Java és web programozás

Kovács Kristóf

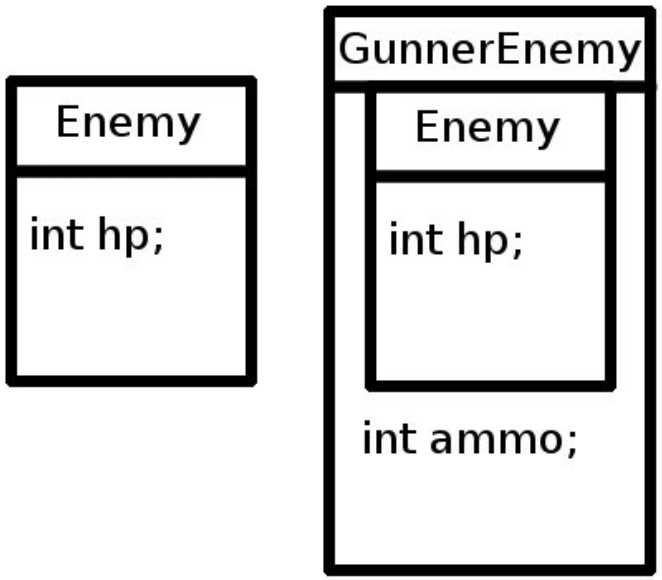
Budapesti Műszaki Egyetem

2015. 02. 11.

4. Előadás

Öröklődés

- Hasonló módon működik javában az öröklődés, mint pythonban (vagy az összes többi programnyelvben).
- Egy B osztály, ami örököl egy A osztályból rendelkezik minden adattaggal és metódussal, ami az A osztályban volt.
- Ezeket az örökölt metódusokat átdefiniálhatjuk az **@Override** kulcsszóval.
- Öröklést jelölni az **extends** kulcsszóval tudunk, a példából látszani fog a használata.



Példa

```
public class Enemy {
    protected int hp;

    public Enemy(int hitpoints) {
        hp = hitpoints;
    }

    public void printHPpub() {
        System.out.println("Enemy pub: " +  hp);
    }
    protected void printHPprot() {
        System.out.println("Enemy prot: " +  hp);
    }
    private void printHPpriv() {
        System.out.println("Enemy priv: " +  hp);
    }
}
```

Példa

```
public class GunnerEnemy extends Enemy {
    protected int ammo;
    public GunnerEnemy(int hp, int am) {
        super(hp); //super hívja az ősosztály konstruktorát
        ammo = am;
    }
    public void printAM() {
        System.out.println("GunnerEnemy ammo: " + ammo);
    }
    @Override
    public void printHPpub() {
        System.out.println("GunnerEnemy pub: "+hp+" "+ammo);
    }
    @Override
    protected void printHPprot() {
        System.out.println("GunnerEnemy prot: "+hp+" "+ammo);
    }
```

Példa

```
public class Main {
    public static void main(String[] args) {
        Enemy e1 = new Enemy(5);
        GunnerEnemy g1 = new GunnerEnemy(2, 3);
        e1.printHPpub();
        e1.printHPprot();
        // e1.printHPpriv();
        System.out.println();
        g1.printHPpub();
        g1.printHPprot();
        // g1.printHPpriv();
        g1.printAM();
        System.out.println();
}
```

```
        Enemy g2 = new GunnerEnemy(6, 1);
        g2.printHPpub();
        g2.printHPprot();
        // g2.printHPpriv();
        // g2.printAM();
        Enemy[] t = new Enemy[3];
        t[0] = e1;
        t[1] = g1;
        t[2] = g2;
    }
}
```

Példa (kommentezve)

```
public class Main {
    public static void main(String[] args) {
        Enemy e1 = new Enemy(5);
        GunnerEnemy g1 = new GunnerEnemy(2, 3);
        e1.printHPpub();
        e1.printHPprot();   // Ha a Main nem egy csomagban
        // e1.printHPpriv(); // van mint az Enemy-vel akkor
        System.out.println(); // a protectedet se érjük el
        g1.printHPpub();   // itt az átdefiniált metódusok
        g1.printHPprot();  // futnak le
        // g1.printHPpriv(); // ez továbbra is privát
        g1.printAM();
        System.out.println();
}
```

Példa (kommentezve)

```
        Enemy g2 = new GunnerEnemy(6, 1); // leszármazottra
        g2.printHPpub();    // referálhatunk az őszosztály
        g2.printHPprot();   // változójával
        // g2.printHPpriv(); // viszont ekkor nem érjük el
        // g2.printAM();   // csak az Enemy részeit
        Enemy[] t = new Enemy[3];
        t[0] = e1;
        t[1] = g1;  // Ezt az ősosztállyal referálást
        t[2] = g2;  // legjobban tárolókkal lehet
    }               // kihasználni, pl tömbökkel
}
```

Eredmény

```
Enemy pub: 5
Enemy prot: 5

GunnerEnemy pub: 2 3
GunnerEnemy prot: 2 3
GunnerEnemy ammo: 3

GunnerEnemy pub: 6 1
GunnerEnemy prot: 6 1
```

Láthatóság

Láthatóság:

- **public:** mindenki látja, mindenki elérheti
- **protected:** csomagon belüliek látják, leszármazott látja
- **default:** csomagon belüliek látják (ezesetben nem írunk ki semmit)
- **private:** csak az adott osztályon belül látható

Láthatóság	Osztály	Csomag	Leszármazott	Mindenki
public	X	X	X	X
protected	X	X	X	
default	X	X		
private	X			

Interface-ek

- Az interface-eket képzeljük úgy, hogy valamilyen tulajdonságot adnak az osztályoknak.
- A célja, hogy hordozhatóbb kódot tudjunk írni, valamint később tanult módszerekhez elengedhetetlen lesz.

Interface-ek folytatás

- Az interface-eknek csak konstansai és metódus deklarálásai lehetnek.
- Deklarálás csak és definíció nem. Az interface csak azt mondja meg, hogy milyen bemenetei vannak a metódusnak, milyen típusú a visszatérési érteke és mi a metódus neve.
- Amikor egy osztály *implementálja* az adott interface-t, akkor kötelező megírnia az interface-ben szereplő metódusokat.

Konkrét példa

```
public interface Controllable {
    public boolean moveForward(double meters,
            double velocity);
    public boolean turnLeft(double degrees,
            double angularVelocity);
    ...
}
```

- A ... nem a szintaktika része, csak jeleztem, hogy a többi metódust is megírhatnánk.
- Hasonlóan hozunk létre interface-t mint osztályt, azzal a különbséggel, hogy itt a metodusoknak nincs kapcsos zárojeles blokkja, hanem pontosvesszővel zárjuk a deklarációjukat.

Implementálás

- Most, hogy már van interface-ünk implementálni kellene egy osztályban. Ezt az **implements** kulcsszóval tehetjük:

```
public class SpaceShip implements Controllable {
    ....
    public boolean moveForward(double meters,
            double velocity) {
        // TODO
        return false;
    }

    public boolean turnLeft(double degrees,
            double angVelocity) {
        // TODO
        return false;
    }
    ....
}
```

Rosszul implementált interface

- Ha nem definiáljuk egy implementált interface metódusat az osztályban, akkor futni se fut a programunk.
- Ezzel garantálja, hogy ha valamit egy adott interface-el láttunk el, akkor ő tud is úgy működni, mint ahogy azt az interface megmondja.

Rosszul implementált interface

- Ha nem definiáljuk egy implementált interface metódusat az osztályban, akkor futni se fut a programunk.
- Ezzel garantálja, hogy ha valamit egy adott interface-el láttunk el, akkor ő tud is úgy működni, mint ahogy azt az interface megmondja.

- A célunk sose az legyen, hogy egy nagy interface-t csináljunk, hanem inkább sok önálló kicsit (minél kevesebb metódussal).

Mire lesz jó nekünk

- Nekünk igaz nem kell sok emberrel összehangolni a kódjaink működését, de így is nagyon fontos az interface-ek használata.
- Képzeljük el, hogy az összes kirajzolandó dolog egy játékban el van látva a *Drawable* interface-el.
- Milyen kényelmesen tudnánk ezután meghívni egy ciklusban a *draw* metódusukat, ha mind egy tárolóban vannak.
- A *draw* máshogy működhet aszteroidáknál, az ûrhajónál és bármi másnál, de ez nem számít. Meg van írva mindegyik osztályban és csak ez a lényeges.

Példa interface-el tárolt objektumokra

```
public interface Drawable {
   public void draw();
}

public class Main {
   public static void main(String[] args) {
      Drawable drawables[] = new Drawable[3];

      drawables[0] = new SpaceShip(...); // ezek
      drawables[1] = new Asteroid(...);  // implementálják
      drawables[2] = new Astaroid(...);  // a Drawable
                                         // interface-t
      for(Drawable d : drawables) {
         d.draw();
      }
   }
}
```

- Amit nem tehetek meg, hogy olyan metódust hívok meg amit nem a megadott interface implementál. Pl:

```
for (Drawable d : drawables) {
    d.move(1.1, 2.2);
}
```

- Még akkor sem lehet ezt megtenni, ha mindegyik drawable-t implementáló osztálynak van *move* metódusa. Mert mi *Drawable*-ként hivatkozunk rájuk és amíg őket *Drawable*-ként látjuk addig csak olyat tudunk tenni amit biztosan tud *Drawable* is.

- Lehet egy osztálynak több interface-e is, és akkor is mindet implementálni kell, ami az interface-ekben van. Pl:

```
public class SpaceShip implements Drawable,
        Controllable, Destroyable {

    public void turnLeft(...) {
        ...
    }
    ...

    public void draw() {
        ...
    }

    public void destoy() {
        ...
    }
}
```

Interface öröklés

- interface-eknél is létezik öröklés
- az öröklődési szabályok ugyanazok, mint osztályoknál, csak természetesen nincsenek adattagok
- itt a függvények átdefiniálásának nincs értelme, mivel csak deklaráljuk a függvényeket az interface-ekben
- az viszont teljesül, hogy egy J interface-t implementáló osztályra, hivatkozhatunk I-ként, de ekkor csak azon metódusait használhatjuk, amik az I-ben vannak

```
public interface J extends I {
    public void method();
}
```

Interface öröklés

- interface-eknél is létezik öröklés
- az öröklődési szabályok ugyanazok, mint osztályoknál, csak természetesen nincsenek adattagok
- itt a függvények átdefiniálásának nincs értelme, mivel csak deklaráljuk a függvényeket az interface-ekben
- az viszont teljesül, hogy egy J interface-t implementáló osztályra, hivatkozhatunk I-ként, de ekkor csak azon metódusait használhatjuk, amik az I-ben vannak

```
public interface J extends I {
    public void method();
}
```

- Az interface-ek is öröklődnek. Így ha az A osztaly implementál egy I interface-t, akkor a leszármazott B osztálya is automatikusan implementálja.