Osztálytervezés és implementációs ajánlások

Ficsor Lajos

Miskolci Egyetem

Általános Informatikai Tanszék

Utolsó módosítás: 2006. 04. 24.

Osztály tervezés

- Egy nyelv szabályrendszere számos konstrukciót megenged
- A software fejlesztés során egy adott problémát kell megoldani
- Az objektum orientált program a valóság valamely részének a modellje
- Szükséges tanulmányozni, hogy adott cél elérése érdekében milyen nyelvi szerkezet(ek) használata a legcélszerűbb

Mi is az osztály?

- Programozás technikai szempontból ***típus.***
- Programtervezési szempontból a modellezendő valóság valamely fogalmának modellje.
- A programokban kétféle osztályok lehetnek:
  - közvetlenül az alkalmazási terület (***application domain***) fogalmait ábrázoló osztályok
  - a programok elkészítéséhez szükséges osztályok

Mit ábrázolhatnak az osztályok?

Alkalmazási terület osztályai:

- Felhasználói fogalmak (pl. teherauto)
- Felhasználói fogalmak általánosításai (pl. jármű)

A megvalósítás osztályai:

- Hardware/software erőforrások (pl. memória, i/o)
- Más osztályok implementálásához szükséges osztályok (pl. listák, vermek stb.)
- Beépített adattípusok és vezérlési szerkezetek

Információ rejtés

- A jól tervezett osztály belső részleteinek ismerete szükségtelen a használatához
- Az osztály használata csak annak **interface**-e segítségével lehetséges.
- Egy objektum a külvilággal csak az **interface**-en keresztül képes kommunikálni.

Osztály interface fogalma

- A **`public`** metódusok összessége.
  - Ezeket kell ismernie az osztály használójának.
  - Használatukhoz nem szükséges ismerni az osztály implementációs részleteit.
- **`protected`** metódusok és adattagok.
  - Kibővíti az interface-t a leszármazott osztályok számára
  - Használata veszélyeket rejt magában, mert implementációs függést hoz létre az ős és a leszármazott osztály között

Osztály interface fogalma (folyt.)

- **Technikai szempontból** az interface részét képezik az esetleges **`public`** minősítésű adattagok is, de **használatuk nem ajánlott**.
  - Ellentmond az információ rejtésnek.
  - A tárgy keretein belül a a fenti "**nem ajánlott**" kifejezés jelentése: "**tilos**"!

A jó osztály interface

Teljes

- minden funkciót tartalmaz, ami az osztálytól elvárható
- nem az adott alkalmazás szempontjai határozzák meg
- újrafelhasználható egységet alkot az osztály

A jó osztály interface (folyt.)

Minimális

- nem tartalmaz a felhasználó számára érdektelen (esetleg veszélyes) elemeket
- belső felhasználású funkciók **`private`** vagy **`protected`** minősítésűek
- a belső áttervezés nincs rá hatással

A jó osztály interface (folyt.)

Kezelhető méretű

- általában legfeljebb néhány tíz metódus
- A sok funkció között nagyobb valószínűséggel lesznek hasonlóak (félreértés veszélye!)
- A terjedelmes interface általában tervezési hibára utal:
  - Az interface része belső funkció is

A jó osztály interface (folyt.)

- terjedelmes interface (folytatás)
  - Az osztály határait nem jól állapítottuk meg, és túl sok feladatot akarunk rábízni. A helyes architektúra kialakítása érdekében az eredetileg tervezett osztályt több osztályra kell bontani, és ezek között leszármaztatással vagy más mechanizmussal megteremteni a kapcsolatot.

Az osztály interface részei

- **Kezelő tagok és metódusok**
  - Konstruktorok, örökölt "kész" metódusok (pl. **`toSting, clone`**, stb)
  - Sokszor nem is a programozó, hanem a program implicite hívja meg.
- **Elérési függvények**
  - Az adattagok értékének elérésére vagy azok értékének módosítására.
- **Munkavégző függvények**
  - Az osztály lényegi funkcióit aktivizáló függvények.

Osztályok közötti kapcsolatok

Logikai szintű kapcsolatok

- általánosítás/pontosítás (**is-a**)
- tartalmazás (**has-a**)
- használat (**use**)

Az általánosítás/pontosítás implementálása

- **Leszármaztatási mechanizmus (öröklődés) segítségével**
- A leszármazott osztály objektuma egyben ős objektum is.

Példa: öröklődés

```
public class Szemely { … }
public class Hallgato extends
  Szemely
{…}
public class Proba {
public void sortIszik(Szemely sz );
public void feladatotBead(Hallgato
  h);
```

Példa: öröklődés (folyt.)

```
public static main(String args) {
Hallgato nagypista = new Hallgato();
Szemely kispista = new Szemely();
SortIszik(kispista ); // OK
//OK, mert egy Hallgato egyben Szemely is
SortIszik(nagypista );
// Hibás, mert egy Szemely nem
// biztos, hogy Hallgato is
FeladatotBead( kispista ); }
}
```

Példa: hibás öröklődés

```
public class Madar {
public Kaja mitEszik();
public void Repul();
}
public class Strucc extends Madar
  {…}
```

- **A Strucc örökli a repülés képességét a Madar osztálytól => hibás modell!**

Példa: hibás öröklődés (folyt.)

A hiba oka: a természetes nyelv pontatlansága.

Mit jelent pontosan: "A madár tud repülni"

- **"Minden madár tud repülni"**
- **"A madarak `általában` tudnak repülni"**

Nem a valóság viszonyait modelleztük.

Példa: hibás öröklődés javítása

Egyik lehetséges javítás:

```
public class Madar {
public Kaja MitEszik();
public boolean TudRepulni();
}
class Strucc extends Madar {…}
```

Csak a repülés **lehetőségét** örökli.

- Nem igazi, mert hibás példányok létrehozását teszi lehetővé.

Példa: hibás öröklődés javítása (folyt)

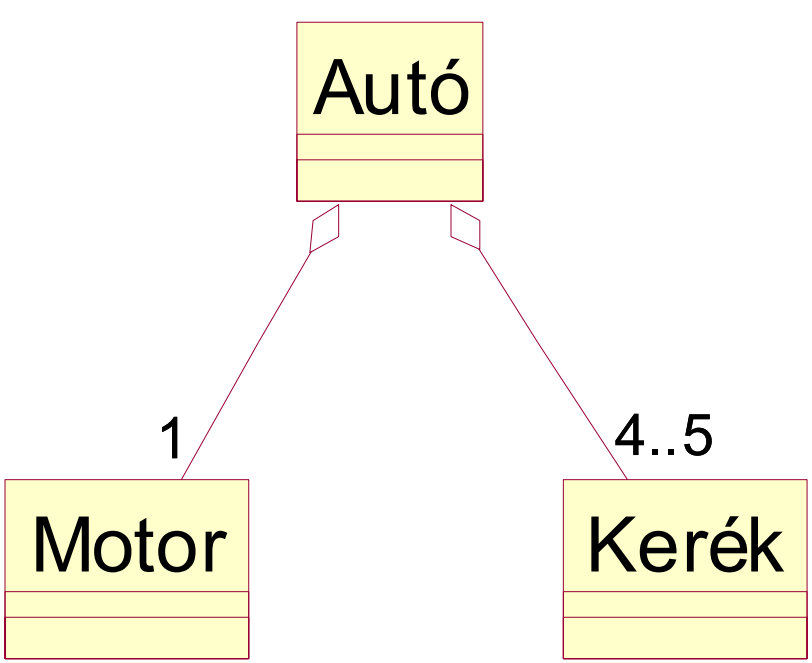
```
public class Madar {
public Kaja mitEszik();
}
public class GyalogMadar extends Madar {…};
public class RendesMadar extends Madar {
public void Repul();
}
public class Strucc extends GyalogMadar
  {...};
class Sas extends RendesMadar{...};
```

A tartalmazás implementálása

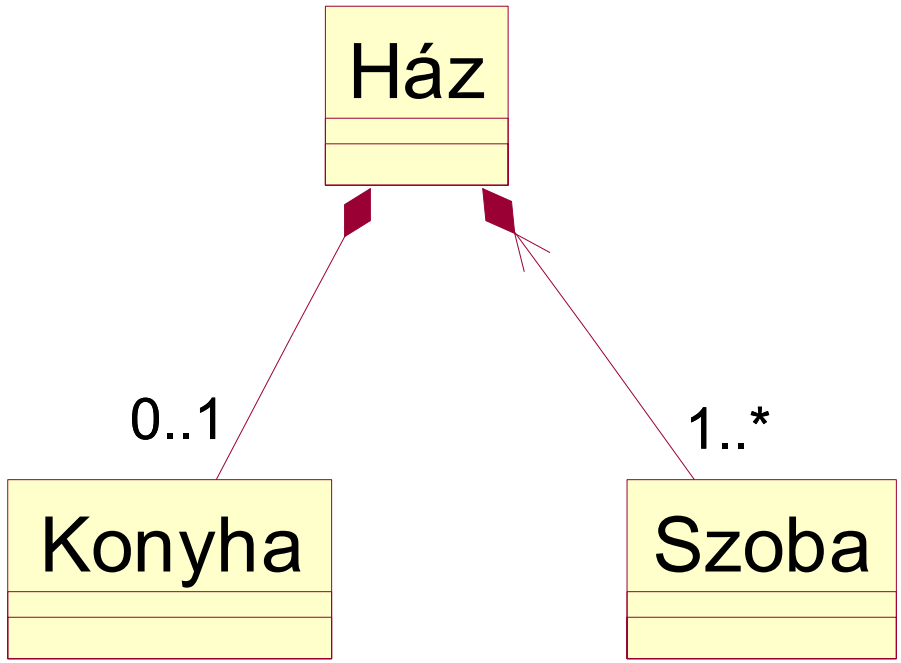
Kétféle tartalmazás kapcsolat:

- **`aggregáció:`** a rész az egészhez tartozik, de önállóan is létező entitás
- **`kompozíció:`** a rész önmagában nem létezhet, csak valaminek a részeként

Tartalmazás: példa és UML jelölés



Aggregáció



Kompozíció

A tartalmazás implementálása: aggregáció

- A tag objektum referenciája a tartalmazó osztályban.
- Ez adattag, tehát általában **`private!`**
- Az egy - több kapcsolat (több "rész") megvalósítása különböző adatszerkezetekkel lehetséges (tömb, **`Vektor`** stb).
- A referenciák beállítása általában a befoglaló osztály konstruktorának feladata, már létező ("külső") objektumok referenciáinak felhasználásával.

Példa: aggregáció implementálása

- Példa:

```
public class Motor { ... }
public class Kerek { ... }
public class Auto  {
private Motor motorja;
private Kerek kerekek[] = new
  Kerek[4];
private Kerek potkerek;
…
}
```

Tartalmazás implementálása: kompozíció

Lehetséges megoldások:

- A tartalmazó osztályban osztálydefiníció a tartalmazott számára, **`private`** hozzáférési kategóriával
- A tartalmazó osztály konstruktorának vagy valamelyik metódusának a feladata a "rész" példányosítása.
  - "Kívülről" nem is lehetséges, a **`private`** minősítés miatt.

Öröklődés vagy tartalmazás?

- Mindkét esetben egy objektum más objektumot tartalmaz.
- Az öröklődés esetén ez implicit módon történik.

Technikai különbségek:

- A leszármazott objektum pontosan egy ősobjektumot tartalmaz.
- Tagobjektumok tetszőleges számú típussal, típusonként tetszőleges számmal definiálhatók.

Öröklődés vagy tartalmazás? (folyt.)

Különbségek tervezési szempontból

- Más logikai kapcsolatot fejeznek ki (**is-a, has-a**).
  - Ha például az **`Auto`** osztályt a **`Motor`** osztály leszármazottjaként definiáljuk, akkor egy **`Auto`** objektumnak lesz egy **`Motor`** része, de nem igaz az, hogy az **`Auto`** egy (speciális) **`Motor`**.
- Az öröklés az **interface újrafelhasználása**: a leszármazott osztály interface-ének része lesz az ősosztály interface-e.

Öröklődés vagy tartalmazás? (folyt.)

Különbségek tervezési szempontból (folytatás)

- A **`private`** tag objektummal az osztályának a funkcióit használjuk fel a befoglaló osztály implementációjához.
  - A tagosztály interface-e nem képezi részét a befoglaló osztály interface-ének.

Öröklődés vagy tartalmazás? (folyt.)

Különbségek tervezési szempontból (folytatás)

- A **`public`** tag objektummal a befoglaló osztály interface-ét kiegészítjük a tag objektumok osztályainak interface-eivel.
  - Nem mindig szerencsés megoldás:
    - Rontja a program áttekinthetőségét.
    - Erős függőséget hoz létre az osztályok között. (Egy tagobjektum osztályának változása nem csak a befoglaló osztály módosítását jelenti, hanem a befoglaló objektum használati pontjainak módosítását is!)

Használat kapcsolat implementálása

Fajtái: (**CX** és **CY** osztályok)

- **CX** használja a **CY** nevet (feltételezi, hogy **CY** hozzáférhető a használat helyén)
  - adattag típusa
  - metódus visszatérési értékének vagy paramétereinek típusa
- **CX** használja a **CY** osztályt (a hozzáférési kategóriák alapján)
  - meghívja a **CY** egy metódusát
  - írja/olvassa a **CY** egy adattagját

Használat kapcsolat impl. (folyt.)

- **`CX`** létrehoz egy **`CY`** típusú objektumot

Probléma a használat kapcsolat implementálásával:

Csak közvetett nyelvi eszközök jelzik az ilyen kapcsolatokat

Korlátozó tényezők:

- csomagok és az osztályok hozzáférési kategóriája
- osztály tagjainak hozzáférési kategóriája