Java és web programozás

Kovács Kristóf, Rimay Zoé

Budapesti Műszaki Egyetem

2013. október 16.

6. 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.
- Az interface-ek is öröklődnek. Így ha az A osztaly implementál egy I interface-t, akkor a leszármazott B osztálya is automatikusan implementálja.

Láthatóság:

- *public:* mindenki látja, mindenki elérheti
- *protected:* csomagon belüliek látják, leszármazott látja
- *private:* csak az adott osztályon belül látható

Példa

```
public class A implements I {
    public int adat;
    public A(int i) {
        adat = i;
    }
    public void fv1() {
        System.out.println("fv1: " +  adat);
    }
    protected void fv2() {
        System.out.println("fv2: " +  adat);
    }
    private void fv3() {
        System.out.println("fv3: " +  adat);
    }
}
```

Példa

```
public class B extends A {
    public int adatB;
    public B(int i, int j) {
        super(i); //super hívja az õsosztály konstruktorát
        adatB = j;
    }
    public void fv4() {
        System.out.println("Bfv4: " +  adat);
    }
    @Override
    public void fv1() {
        System.out.println("Bfv1: " +  adat + " " + adatB);
    }
    @Override
    protected void fv2() {
        System.out.println("Bfv2: " +  adat + " " + adatB);
    }
```

Példa

```
public interface I {
    public void fv1();
}

public class Main {
    public static void main(String[] args) {
        A a1 = new A(5);
        B b1 = new B(2, 3);
        A b2 = new B(6, 1);
        a1.fv1();
        a1.fv2();
        // a1.fv3();
        System.out.println();
        b1.fv1();
        b1.fv2();
        // b1.fv3();
        b1.fv4();
```

Példa

```
        System.out.println();
        b2.fv1();
        b2.fv2();
        // b2.fv3();
        // b2.fv4();
        I[] tomb = new I[3];
        tomb[0] = a1;
        tomb[1] = b1;
        tomb[2] = b2;
    }
}
```

Példa (kommentezve)

```
public interface I {
    public void fv1();
}

public class Main {
    public static void main(String[] args) {
        A a1 = new A(5);
        B b1 = new B(2, 3);
        A b2 = new B(6, 1); // beleférA-ba egy B objektum
        a1.fv1();
        a1.fv2(); // Ha nem egy csomagba vannak ez se műxik
        // a1.fv3();  // Ez privát azért nem lehet hívni
        System.out.println();
        b1.fv1(); // Ez privát azért nem lehet hívni
        b1.fv2();// Ha nem egy csomagba vannak ez nem műxik
        // b1.fv3(); // Ez privát azért nem lehet hívni
        b1.fv4();
```

Példa (kommentezve)

```
        System.out.println();
        b2.fv1();
        b2.fv2(); // Ha nem egy csomagba vannak ez se műxik
        // b2.fv3(); // Ez privát azért nem lehet hívni
        // b2.fv4(); // A-val referálunk ra ezért nem műxik
        I[] tomb = new I[3]; // az interface-el tömb
        tomb[0] = a1; // belefér mindegyik
        tomb[1] = b1;
        tomb[2] = b2;
    }
}
```

Eredmény

```
fv1: 5
fv2: 5

Bfv1: 2 3
Bfv2: 2 3
Bfv4: 2

Bfv1: 6 1
Bfv2: 6 1
```

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 fv5();
}
```

Kivétel kezelés

- a kivétel (exception) egy esemény, mely futás közben megbontja a program normális futási folyamatát
- például kivétel *dobódik* amikor 0-val osztunk
- a kivételeket lekezelhetjük, hogy ne a teljes program szálljon el tőle, ezt úgy szokták mondani, hogy *elkapjuk* (catch) a kivételt
- kivételeket elkapni *try* blokkokon belül tudunk, az ehhez tartozó *catch* blokk fut le ha megfelelő a paramétere
- megfelelő a paraméter, ha igaz, hogy az adott paraméter olyan tipusú mint amivel lehet a dobott kivételre hivatkozni

Kivétel kezelés

- a kivétel (exception) egy esemény, mely futás közben megbontja a program normális futási folyamatát
- például kivétel *dobódik* amikor 0-val osztunk
- a kivételeket lekezelhetjük, hogy ne a teljes program szálljon el tőle, ezt úgy szokták mondani, hogy *elkapjuk* (catch) a kivételt
- kivételeket elkapni *try* blokkokon belül tudunk, az ehhez tartozó *catch* blokk fut le ha megfelelő a paramétere
- megfelelő a paraméter, ha igaz, hogy az adott paraméter olyan tipusú mint amivel lehet a dobott kivételre hivatkozni
- minden kivétel az *Exception* osztályból örököl, így ezzel a paraméterrel minden kivételt elkaphatunk
- például amikor nullával való osztás történik a számológép programunkban, akkor félbehagyjuk a számolást kiírjuk, hogy hiba és várunk új parancsot
- a kivételeket mindig a legközelebbi megfelelő *catch* blokkban kezeli le, ha nincs ilyen blokk akkor elszáll a teljes program

Példa

- Példa, képzeljük el, hogy az alábbi osztályban a matematikai függvények jól meg vannak írva:

```
public class Pelda {
    public static void szamol() throws Exception {
        double d = sqrt(5 * log(sin(6) + cos(4 / 0)));
    }

    public static void main(String[] args) {
        try {
            szamol();
        } catch (Exception e) {
            System.out.println("elkapva");
        }
    }
}
```

- amint láthatjátok, ha egy metódus továbbdobhat egy kivételt azt jeleznünk kell a *throws* kulcsszóval

Példa magyarázat

- a kivételt a nullával való osztás dobja
- ha a *cos* függvény lekezeli (a *try* blokkban van a számolás és a *catch* blokkja kompatibilis a kivétellel, akkor minden ok, ott le volt kezelve a kivétel)
- ha a *cos* nem kezeli le, akkor a kivétel tovabb dobódik
- ekkor a *log*nak van esélye elkapni
- majd az *sqrt*nek
- a *szamol* függvénynek, de ennek láthatjuk is, hogy nincs *try* blokkja így automatikusan tovább dobódik
- és végül a *main* kezelheti le
- ha egy metódus továbbdobhat egy kivételt azt jeleznünk kell a *throws* kulcsszóval

Saját kivétel

- csinálhatunk saját kivétel osztályt, annyit kell csak tenni, hogy egy eleve létező kivételből kell örököltetni

```
public class MyException extends Exception {
    public String messege;
    public MyException(String m) {
        messege = m;
    }
}
```

Példa2

```
public class Pelda2 {
    public static void dobo() throws MyException {
        throw new MyException("uzenet");
    }

    public static void main(String[] args) {
        try {
            dobo();
        } catch (MyException me) {
            System.out.println(me.messege);
        }
    }
}
```

Példa2 magyarázat

- a *MyException* kivételnek van egy *messege* adattagja, így amikor elkapjuk ezt az adattagot elérhetjük
- egy *catch*en belül tovabb is dobhatjuk a kivételt a *throw* kulcsszóval:

```
catch (MyException me) {
    if (me.messege.equals("nemitt")) {
        throw me
    }
}
```

- ekkor viszont ne feledjük, hogy a *main*nek meg kell mondani, hogy dobhat *MyException* kivételt

```
public static void main(String[] args) throws MyException {
    ...
```