IV. Öröklődés

1. Az öröklődés fogalma

Osztályok létrehozásának nem csak az az egyetlen módja, hogy egy új osztályt definiálunk. Sok esetben sokkal kényelmesebb és praktikusabb, ha egy már meglévő osztály felhasználásával hozunk létre egy új osztályt.

Def.: Azt a folyamatot, amikor egy már meglévő osztály felhasználásával, annak kiterjesztésével hozunk létre egy új osztályt, **öröklődésnek** vagy *kiterjesztésnek* nevezzük.

Elnevezések:

- ***ősosztály (superclass)***: a már meglévő osztály, ebből hozzuk létre az új osztályt
- ***utódosztály (subclass)***: ez a leszármazott osztály, amely az ősosztály tulajdonságait örökli

Az öröklődés során az utódosztály megörökli az ősosztály tulajdonságait és metódusait, de ezen felül lehetőségünk van újabb tulajdonságok (változók) és metódusok definiálására illetve az örökölt metódusok felüldefiniálására.

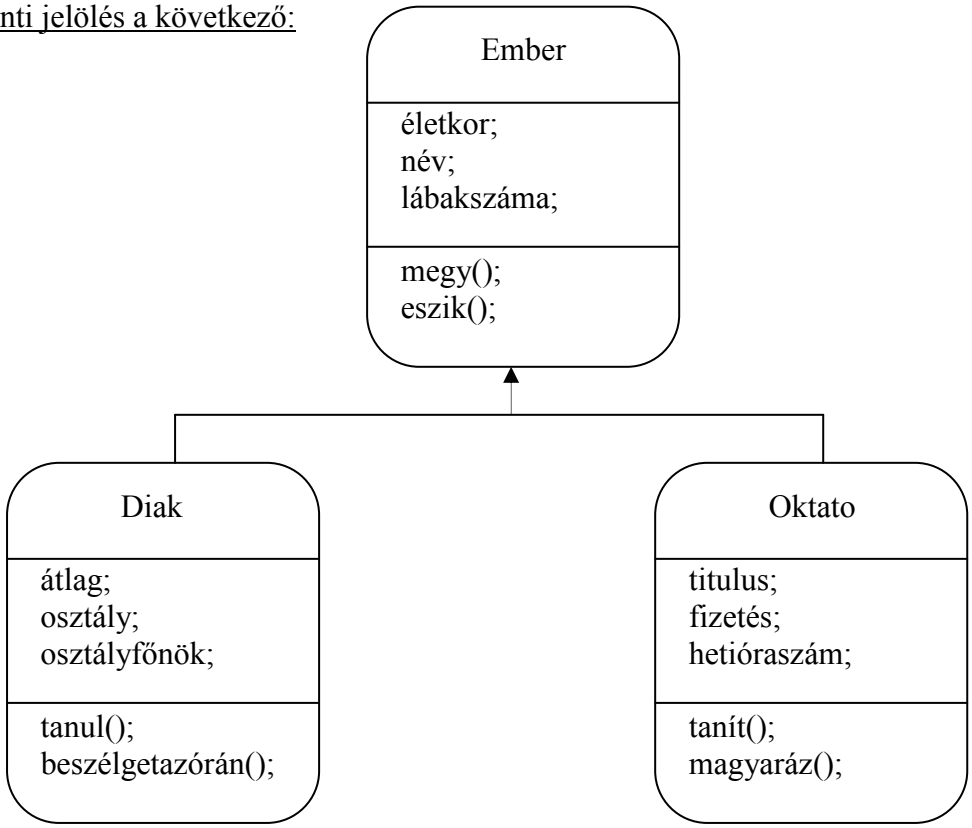
Az öröklődés szabályai:

- egy osztályból több osztály is származtatható
- egy osztályhierarchia mélysége tetszőleges lehet (de túl sok szint nehezen áttekinthető)
- az öröklődés tranzitív: ha B örökli A-t és C örökli B-t, akkor C örökli A-t is
- Javaban minden osztálynak csak egy közös őse lehet (csak egyszeres öröklődés lehetséges)

2. UML jelölési rendszer – példa alapján

Legyen egy meglévő osztályunk a korábban megismert Ember osztály. Feladatunk az, hogy modellezzük a diákokat. Tudjuk, hogy a diákok is emberek, ezért felesleges lenne újra definiálni azokat a tulajdonságokat és metódusokat, amelyek már megtalálhatók az Ember osztályban. Ebben az esetben célszerű az öröklődés használata, melynek során a Diák osztály örökli az Ember osztály minden tulajdonságát és ezen kívül kiegészíthető új tulajdonságokkal is, mint pl.: átlag, tanul() …stb.

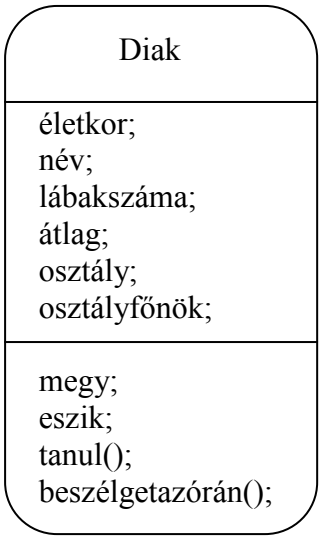
Az UML szerinti jelölés a következő:



(Az UML jelölési rendszerben az ős felé mutat egy üres fejű nyíl a leszármazottól.)

Látható, hogy egy osztálynak több leszármazottja is lehet. (Bármilyen furcsa, az Oktató is Ember)

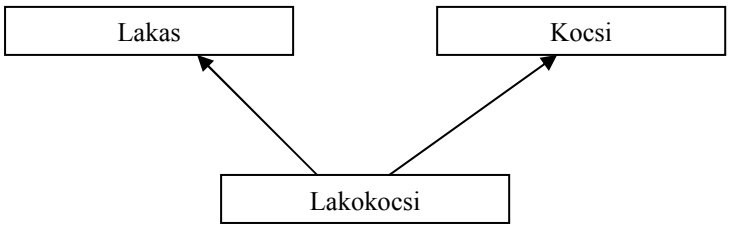
Az öröklés után az újonnan létrehozott osztály (pl.: a Diak osztály) rendelkezni fog az Ember osztály attribútumaival is. Ezért a Diak osztály végső specifikációja a következőképpen néz ki:



3. Többszörös öröklődés

A Java nyelvben kizárólag egyszeres öröklődés van, azaz egy osztálynak csak egy ősosztálya lehet. (A C++-ban lehet többszörös öröklődést is használni). Ennek ellenére a Java biztosít olyan eszközöket, amivel meg lehet oldani a többszörös öröklődést, de erről majd csak a későbbiekben lesz szó (interfészek).

Nézzünk egy példát a többszörös öröklődésre:
Tegyük fel, hogy egy Lakókocsi osztályt szeretnénk készíteni. A lakókocsi rendelkezik a Lakás és a Kocsi tulajdonságaival egyaránt, ezért érdemes lenne e két osztályt ősnek választani:



A Lakokocsi örökli mind a Lakas, mint a Kocsi tulajdonságait és metódusait.

4. Az öröklődés használata Java-ban

A fordítóval az extends kulcsszó felhasználásával közölhetjük származtatási kérelmünket:

```
class UtodOsztaly extends OsOsztaly
{
  …
}
```

Ezután az *UtodOsztaly* osztályban az *OsOsztaly* osztály összes adateleme és metódusa rendelkezésünkre áll. Természetesen annak nem sok értelme lenne, ha az utód osztály az ős osztály egyszerű másolata lenne, ezért a következő lehetőségeink vannak:

- a származtatott osztály új adatelemekkel és metódusokkal való bővítése
  és/vagy
- az örökölt metódusok felüldefiniálása

A következőkben az öröklődéssel kapcsolatos fontosabb tudnivalókról és eszközökről lesz szó.

a) Az örökölt metódusok felüldefiniálása

Ahhoz, hogy egy származtatott osztály örökölt metódusát felüldefiniáljuk, csupán újra kell definiálnunk azt. Alapvetően ugyanarról van szó, mint amikor egy új osztályt hozunk létre, csak arra kell ügyelni, hogy a metódus szignatúrájának (visszatérési érték típusa, neve, paraméterei) teljes mértékben meg kell egyezniük az ősosztály metódusának szignatúrájával.

Nagyon fontos, hogy az öröklődés csak az egyik irányba működik, tehát ha az utódosztályban egy ősosztálytól örökölt metódust megváltoztatunk, attól az ősosztály metódusa nem változik meg, sőt az ősosztály eredeti metódusa továbbra is hozzáférhető (csak egy kicsit másként kell hívni, de ezt a későbbiekben tárgyalni fogjuk).

b) A ***this*** és a ***super*** objektumreferenciák

- A ***this*** objektumreferenciáról már volt szó a III. részben. Ez a referencia mindig az éppen megszólított objektum osztályára utal.
  Pl.: *this.adat* vagy *this.metódus()*
- A ***super*** referencia mindig a közvetlen ősosztályra utal. Ezzel a referenciával mindig csak az osztály feletti első (hozzá legközelebb eső) ősosztályra hivatkozhatunk.
  Pl.: *super.adat* vagy *super.metódus()*

c) Konstruktorok használata az öröklődésben

Az előző fejezetben láttuk, hogy a konstruktorok célja a létrejövő objektumok inicializálása. A konstruktorokat az utódok nem öröklik. Ez arra kényszeríti a programozót, hogy meggondolja, elegendő-e az ős konstruktorát végrehajtani, vagy szükség van-e kiegészítő tevékenységekre (pl.: az új változók inicializálására). Ha a szülő konstruktorát akarjuk meghívni, akkor a super() kulcsszót használjuk megfelelő paraméterezéssel. Pl.: super(nev,atlag)

d) Hozzáférési módosítók

Eddig kissé felületesen azt mondtuk, hogy az utód osztály az ősosztály adatait és metódusait örökli. Ezt a kijelentésünket korlátoznunk kell, mert az utód csak azokat a dolgokat örökölheti, amelyeket az ős engedélyez, azaz nem tilt meg.

Ilyen korlátozás lehet például az, hogy ha az ős osztályban egy változót *private* láthatósági módosítóval látunk el, mert ilyenkor az utód közvetlenül nem érheti el ezt a változót (esetleg az örökölt metódusokon keresztül elérhető a változó).

Egy hasonló korlátozás az is, ha az ős osztályban például final módosítóval deklarálunk egy metódust. Ebben az esetben az utódnak nincs lehetősége a metódus felüldefiniálására.

e) Ős osztály elhelyezkedése

Ahhoz, hogy a fordító el tudja végezni az örökítés folyamatát, ahhoz az ős osztálynak elérhetőnek kell lennie (tipikusan ugyanabban a könyvtárban található az ős bájtkódja).

f) Az ősök őse – az object alaposztály

Az object alaposztály minden osztálynak őse.

5. A többszörös öröklődés megvalósítása Java-ban – az interfész fogalma

A Java nyelvben a többszörös öröklődést az interfészek használatával tudjuk megoldani. Az interfész konstans értékek és absztrakt metódusok deklarációjának az összessége. Ez azt jelenti, hogy az interfész metódusai csak deklarálva vannak, nincs megadva azok implementációja (nincsenek utasításaik…stb). Láthatjuk, hogy önmagában az interfészek csak egy absztrakt vázlatot adnak, de közvetlenül nem használhatók. Az interfészek az implementáció során válna használhatóvá. Az implementációt egy osztály fogja elvégezni, amely az absztrakt metódusokat definiálni fogja, azaz tartalommal fogja feltölteni.
(Az öröklődés az osztályokhoz hasonlóan az interfészek között is működik az extends kulcsszó felhasználásával). Ez a megoldás azért lesz alkalmas a többszörös öröklődés megvalósítására, mert egy osztály több interfészt is implementálhat.

a) Interfészek készítése

[módosítók] **interface** InterfészNév [extends …]
{
//törzs
}

b) Interfészek implementálása

[módosítók] **class** OsztályNév **implements** InterfészNév1 [,InterfészNév2…]
{
//törzs : kötelező az interfész absztrakt metódusainak megvalósítása (implementálása)
}