2.2 Örökldés, csomagok

Örökldés

Ez elzekben már találkoztunk az objektumorientáltság egyik alapfogalmával, az *egységbezárással*, mely szerint az adatokat és rajtuk végzett mveleteket egységbezártuk, egy osztályba. Ezen a gyakorlaton pedig megismerkedünk az UML-bl már ismersen hangozható örökldéssel. Az öröklés osztályok között valósul meg, egy szül (s, base, parent) és gyerek (leszármazott, derived) között.

Ennek során a gyerek osztály örökli a szüljének tulajdonságait és viselkedését. Ez jó dolog, mert nem kell újra írnunk ket, viszont a szültl örökölt egyes metódusokat speciálisabban is megvalósíthatjuk, felülírhatjuk ket(override), vagy akár újakat is definiálhatunk.

Javaban csak egyszeres örökldés van!

Ez azt jelenti, hogy egy osztálynak nem lehet kett, vagy több szülosztálya. Azt viszont nem zárja ki, hogy egy osztálynak több gyerek osztálya legyen, vagy hogy a gyerek osztálynak lehessenek saját gyerek osztályai.

Láthatóságok - protected

Találkozhatunk egy új láthatósággal, amelynek neve `protected`. Ennek a láthatóságnak segítségével biztosíthatjuk, hogy az egyes adattagok, metódusok a gyermekosztályok számára is láthatóak legyenek. A láthatóságokról egy összefoglaló táblázat:

Módosító	Osztály	Csomag	Leszármazottak	Mindenki
public	Látható	Látható	Látható	Látható
protected	Látható	Látható	Látható	Nem látható
nincs kulcsszó	Látható	Látható	Nem látható	Nem látható
private	Látható	Nem látható	Nem látható	Nem látható

A szülosztályban nem szükséges módosításokat eszközölnünk, a gyerek osztályban viszont jeleznünk kell, hogy melyik szülosztályból származik az adott osztály. Tekintsük az alábbi példát:

```
public class Torta {
    protected int szelet;
    private String iz;

    public String getIz() {
        return iz;
    }

    public void setIz(String iz) {
        this.iz = iz;
    }

    public Torta(int szelet, String iz) {
        this.szelet = szelet;
        this.iz = iz;
    }

    public void info() {
        System.out.println("Ez a torta " + this.iz + " íz, és "
                + this.szelet + " szeletes.");
    }
}
```

Extends, super

Ez egy általános torta osztály, tudjuk, hogy egy torta milyen íz és hogy hány szeletbl áll. Tortát bármilyen alkalomra vásárolhatunk, azonban lehetnek speciális alkalmak, amelyek esetében szeretnénk használni a már megírt `Torta` osztályunkat, de szeretnénk új, speciálisabb adatokat, metódusokat létrehozni a tortáinknak. Ilyen lehet például egy `SzulinapiTorta` osztály, amely torta, de szeretnénk egy új adatot is tárolni, mégpedig a rajta lév gyertyák darabszámát.

`extends` - ezzel a kulcsszóval érhetjük el az örökldést, az osztály deklarációjában, az osztály neve után kell írnunk, majd az `extends` kulcsszó után az sosztály nevét írjuk.

`super` - a gyerek osztályból hivatkozhatunk a szülre, annak adattagjaira (amiket látunk) és metódusaira is, ezeket `super.szuloMetodusanakNeve()`-szer parancsokkal érhetjük el.

```
public class SzulinapiTorta extends Torta {
    private int gyertyakSzama;

    public SzulinapiTorta(int szelet, String iz, int gyertyakSzama) {
        super(szelet, iz);
        this.gyertyakSzama = gyertyakSzama;
    }

    public void kivansagotTeljesit() {
        System.out.println("Kívánságod teljesült!");
    }

    public void info() {
        System.out.println("Ez a szülinapi torta " + this.getIz() + " íz, és "
                + this.szelet + " szeletes." + this.gyertyakSzama
                + " db gyertya van rajta");
    }

}
```

A szül konstruktora pedig egyszeren a `super` kulcsszó metódusként való használatával érhet el, például:

```
super(szelet, iz);
```

Ha az sosztály paraméter nélküli konstruktorát szeretnénk meghívni, akkor a `super();` hívás a gyermek osztály konstruktorában elhagyható. Ha nem a default konstruktorát használjuk az sosztálynak, akkor viszont **kötelez** a `super(arg1,arg2...argn);` meghívása a gyerekosztály konstruktorában!

A gyerekosztályban láthatjuk, hogy az sbl örökölt `info()` metódust felüldefiniáltuk (override), annak egy speciálisabb mködést adtunk.

Polimorfizmus - Többalakúság

A gyerek osztály egy példánya kezelhet a szül egy példányaként is, egy `SzülinapiTorta`objektumot tárolhatunk `Torta` példányként is (azaz egy `Torta` típusú referenciában), st akár egy s típusú tömbben eltárolhatjuk az s és gyerektípusokat vegyesen. Azonban, ha s típusként tárolunk egy gyerek típusú objektumot, akkor a gyerek típusú objektum saját osztályában definiált metódusait nem látjuk. Például:

```
public class TortaMain {
    public static void main(String[] args) {
        Torta csokiTorta = new SzulinapiTorta(15, "csoki", 9);
        csokiTorta.kivansagotTeljesit();    // Ez nem fog mködni
    }
}
```

A fenti kódrészlet nem mködik, mert az átlagos Torta nem tud kívánságot teljesíteni. Mivel egy `Torta` referenciában tároljuk a `SzülinapiTorta` objektumot, így csak a Tortában definiált metódusokat használhatjuk! *Ennek kiküszöbölésére később látni fogunk egy módszert.*

Ugyanígy nem lehetne a gyertyák számát lekérni vagy módosítani sem (még akkor sem, ha public láthatóságú lenne ez az adattag), mert az érintett Torta nem biztos, hogy rendelkezik ilyen adattaggal.

```
Torta[] cukraszda = new Torta[3];
cukraszda[0] = new Torta(20, "csokis-meggyes");
cukraszda[1] = new Torta(12, "epres");
cukraszda[2] = new SzulinapiTorta(12, "karamell", 12);
```

Fordításkor még nem tudjuk, hogy a `Torta` tömbbe milyen típusú objektumok lesznek: `Torta`objektumok, vagy pedig `SzulinapiTorta` objektumok, esetleg vegyesen, hiszen megtehetjük, hogy egy tömbbe gyerek típusokat teszünk.
Viszont, ha meghívjuk mindegyik elem `info()` metódusát, azt látjuk, hogy a sima torták esetében a `Torta` osztályban definiált metódus fut le, míg a születésnapi torta esetében a `SzulinapiTorta` osztályban definiált `info()` metódus hívódik meg. Ennek oka pedig a **kései kötés** (**late binding**). A kései és korai kötésrl bvebben itt és itt olvashatsz.

```
for (int i = 0; i < cukraszda.length; i++) {
    cukraszda[i].info();
}
```

Ennek kimenete:

```
Ez a torta csokis-meggyes íz, és 20 szeletes.
Ez a torta epres íz, és 12 szeletes.
Ez a szülinapi torta karamell íz, és 12 szeletes. 12 db gyertya van rajta
```

Ahogy említettük, a gyerek típus kezelhet sként, viszont ez **fordítva nem mködik**! `SzülinapiTorta` tömbbe nem tehetünk s típusú, azaz sima `Torta` objektumokat.

Final

Ha egy osztály final, akkor nem lehet gyereke. Ha egy metódus final, akkor nem lehet felülírni gyerekben.

```
final class EskuvoiTorta extends Torta {
}
```

Ennek jelentése, hogy az `EskuvoiTorta` osztályból nem származhat gyerekosztály. Errl bvebben itt és itt olvashatsz.

Csomagok

Osztályainkat csomagokba rendezhetjük, ahogy errl UML esetében is említést tettünk. Java osztályok esetében ez egy fizikai és egy logikai csoportosítást is jelent, általában különböz logikai egységenként hozunk létre csomagokat, illetve megkönnyíti a láthatóságok kezelését, és kiküszöbölhetek vele a névütközések is. Csomagokban lehetnek egyéb csomagok is, teljes csomag-hierarchiát hozhatunk vele létre (például egy csomag, ami a felhasználói felület megjelenítésével foglalkozik, ennek is lehetnek logikailag különálló részei, melyek ezen a csomagon belül helyezkedhetnek el).

Csomagokba szervezés

Ha az osztályunkat egy csomagba szeretnénk berakni, akkor egyrész be kell tennünk fizikailag abba a mappába vagy mappaszerkezetbe, ami a csomagunkat/csomaghierarchiánkat szimbolizálja, majd a forrásfájl els nem komment sorába ezt jelölni is kell a `package` kulcsszó használatával.

```
package hu.cukraszda.tortak;
```

Ennek jelentése: az osztályunk a `tortak` nev csomagban van, ez a csomag a `cukraszda` nev csomagban van, ez pedig a `hu` nev csomagban. Fizikailag az osztály a `projekt gyökérkönyvtár/hu/cukraszda/tortak` mappában van. A projekt gyökérkönyvára a projektben már sokszor látott *src* mappa.

Általában a fordított domain jelölést szokták alkalmazni csomaghierarchiák szervezésére. Bvebb információ a csomagok elnevezésérl itt olvashat ó.

Csomagot készíthetünk Eclipse alatt, viszont ilyenkor is az osztály elején ott kell lennie egy `package csomagneve;` sornak. Ezt a sort csak a ZH-n elég egyszeren odaírni, de a valóságban figyelni kell, hogy a könyvtárszerkezet ezzel megegyez legyen. ***A kötelez programoknál sokszor fordul el hiba ezzel kapcsolatban.***

Az osztályoknak van egy teljes nevük, amely a teljes csomag hierarchia + osztály nevébl tevdik össze. Tehát a fenti Torta osztálynak a teljes elérési neve (fully qualified name), ha a `hu.cukraszda.tortak` csomagban van: `hu.cukraszda.tortak.Torta`. Erre néhány további példa: `java.lang.String`.

Ugyanakkor ilyen hosszú nevet mi sosem írtunk, ha egy Stringet szerettünk volna létrehozni. És ezt a továbbiakban is elkerülhetjük, ha a szükséges csomagok tartalmát importáljuk a programunkba.

Csomagok importálása

Másik csomagban lév osztályokra hivatkozás eltt be kell ket importálni, vagy pedig teljes nevet kell használni, hogy a fordító tudja, mire gondolunk.

```
import hu.cukraszda.tortak.Torta;
import hu.cukraszda.tortak.SzulinapiTorta;
```

Ez vonatkozik saját osztályainkra, de az egyéb, nem általunk megírt osztályokra is (például a JDK kész osztályai). Kivétel ez alól a `java.lang` ö sszes osztálya, melyekbe tartozik például az összes csomagoló osztály vagy például a `String` osztály, ezt a csomagot eddig sem importáltuk soha, és tudtunk `String` példányokat létrehozni.

Használhatunk statikus importot is, ám ez sok esetben csak ront a kód olvashatóságán, érthetségén, karbantarthatóságán. A statikus importról bvebben itt olvashatsz.

Java fájljainkban a csomag jelölése (ha létezik ilyen) mindig megelzi az importálásokat.

Feladatok

Kocsmaszimulátor part 2:

A meglév `Kocsmáros` osztályunkat örököltessük az `Ember`-bl, mivel a kocsmárosok is emberek. Ettl automatikusan megkapja az `Ember` osztályb an deklarált adattagokat (változókat) is. A `Kocsmáros`hoz írjunk egy új paraméteres konstuktort, ami az `Ember` (tehát a s) konstruktorát is meghívja (`super` kulcsszó). *Elvileg a Kocsmáros így mködképes lesz, ha nem, akkor hozzuk mködképes állapotba további hibaelhárítással.*

Az `Ember` osztályunkból örököltessünk egy `Diák`ot is, aki egy különleges ember lesz. A diák rendelkezzen privát ösztöndíj változóval, ami megadja, hogy mennyi pénz keres a tanulással. Legyen egy `tanul()` metódusa, amitl az ösztöndíja mennyiségével n a pénze. Legyen egy paraméteres `tanul(int mennyit)` metódusa is, amivel be lehet állítani, mennyit tanuljon, és ennek megfelel pénzt kap.

Az `Ember`bl származzon még egy `Kidobó` is. Neki legyen egy privát `boolean dolgozik` adattagja. Legyen egy `static` válatozója is, ami azt fogja megadni, hogy hány olyan kidobó van, aki dolgozik éppen. Ez alapból 0. Új kidobó példányok konstruktorába rakjuk ennek a változónak a növelését, és változtassuk mindig megfelelen, ha a dolgozik változó értéke változhat (setter, `szolgalatbaAll()`, `szolgalatbolKilep()`). Legyen tehát `szolgalatbaAll()`, és `szolgalatbolKilep()`metódusa is, amik a dolgozik értékét állítják. Írjuk felül az `iszik` függvényét úgy, hogy ellenrizze le, az adott kidobó éppen dolgozik-e, mivel szolgálatban nem ihat alkoholtartalmú italt. Azonban 0 alkoholtartalmút igen. Ha nincs szolgálatban, hívjuk meg az `Ember` (tehát super) `iszik`függvényét.

A `Diák`, és a `Kidobó` is rendelkezzen paraméteres konstruktorral.

A `Diák`, `Kocsmáros`, és `Kidobó` `toString()`-jét írjuk felül, hívjuk meg benne az `Ember` `toString()`-jét is, valamint írjuk ki az adott osztályra jellemz egyéb adatokat is.

Az `Ember` rendelkezzen egy `kötözködik(Ember kivel)` metódussal, ami egy másik embert vár paraméterül. Le kell ellenrizni, hogy van-e szolgálatban lév kidobó, ha igen, az els embert hazaküldjük. Ha nincs, kiírjuk, hogy jót kötözködött.

Italok

Az `Ital` osztályból örököltessünk egy `Sör`, `Bor`, és egy `Kevert` osztályt.

A Bor rendelkezzen egy privát `int évjárat` változóval. A `Sör`nek ne legyen új változója. Ezek kapjanak megfelel konstruktort.

A `Kevert` osztálynak legyen 2, 3, 4 paraméteres konstruktora is, ami italokat vár paraméterül, például: `public Kevert(Ital it1, Ital it2, Ital it3, Ital it4)`

Ezek az italok lehetnek sima `Ital`, `Sör`, vagy `Bor` tipusúak, ezt nem kell külön lekezelnünk, mert erre is jó az örökldés, mivel ezek mind kezelhetk `Ital`ként.

A konstruktor a `Kevert` ital alkoholtartalmát a megadott italok átlag-alkoholtartalmára állítsa be. Egy példa a konstruktor hívására:

```
Kevert kevert1 = new Kevert(sor1, sor2, bor1, ital1);
```

Minden osztály privát adattagjainak legyenek getterei és setterei.

A main függvényben játsszunk kicsit a beépített új funkciókkal is.