
Java VII.

Polimorfizmus a Java nyelvben

Ficsor Lajos

Miskolci Egyetem

Általános Informatikai Tanszék

Utolsó módosítás: 2005. 10. 19.

A kötés (binding) fogalma

- Kötés (binding) fogalma: egy metódus híváshoz a megfelelő definíció megtalálása.
- Ehhez a metódusnév nem elég.
- Ha az aktuális objektum osztályában a név egyedi (beleértve az örökölt metódusokat is), a hatáskör és a név elegendő az azonosításhoz.
- Az objektum statikus típusa alapján az azonosítást a fordítóprogram el tudja végezni.
 - Ez **korai kötés** (early binding).
 - Nem polimorfizmus, mert a név egyedi.

A poliformizmus implementációi

- Metódusnév túlterhelés
 - A már ismert mechanizmus: egy osztályon belül azonos nevű függvények, eltérő paraméter szignatúrával.
 - A saját és az örökölt függvények együttesen tekintendők.
 - **Korai kötés**: a fordítóprogram az aktuális paraméterek *statikus típusa* alapján dönteni tud.
 - **Polimorfizmus**, mert a név nem egyedi, további információkat kell felhasználni.

A poliformizmus implementációi (folyt.)

- Metódus felüldefiniálás
 - Az előzőnél még hatékonyabb implementációs forma.
 - A leszármazott osztály az ős osztálytól örökölt metódust felüldefiniálhatja.
 - Egy ilyen metódus hívásánál dönteni kell, hogy az örökölt vagy a saját változat hívódjon meg - **polimorfizmus**.
 - A döntés alapja a hivatkozás **dinamikus** típusa.
 - Mivel a dinamikus típus fordítási időben nem ismert, a felüldefiniált metódusok közötti választást futásidőre kell halasztani - **késői kötés**.

Metódus felüldefiniálás - alapszabályok

- Egy ős osztálybeli metódus felüldefiniálásához a következő feltételeknek kell teljesülnie:
 - A felüldefiniáló metódus **visszatérési típusának, nevének, és paraméter szignatúrájának** meg kell egyeznie az ős osztálybeli metódusával.
 - A felüldefiniáló metódus hozzáférési kategóriája nem lehet szűkebb az eredeti metódusénál. (Bővebb lehet.)
 - A felüldefiniáló metódus csak olyan ellenőrzött kivételeket válthat ki, amelyeneket az eredeti is kiválthat. (Pontos jelentése később.)

Felüldefiniált metódus hívása

- A hívásban szereplő valamennyi információ illik minden metódus változatra - ez alapján nem lehet dönteni.
- A döntés alapja a hivatkozás dinamikus típusa.
- A döntés csak futás időben történhet.
- A felüldefiniáló metódus az ős osztály metódusát elérheti a

`super.metódusnév( . . . )`

formájú hivatkozással.

Egyszerű példa

```
package jarmu3;  
public class jarmu {  
    private int kerekek;  
    private double suly;  
    public jarmu(int k, double s)  
        { kerekek = k; suly = s; }  
    public int kerekszam ()  
        { return kerekek; }  
    public double kerekterheles()  
        {return suly / kerekek; };  
    public double sulya()    {return suly;}  
}
```

Egyszerű példa (folyt.)

```
package jarmu3;  
public class gepkocsi extends jarmu {  
    private int személyek;  
    public gepkocsi(int k, double s,  
        int szem) {  
        super(k, s);  
        személyek = szem; }  
    public double kerekterheles()  
        {return sulya() + személyek*80 /  
            kerekszam(); }  
}
```

Egyszerű példa (folyt.)

```
package jarmu3;  
public class proba {  
    public static void main  
        (String[] args) {  
jarmu bicikli = new jarmu(2, 15.0);  
System.out.print  
    ("Bicikli kerekterhelese: ");  
System.out.println  
    (bicikli.kerekterheles());  
}
```

Egyszerű példa (folyt.)

- Futás eredmény:

Bicikli kerekterhelese: 7.5

- A **jarmu2** példához képest semmi változás
- Az ok, hogy a **bicikli** referencia statikus és dinamikus típusa azonos.

Egyszerű példa (folyt.)

```
jarmu trabant =  
    new gepkocsi(4, 600.0, 4);  
System.out.print  
    ("Trabant kerekterhelese: ");  
System.out.println  
    (trabant.kerekterheles());  
} // main vege  
} // class proba vege
```

Egyszerű példa (folyt.)

- Futás eredmény:

Trabant kerekterhelese: 680.0

- A **gepkocsi** osztályban definiált **kerekterheles** metódus hívódik meg, a dinamikus típusnak megfelelően.

A késői kötés használata

- Az leszámaztatás lehetőséget teremt, hogy viselkedésformákat örököljön egy osztály.
- Bizonyos esetekben a változatlanul öröklődő viselkedés nem felel meg a leszámazottnak.
- A felüldefiniálás lehetősége ezt a problémát tudja megoldani.
- A késői kötés automatizmusa a használatot kényelmessé teszi.

Metódus felüldefiniálás - további szabályok

- Nem kötelező a leszármazás minden szintjén felüldefiniálni a metódust.
 - Egy osztály örökölheti a felüldefiniált metódust.
- Statikus metódus nem definiálható felül.
(Értelmetlen lenne, mert hívása a statikus típus alapján történik.)
 - Ugyanolyan nevű statikus metódus a leszármazott osztályban elfedi az őс osztály metódusát.

Példa: grafikus editor

- Megvalósítandó egy grafikus editor.
- Egyszerűsítések:
 - síkbeli alakzatok,
 - nincsenek színek.
- A rajzfelületen megjelenő egységek **objektumok**, mert
 - vannak adatszerű tulajdonságai (hely, méretek stb.)
 - vannak olyan műveletek, amelyek elvégezhetők rajtuk (transzformációk stb.).
- Az objektum orientált szemlélet hasznos.

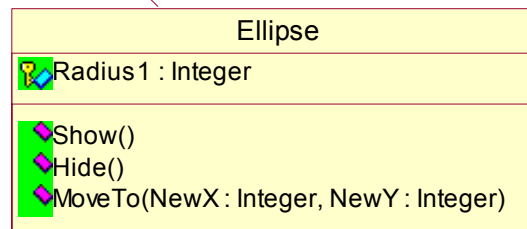
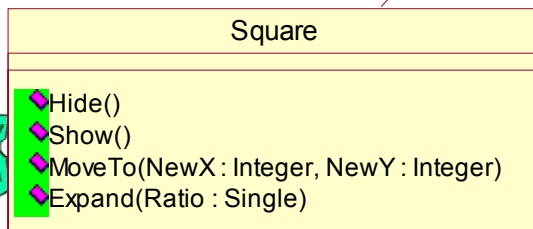
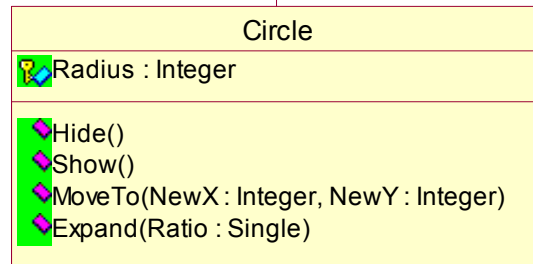
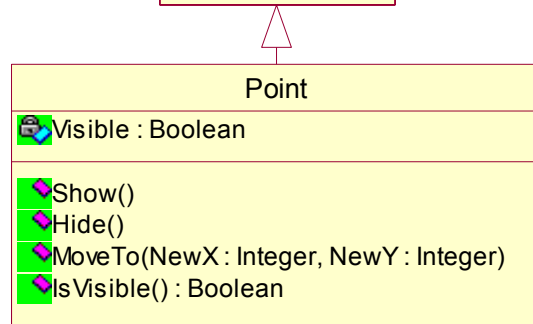
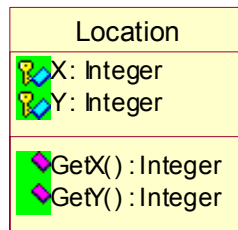
Példa: tervezési döntések

- Minden objektum helyzetét egy referencia pontja, és ha szükséges, egyéb adatok (például szög) határozza meg.
- Az egyes síkidomok közös tulajdonságait megfigyelve egy öröklődési hierarchia építhető fel.
 - A közös tulajdonságokat és viselkedés mintákat csak egyszer kell implementálni, a leszármazottak készen kapják.

Példa: tervezési döntések (folyt.)

- Minden objektumon elvégezhetőek legyenek az alapvető geometriai transzformációk.
 - Például eltolás, forgatás, átméretezés stb.
- A transzformációk végrehajtásáért maguk az objektumok legyenek felelősek.
 - A megfelelő metódusok fogják végrehajtani.
 - A transzformációs metódusok is öröklődhetnek => kód újrafelhasználás.
- Ezen döntések alapján az osztályterv egy első változata:

Példa: osztályterv (1. verzió)



Figyelem!

Terjedelmi okokból nem teljes.

Első változat, azaz

- felülvizsgálatra,

- finomításra

szorul.

Példa: osztályterv finomítása

- Nézzük meg a **MoveTo** implementációit.
- **Point**:

```
void MoveTo(int NewX, int NewY) {  
    Hide();    // aktuális törlése  
    X = NewX;  // új pozíció  
    Y = NewY;  
    Show();    // megjelenítés az új helyen  
}
```

Példa: osztályterv finomítása (folyt.)

- Circle:

```
void MoveTo(int NewX, int NewY) {  
    Hide();    // aktuális törlése  
    X = NewX;  // új pozíció  
    Y = NewY;  
    Show();    // megjelenítés az új helyen  
}
```

- A két implementáció azonosnak tűnik, mert azonos az algoritmus:
 - törlés az aktuális pozícióban,
 - referencia pont módosítása,
 - megjelenítés.

Példa: osztályterv finomítása (folyt.)

A különbség:

- a **Point** osztály **MoveTo** metódusában a **Point** **Hide()**-ja hívódik meg
- a **Circle** osztály **MoveTo** metódusában a **Circle** **Hide()**-ja hívódik meg
- Az algoritmus bármilyen alakzatra ugyanaz.
- Kérdés: örökölhető-e a **MoveTo** , vagy külön implementáció szükséges minden osztályra?

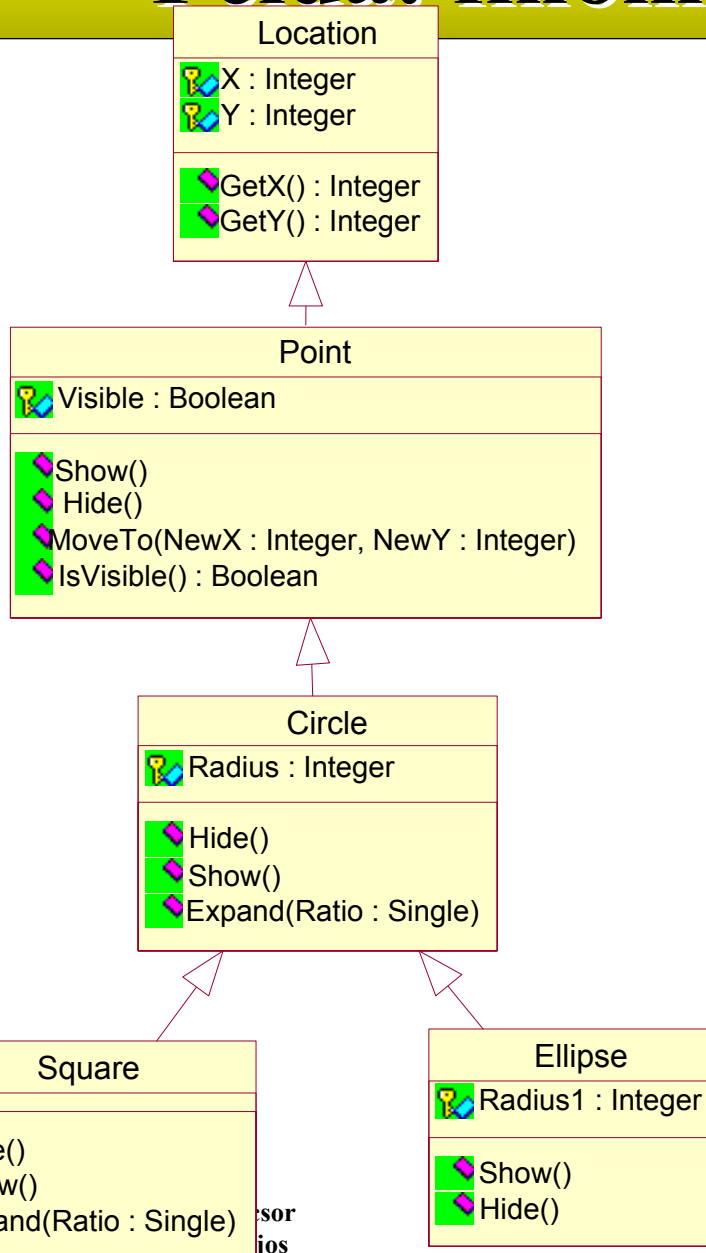
Példa: osztályterv finomítása (folyt.)

A válasz: **örökölhető!**

A magyarázat:

- A **MoveTo** metódus a **this.Show** és **this.Hide** metódusokat hívja meg.
- A **Show** és **Hide** metódust minden osztály felüldefiniálja (hiszen működésük az alakzattól függ).
- A **this** pszeudóváltozó dinamikus típusa (amellyel a **MoveTo**-t használtuk) dönt a felüldefiniált **Show** és **Hide** metódusok között.

Példa: finomított osztályterv



Figyelem!

Terjedelmi okokból nem teljes.

Második változat, azaz jobb, de még lehet javítani.

Példa: további implementációs részletek

- Létrehozott rajzelemek tárolása:

```
final int MAXELEM=1000;  
int aktElemszam = 0;  
Location elemek[];  
elemek = new Location[MAXELEM] ;  
...  
// Új rajzelem létrehozása  
aktElemszam++;  
elemek[aktElemszam] = new .....
```

Példa: további implementációs részletek

```
// Osszes elem ujrarajzolasa  
for (int i=0; i<aktElemszam; i++)  
    elemek[i].Show();
```

Megjegyzés:

- Természetesen a rajzelemek tömbben való tárolása ügyetlen (elegánsabban szólva *naív*) megoldás.
- A helyes megoldás egy dinamikusabb adatszerkezet (például lista) alkalmazása lenne.

final metódus

- Egy metódus kaphat **final** minősítőt.
- A **final** minősítésű metódust nem definiálhatja felül egyetlen leszármazott osztály sem.
- Szerepe, hogy megakadályozza bizonyos viselkedés formák megváltoztatását, ha az veszélyezteti a helyes működést.

Absztrakt metódus és osztály

- Gyakran előfordul a tervezés során, hogy egy osztály szintjén tudjuk, hogy valamilyen metódus szükséges lesz a leszármazottakban, de még nem lehet megadni az implementációját.
- Ezért a Java nyelv megengedi törzs nélküli metódus definiálását.
- Az ilyen metódust az **abstract** minősítővel kell ellátni.
- Ha az osztály tartalmaz absztrakt metódust, az osztályt is az **abstract** minősítővel kell ellátni.

Absztrakt metódus és osztály (folyt.)

Répási tanár úr kedvenc példáját idézve:

```
public abstract class Sikidom {  
    public abstract double terület();  
    public abstract double kerulet();  
}
```

- A fenti osztálydefiníció az alábbi tervezési meggondolásokat fejezi ki:
 - minden síkidomnak van területe és kerülete,
 - nincs algoritmus a terület és kerület számítására a síkidom tulajdonságainak ismerete nélkül.

Absztrakt metódus és osztály (folyt.)

Formai szabályok:

- Absztrakt egy metódus, ha nincs törzse.
Megvalósítást (törzset), majd csak a felüldefiniálás során kap.
- Absztrakt metódusnak nem lehet módosítója a **private**, **final**, **static** hiszen az ilyen metódusokat nem lehet felüldefiniálni.
- Absztrakt egy osztály, ha van legalább egy absztrakt metódusa.

Absztrakt metódus és osztály (folyt.)

- Absztrakt osztályt nem lehet példányosítani.
- Egy absztrakt osztály arra szolgál, hogy ős osztálya legyen további osztályoknak.
- A leszármazott osztály(ok) feladata az absztrakt metódusok felüldefiniálása.
- Absztrakt osztály gyermeke lehet absztrakt, ha nem minden absztrakt metódust valósít meg.
- Az absztrakt osztály is használható referencia statikus típusaként.

Absztrakt metódus és osztály (folyt.)

Az absztrakt metódusok szerepe:

- Rögzít egy tervezési döntést (szükséges metódusok halmaza).
- Kényszeríti a leszármazott osztály(ok) programozóját meghatározott metódusok definiálására.

Absztrakt metódus és osztály (folyt.)

Hibalehetőségek:

- Törzs nélküli metódus, **abstract** minősítő nélkül.
- Absztrakt metódust tartalmazó osztály **abstract** minősítő nélkül.
- A fordítóprogram figyel az ilyen hibákat.