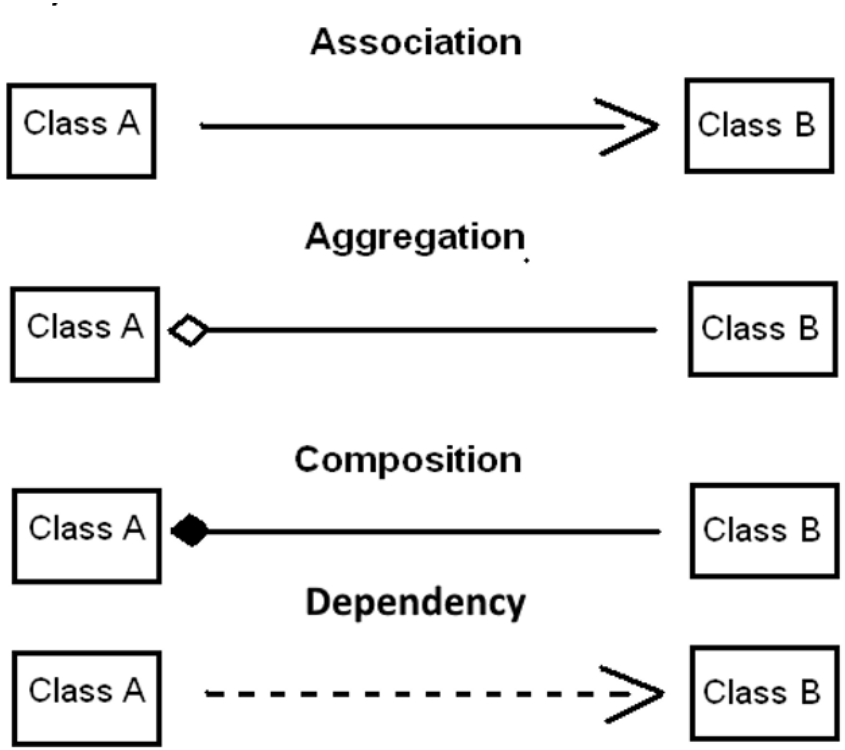
A Programozás Alapjai 2
Objektumorientált szoftverfejlesztés

Dr. Forstner Bertalan
forstner.bertalan@aut.bme.hu

Ismétlés: asszociációs formák, UML

- Asszociáció (association)
  - objektumoknak saját életciklusuk van (egymástól függetlenül léteznek)
  - nincs tulajdonosi viszony
- Aggregáció (aggregation)
  - Ez volt előadáson a „father”
  - asszociáció specializált formája
  - különbség: van tulajdonosi kapcsolat
  - objektumoknak saját életciklusuk van (egymástól függetlenül léteznek)
- Tartalmazás (composition)
  - aggregáció specializált formája
  - különbség: tartalmazó megszűnésekor a tartalmazott is megszűnik
  - erős kapcsolat („death” relationship)
  - a tartalmazottaknak életciklusát a tartalmazó irányítja, így nincs saját életciklusuk



Statikus és konstans tagok

Tagváltozók inicializálása

- Inicializációs lista
- Példa

Tagváltozók inicializálása

- Inicializációs lista
- Példa

- Az inicializálási lista **hamarabb lefut**, mint a **konstruktor törzse**.

```
class Valami1 {
public:
	Valami1() {}
	~Valami1() {}
};

class Valami2 {
public:
	Valami2(int param) {}
	~Valami2() {}
};
```

```
class A {
	int x;
	Valami1& v1;
	Valami2 v2;
public:
	A(int px, Valami1& pv1, int intpar) :x(px), v1(pv1), v2(intpar)
	{
	}
};
```

Statikus tagváltozók

- Példa: euró bankszámla

Statikus tagváltozók

- Példa: euró bankszámla
- Az árfolyam független az egyes bankszámla példányoktól, mindig ugyanannyinak kell lennie
- Azt mondjuk, hogy **„statikus tagváltozó”**

```
static int rate;
```

- Mikor történik meg a helyfoglalás neki? Kézzel kell megoldanunk…

```
int Account::rate = 310;
```

A statikus tagváltozó

- **Egy darab** van belőle az egész osztályra vonatkozóan.
  > Közös az osztály minden objektuma számára, (ugyanaz az értéke).
- Már **azelőtt is létezik**, hogy objektumot hoznánk létre az osztályból.
- Mikor szoktuk használni: amikor minden objektum számára közös változót szeretnénk.

Statikus tagfüggvény

- A példa kiegészítése

A statikus tagfüggvények

- Tipikusan statikus tagváltozókon dolgoznak.
- Olyan, mint egy globális függvény (nem kapja meg a this-t), csak éppen az osztályhoz tartozik.
- Statikus tagfüggvényen belül nincs is this pointer, ebből következik:
  > Statikus tagfüggvényből nem statikus tagváltozó nem érhető el. Melyik objektumét is változtatná? **Pl. írja ki az EUR balance-ot.**
  > Ugyanígy nem statikus tagfüggvény sem hívható (melyik objektumra hívná!). **Pl. hívja meg a balance kiíró függvényt.**
  > Nem statikus tagfüggvényből statikus tagváltozó elérhető: a közös értéket jelenti.
  > Ugyanígy statikus tagfüggvény is hívható.

Statikus változó inicializálása

- Mindig kell, az előző példa is csak így teljes
- Itt történik meg a helyfoglalás a változó számára
  > Ne a headerbe tegyük…

Konstansok

- Volt: konstans paraméterek
- Konstans tagváltozó
  > Az osztályomnak van egy tagváltozója, amit nem szeretnék megváltoztatni
  > Valamikor kezdőértéket kell kapnia! Az objektum létrehozásakor
- Példa: accountId
- Statikus konstans példa (pl. rögzített árfolyam)

Konstansok

- Mit jelent, ha egy objektum konstans?
  > Hogy nem változhat meg, vagyis az állapotát nem változtathatjuk meg.
  > Vagyis a tagváltozóit nem írjuk át, még akkor sem, ha public.
- De ez nem elég: hívhatok rajta tagfüggvényt, ami ezt kijátszhatja.
- Példa

Konstans tagfüggvény

- Jelezni kell, hogy ez a függvény nem fogja megváltoztatni az állapotot.

- Ez a **konstans tagfüggvény**. Olyan, mintha a 0. paraméter, a **this, konstans** lenne.

```
int getBalanceHUF() const {
    return balanceEUR * rate;
}
```

A láthatóság enyhítése

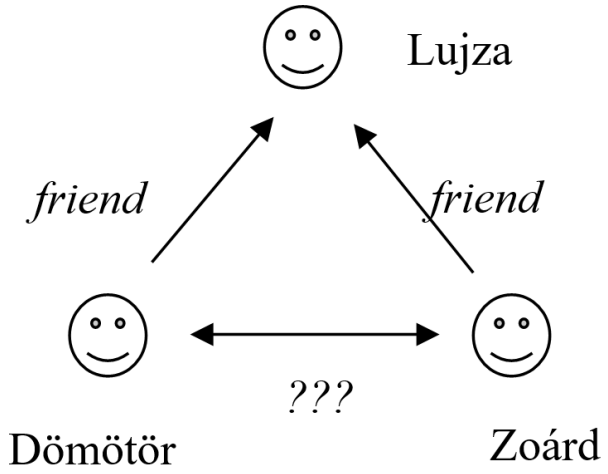
- Írhatunk olyan globális függvényt, amit egy adott osztály felhatalmaz arra, hogy a védett (private, protected) tagjait is elérje.
  > Így mindazokkal a lehetőségekkel bír, mint a tagfüggvény, de mégsem az.
- *Friend* kulcsszó
- Csak akkor használd, ha elkerülhetetlen!
  > A legtöbbször getter, setter függvények a jó megoldás
- Példa

Friend osztályok

- Egész osztályt hatalmazunk fel a hozzáférésre
- Példa

Friend osztályok

- Egész osztályt hatalmazunk fel a hozzáférésre
- Példa
- Vajon tranzitív?



Névterek

- Miért állományszintű a hozzáférés szabályozása?
- Hogyan lehet több osztály ugyanolyan láthatóságú?
- Pl.: sort függvény létezik a standard kódkönyvtárban. A string osztály is létezik. Attól még én is írhatok. Melyiket használjuk?

Névterek

- Megoldás: Névterek.
- Függvények, osztályok, típusok (typedef), konstansok, globális változók definíciójának hierarchiába szervezését teszi lehetővé.

- Példa

Using

- A using hatása az adott *deklarációs régióra* terjed ki
  > Analógia: ha deklarálnánk a *using namespace* helyén egy változót, akkor az honnan lenne látható.
  - A példa szerint használva a fordítási egységre terjed ki a hatása az *adott ponttól*.
  - Akár függvényen belül is használható a using: csak azon belül terjed a hatása (az általa hívott függvényekre már nem).

- De: headerbe ne tegyünk using-ot, mivel nem tudjuk, hova lesz beépítve, és ott milyen hatása lesz.

Using

Az using használatát a legtöbb kódolási konvenció nem javasolja, mert a névütközéseket újra előhozhatja.
Leginkább kompatibilitási okból került be, amikor a standard kódkönyvtárból minden átkerült az std névtér alá (így a régi programokat csak egyszer kellett módosítani)

Ja, meg a lusta oktatók használják az órai kódokban, hogy keveset kelljen gépelni :D

Összefoglalás

- Tagváltozók inicializálása
- Statikus tagváltozók és tagfüggvények
- Konstans tagváltozók és tagfüggvények
- A láthatóság enyhítése (friend)
- Névterek