése
- Mezők, módosítók
- Metódusok, módosítók
- JavaBeans
- Példányosítás
- Öröklés
- Interfészek

Interfészek

- Az interfész olyan programegység, amely publikus és absztrakt metódusokat valamint publikus és statikus konstansokat tartalmazhat
- Mivel az interfész csak ilyen metódusokkal rendelkezhet, ezért ezt külön nem jelezzük, a fordító eleve elé érti a `public abstract` kulcsszavakat
  Hasonlóan a mezők `public final static` kulcs-szavai is elhagyhatók
- Példa (interfész)

```
interface IDemo {
    int EGY=1;
    void Kuld(String mit, String hova);
    String Fogad(String honnan);
}
```

Interfészek (2.)

- Az interfész önmagában nem példányosítható, valójában egy tervezési eszköz. Ahhoz, hogy tudjuk használni, implementálni kell egy osztálydefinícióval. Az implementálás azt jelenti, hogy az interfész összes metódusának konkrét jelentést adunk (megvalósítjuk)
- Példa (interfész implementálása)

```
public class Odemo implements IDemo {
   void Kuld(String mit, String hova) {
     System.out.println(mit+" → "+hova);
   };
   String Fogad(String honnan) {
     return honnan+" érkezett.";
   };
}
```

- Az így implementált osztály már használható, sőt ős-osztályként is megjelenhet

Interfészek (3.)

- Az interfész is tud örökölni más interfészektől, sőt egyszerre több interfésztől is. Az interfészek az osztályokhoz hasonlóan egy hierarchiát építenek fel
- Példa (interfész öröklés)

```
public class IOroko implements IEgyik,IMasik{
}
```

- Az így létrejött *IOroklo* interfész tartalmazni fogja mind az *IEgyik*, mind pedig az *IMasik* absztrakt metódusait és konstansait. A konstansok névütközése esetén minősítéssel tudjuk a használat során egyértelművé tenni a dolgokat. A metódusok esetén – mivel a törzs hiányzik – legfeljebb metódus túlterhelés lehet

Interfészek (4.)

- Az interfész alkalmas többszörös örökléshez hasonló megoldás létrehozására. Konkrét kódot ugyan így sem lehet örökölni több osztályból, viszont a polimorfizmus tekintetében a több interfészt megvalósító osztály megjelenhet bármelyik interfészt váró helyeken
- Például szeretnénk egy appletet, amelyben többszálú alkalmazás fut. Az *Applet* osztály és a *Thread* osztály együtt az egyszeres öröklés miatt nem használható. A *Thread* osztály viszont valójában a *Runnable* interfész implementációja, ezért az interfészt tudjuk használni:

```
public class MyApp extends Applet
                   implements Runnable {
}
```

Az utód tehát örökli az applet minden képességét, de felhasználható többszálú környezetben is!

Feladat

2.6. feladat: Készítsen interfészt, amely a korábbi kölcsönzéses feladathoz illeszkedik! Legyen az interfész azon metódusok gyűjteménye, amelyek a kölcsönzést végzik, valamint a konstansok a szükséges jegyérték minimumot tartalmazzák!

Ajánlott irodalom

Az óra anyagához kapcsolódó irodalom

- Nyékyné Gaizler Judit: Java 2 útikalauz programozóknak 1.3 II.; ELTE TTK Hallgatói alapítvány, Budapest

  74.-84.o., 35.-59. o.,523.-553. o.,
- The Java™ Tutorials: JavaBeans
  http://java.sun.com/docs/books/tutorial/javabeans