**Öröklődés, polimorfizmus**

*3. OOP alapelv*

Egy osztály létrehozható úgyis, hogy egy másik osztály leszármazottja. Ilyenkor az osztály:

– örökli a szülője adattagjait, metódusait (tehát már nem kell ezeket definiálni),

– lehet új adattagokat és metódusokat definiálni

– lehet az örökölt metódusokat módosítani (felüldefiniálni, overriding), azaz a viselkedés a leszármazottban is létezik, csak másképpen.

Az öröklődés jelentősége:

– valós életbeli osztályozás (leszármazás) modellezése.

– kód újrafelhasználás. Gyakran előfordul, hogy egy program különböző osztályai azonos adattagokkal és metódusokkal rendelkeznek.

Öröklődés használatának előnyei:

– A leszármazott osztályok definíciója egyszerűbb (csak a különbözőségeket tartalmazza, az öröklött adattagok és metódusok definícióit nem)

– A közös részek módosítása egyszerűbb (csak egy helyen, a szülőben szerepelnek)

– Egy leszármazott mindig helyettesítheti az őst (minden adattaggal és metódussal rendelkezik, amivel az ős)

Öröklődés használatának hátrányai:

– Alapos tervezést igényel: egy osztály tagjait úgy kell megírni, hogy ha később leszármaztatjuk, ne legyen probléma (ne kelljen újratervezni)

– Ha nem akarom, hogy az osztályból lehessen leszármaztatni, akkor ezt explicite meg kell adnom a definícióban (final).

Öröklődés jellemzői Java-ban:

– Egyszeres öröklés: egy leszármazott osztálynak csak egy őse lehet.

– Egy osztályból több osztályt is le lehet származtatni.

– Egy leszármazott osztály lehet egy másik osztály őse.

– Egy osztály nem lehet saját maga őse.

– Final módosítójú osztálynak nem lehet leszármazottja.

Terminológia:

– ősosztály (super class, base class)

– leszármazott osztály (subclass, derived class). Ha nem adunk meg ősosztályt (nincs extends kulcsszó), akkor automatikusan a java.lang.Object osztály az ős. Az Object minden osztály őse, tagjait minden osztály örökli.

– Adattagja nincs – Egyetlen konstruktora paraméter nélküli.A leszármazott osztály örökli az ősosztály tagjait, de azokat a hozzáférési módosítóktól függően használhatja.

– private: a leszármazott nem látja

– default (félnyilvános): a leszármazott csak akkor fér hozzá, ha ugyanabban a csomagban van mint az ős (ezeket az ősosztály public metódusai segítségével kezelheti)

– protected, public: a leszármazott közvetlenül hozzáfér.

**A konstruktor NEM öröklődik**.

Lehet azonban olyan konstruktort írni, mely az örökölt tagokat is inicializálja (még akkor is, ha a leszármazott ezekhez közvetlenül nem fér hozzá).

Konstruktor első utasítása lehet :

– this(…) : adott osztály másik konstruktorának hívása

– super(…) : ősosztály konstruktorának hívása

– Ha egyik sem, akkor első utasításként automatikusan beszúrásra kerül egy super(); paraméter nélküli hívás. A konstruktor első utasítása: ha nem a programozó által megadott this(…) vagy super(…), akkor automatikusan super();

Az automatikus első utasítás következtében egy hívási lánc alakul ki egészen a hierarchia csúcsáig (Object osztály) amikor egy objektum létrejön. Ha egy osztály konstruktorába nem írok this(…)-t, vagy super(…)-t, vagyis az automatikus super() (paraméter nélkül) kerül beszúrásra, de az ősosztálynak nincs paraméter nélküli konstruktora, akkor a fordító hibát jelez.

Ha a leszármazott osztályban egyáltalán nem írok konstruktort, azaz default konstruktor kerül beszúrásra (paraméter nélkül), melynek első utasítása a super() paraméter nélküli hívás, de az ősosztálynak nincs paraméter nélküli konstruktora, a fordító hibát jelez.

Következmény: célszerű minden leszármaztatható osztályban paraméter nélküli konstruktort (is) definiálni

Az Object osztály konstruktora paraméter nélküli. Az inicializáló blokkok NEM öröklődnek.

Az ősosztály inicializáló blokkja lefut: – osztályszintű inicializáló blokk (static): az osztályra történő első hivatkozáskor, a leszármazás sorrendjében – példányszintű inicializáló blokk: példányosításkor, mindig az adott osztály konstruktorának lefutása előtt (azaz a konstruktor hívási lánc miatt az ősosztályok inicializáló blokkjai az adott osztály konstruktora előtt lefutnak). Minden adattagot örököl a leszármazott osztály.

A leszármazott osztály új adattagokat is definiálhat.

Ha a leszármazott osztályban ugyanolyan nevű adattagot definiálunk, mint egy örökölt tag, akkor elfedjük az örökölt tagot (hide). Az örökölt, de elrejtett tagra a közvetlen leszármazottban még tudunk hivatkozni a super.tagnév-vel, de a leszármazott osztály további leszármazottaiban már nem. A leszármazott osztály örökli az ősosztály minden metódusát.

A leszármazott osztály új metódusokat is definiálhat. Akkor számít újnak egy metódus, ha a neve nem egyezik meg egyik örökölt metódus nevével sem.

A leszármazott osztályban definiálhatunk egy örökölt metódussal azonos nevű, de eltérő paraméter szignatúrájú metódust (overloading). Lásd függvénytúlterhelés.

A leszármazott osztály felüldefiniálhat egy örökölt példányszintű metódust (overriding). Ekkor a két metódus neve és paraméter szignatúrája megegyezik.

Ha a felüldefiniált metódust az @Override annotációval jelöljük, a fordító ellenőrzi a felüldefiniálás szintaktikai helyességét.

A leszármazott osztály elrejthet (hide) egy osztályszintű (static) metódust, ha az örökölt osztályszintű metódussal azonos nevű, paraméter szignatúrájú és visszatérési típusú metódust definiálunk (statikus polimorfizmus).

Az elrejtett osztályszintű metódusra a leszármazottban tudunk hivatkozni (ha elérhető) az osztálynévvel minősítve. Az öröklődés egyik előnye, hogy a leszármazott mindent tud, amit az ős. Ezért lehetséges, hogy egy ős típusú változó egy leszármazott típusú objektumra mutasson. De minden változóra csak a saját típusához definiált (vagy öröklött) metódusok hívhatók! Jelentősége: tudok olyan műveletet írni, amely nem csak konkrétan az ős típussal működik, hanem annak bármelyik leszármazottjával is. Egy referencia változónak két típusa van:

1. A deklaráláskor megadott típus (statikus típus): ez határozza meg milyen műveleteket végezhetünk a referenciával hivatkozott objektummal (azaz milyen metódusokat hívhatunk meg).

2. A hivatkozott objektum tényleges típusa (dinamikus típus): amelyik osztályból példányosítom. Ez csak olyan típus lehet, amely rendelkezik ugyanazokkal a műveletekkel, adatokkal, mint a statikus típus. Ez csak a statikus típus leszármazottai esetén garantált.

**Szabályok:**

Egy referencia felvehet értékül a statikus típusával megegyező vagy annak leszármazottja típusú referenciát(–Nem vehet fel egyéb típusút). Egy referencia átkonvertálható (explicit típuskonverzióval) a dinamikus típusára, vagy annak ősére. Egy referencia dinamikus típusának vizsgálata: instanceof operátorral referenciaváltozó instanceof Típus értéke igaz, ha a vizsgált referenciaváltozó dinamikus típusa Típus (vagy annak leszármazottja). A polimorfizmus egyik implementációja: Bizonyos viselkedések működése függ a környezettől, ahol alkalmazzuk.

Az ősosztály egy viselkedése, másképpen működik egy leszármazottban. Pl.: A Kör osztály a Síkidom osztály leszármazottja. A Kör osztály örökli a Síkidom osztály „elmozdít” metódusát, de ennek implementációja eltér az ősosztálybeli definíciótól.

Implementáció: a leszármazottban is definiálunk egy ugyanolyan szignatúrájú metódust (azonos név és azonos paraméterszignatúra) mint az ősosztályban lévő, csak más függvénytörzzsel. A metódus felüldefiniálás (overriding) csak példányszintű metódusokra működik.

Osztályszintűek esetén elrejtés (hide).

A leszármazottban lévő metódusban meghívhatom az ősosztály felüldefiniált metódusát a super.metódusnév(…); hívással, ha kiegészíteni akarom a korábbi működést.

Felüldefiniálás szabályai:

– visszatérési érték egyezőség (kompatibilitás)

– throws rész kompatibilitás (lásd a kivételkezelésnél) – láthatóság nem lehet szűkebb. @Override annotáció: a felüldefiniáló metódus elé írjuk. Hatására a fordító leellenőrzi, hogy létezik-e az ősosztályban a felüldefiniált metódus és ha nem, akkor hibát jelez.

Szerepe: véd a véletlen hibák ellen.

Például: – Az ősosztálybeli toString()-et akarom felüldefiniálni és véletlenül tostring()-nek nevezem. Ha nincs az annotáció, akkor a fordító új metódusnak tekinti; ha ott van az annotáció és az ősben nincs tostring() metódus, akkor hibát jelez. – Az ősosztálybeli toString() metódust akarom felüldefiniálni, de nem azonos paraméterszignatúrát alkalmazok. Ha nincs az annotáció, a fv túlterhelés esete áll fenn: a hatáskörön belül 2 azonos nevű fv lesz elérhető, különböző paraméterezéssel. Ha kiírom az annotációt, a fordító hibát jelez. Tehát egy objektum minden példányszintű metódusának lehet több változata (az ősosztályban definiált és a leszármazottban felüldefiniált).

Amikor egy objektumnak meghívjuk a példányszintű metódusát, akkor a fordító az alapján dönti el, hogy melyiket kell meghívnia, hogy mi a referencia változó dinamikus típusa. Kötés (binding): egy metódus híváshoz a megfelelő definíció megtalálása. – Korai kötés: a kötést a fordító program el tudja végezni fordítási időben. (Pl. C-ben minden függvényre, Javaban statikus metódusra) – Késői kötés: a kötést a fordító nem tudja elvégezni fordítási időben. (Pl. Java-ban példányszintű metódusokra). Mi a különbség a metódus elrejtése (hiding) és felüldefiniálása (overriding) között? A metódus elrejtés korai kötéssel valósul meg (fordítási időben); míg a metódus felüldefiniálás késői kötéssel valósul meg (futási időben).

Ezért Java-ban nem lehet példányszintű metódust elrejteni, csak felüldefiniálni.

Más OOP nyelvekben van arra lehetőség, hogy a programozó eldöntse h. egy példányszintű metódust elrejt vagy felüldefiniál (kulcsszóval kell jelezni a leszármazottban az azonos nevű metódus definíciójában) (pl. C#).Java-ban a programozó nem definiálhat felül operátort.

Pl. C++-ban, C#-ban ez a polimorfizmus egyik megvalósulási formája (a függvénytúlterhelés és a metódus felüldefiniálás mellett).

Lényege, hogy egy operátor jelentését ki lehet terjeszteni új típusokra; de csak olyan helyen szabad használni, ahol egyértelmű a jelentése (egyébként nehezen értelmezhető kódot kapunk). Mindenkinek egyértelmű lenne például, ha az Alkalmazott osztályra kiterjesztenénk a ++ operátort?

Java példa operátor felüldefiniálásra: a String osztályban felül van definiálva a + operátor, ezáltal használható String típusú adatok összefűzésére.