Objektumorientált paradigma

V. rész

A VMT és a DMT

A futás alatti kötés könnyű megértése végett nélkülözhetetlen, hogy ezen a ponton beletekintsünk a probléma technikai megvalósításába:

Mi a különbség a virtuális és a dinamikus metódusok között, mi a megvalósítási módszerük, hogyan működnek? Ezekre a kérdésekre igyekszünk válaszolni a továbbiakban.

Láttuk, mi a különbség a statikus és a virtuális, illetve dinamikus metódusok között. Most már azt is megérthetjük, mi a különbség az *elfedés* és a *felüldefiniálás* között. Az osztálymetódusok mindig statikusak, így őket nem lehet felüldefiniálni, csak elfedni.

A virtuális vagy dinamikus metódusok használatával valósul meg tulajdonképpen a futás alatti kötés vagyis a teljes polimorfizmus. A statikus metódusok továbbra is lehetőséget biztosítanak az elfedésre, sőt statikus metódusok használatával meggátolhatjuk azt is, hogy adott esetben az objektum saját metódusa fusson, vagyis az objektum elveszítheti önmagát, de a statikus metódus egyfajta védelmet is jelenthet. Például azokban a nyelvekben, ahol nincs *protected* adatrejtés, így védekezhetünk bizonyos metódusok felüldefiniálása ellen.

A futás alatti kötést azonban meg is kell valósítani. Vagyis biztosítani kell a fordítóprogramnak egy olyan mechanizmust, amely megengedi a futás közbeni cím-számítást, és ugyanakkor helyes címeket is szolgáltat. Erre a megvalósításra alapvetően két különböző megoldás született: a *virtuális metódusok* és a *dinamikus metódusok*. Nézzük meg külön-külön mit is jelentenek ezek.

Minden egyes virtuális metódussal rendelkező osztályhoz tartozik pontosan egy olyan táblázat, amely a virtuális metódusok címét tartalmazza. Ezt a táblázatot *VMT*-nek (*Virtual Method Table*, *Virtuális Metódus Tábla*) nevezzük és a fordító építi fel, a programozó direkt módon nem fér hozzá. Fontos kihangsúlyozni, hogy a VMT az osztályhoz tartozik és nem a példányokhoz, az objektumokhoz! Az objektumok viszont hozzá vannak kötve az osztály VMT-jéhez. Ezt a kötést valósítja meg a konstruktor, és ezt a kötést bontja le tulajdonképpen a destruktor úgy, hogy minden objektum tartalmaz egy *VMT-mezőt*, és ez a mező mutat az osztály VMT-jére. A VMT-mezőhöz sem lehet hozzáférni direkt módon.

A virtuális metódusok táblázata lényegében a megfelelő osztályhoz tartozó virtuális metódusok címeit tartalmazza, deklarálási sorrendben. A statikus metódusok címeit, ha vannak egyáltalán ilyenek, nem kell beleírni a táblába, hisz ezek már a fordítás pillanatában ismertek és bekerülnek a kódba. Fontos megjegyezni azt is, hogy egy öröklődési ágon az egyes osztályokból származtatott osztályokban az ugyanolyan nevű virtuális metódusok címei a VMT ugyanazon relatív címén helyezkednek el, és az újonnan deklarált metódusok a deklarálás sorrendjében kerülnek bele a táblába. Tulajdonképpen ez a futás alatti kötés működési elve. Így minden pillanatban biztosított a fordítóprogram számára a címszámítás lehetősége és mechanizmusa. A VMT felépítése tehát a következő:

Az osztály adatainak mérete (az ős adatokkal együtt) + a VMT mező mérete.
Az osztály adatainak mérete negatív (-) előjellel. Ellenőrző mező.
Virtuális metódusok címei, a deklarálás sorrendjében úgy, hogy öröklődés esetén a hasonló nevű virtuális metódusok címei a VMT ugyanazon relatív címén helyezkednek el.
…

1. ábra A VMT szerkezete

A virtuális metódusok címei mellett a VMT tartalmazza az osztály adatainak összméretét és ugyanezt a méretet negatív előjellel is. Ennek ellenőrző szerepe van. Az objektum VMT-mezőjét a konstruktor köti az osztály VMT-jéhez. Ha tehát nem hívtuk meg a konstruktort, azt jelenti, hogy a VMT-mező egy véletlen memóriahelyre mutat, és ez a hely nagyon-nagyon ritkán rendelkezik azzal a tulajdonsággal, hogy a két egymás után következő zóna értékének összege zéró legyen, ugyanakkor egyiknek az értéke sem zéró. A program így megnézi, valóban egy igazi VMT-re mutat-e a VMT-mező, vagyis a konstruktor meg volt-e hívva és a címszámítás jól működik-e.

A VMT-nek van egy óriási előnye: a keresés benne nagyon gyors, hisz minden osztálynak van egy VMT-je, amely a saját és az örökölt (és felüldefiniált) virtuális metódusok címét tartalmazza, így, ha a programnak szüksége van egy metódus címére, akkor minden további nélkül megkaphatja az osztályhoz tartozó VMT bejárásával. De ez az előny egy nagy hátrány is ugyanakkor: mivel az öröklődési hierarchia során minden osztálynak megvan a saját VMT-je, amely az összes virtuális metódust tartalmazza, ezek a táblázatok igencsak nagy helyet foglalnak le a memóriában.

Ezt az előnyt és hátrányt próbálták meg másképp ötvözni (mert teljesen kiküszöbölni nem lehet) a *Dinamikus Metódus Tábla* (*DMT – Dynamic Method Table*), és ezáltal a *dinamikus* metódusok bevezetésével. A különbség csupán annyi, hogy az öröklődési hierarchia során az osztályok DMT-i nem tartalmazzák az összes örökölt dinamikus metódus címét, csak a saját, felüldefiniált metódusaiknak a címét, viszont, hogy a keresés meg tudjon valósulni, a DMT-k kapcsolódnak egymáshoz. Így el lehetett érni azt, hogy ne foglaljanak nagy helyet a táblázatok, viszont a keresési sebesség csökkent, mert nem elég bejárni csak egy osztályhoz tartozó táblázatot, hanem adott esetben az egész hierarchiát végig kell keresni.

Az, hogy melyik metódus legyen virtuális és melyik dinamikus, teljesen a programozóra van bízva, az általános, követendő elv azonban az, hogy a virtuális metódusok valósítják meg a teljes polimorfizmust (ott, ahol gyakran definiálunk át metódusokat, de kevés virtuális metódusunk van), a dinamikus jelzővel pedig olyan metódusokat látunk el, amelyek csak ritkán definiálódnak felül, ezek nagy számban vannak jelen valamelyik ősosztályban, de sejtjük róluk, hogy a hierarchia során ezeket valamelyik pillanatban felül kell definiálni. Így elérhetjük azt, hogy alkalmazásaink, programjaink megírása során optimális kód jön létre.

Természetesen, a metódusok jellegét nem lehet megváltoztatni a hierarchia során. Ha egy metódus virtuális, akkor az mindvégig virtuális marad és címe a VMT-be kerül. Ha egy metódus dinamikus, az mindvégig dinamikus marad és címe a DMT-be kerül. Ilyen értelemben a hierarchia adott pontján, ha felül akarunk definiálni egy metódust, nem kell explicit tudással rendelkezzük arról, hogy az illető metódus dinamikus vagy virtuális, ez már az első deklarálás pillanatában eldőlt, így a felüldefiniáláskor nem kötelező használni a **virtual** vagy a **dynamic** direktívát, elég, ha csak azt mondjuk meg, hogy az illető metódus felüldefiniált (**override**), ahhoz, hogy meg lehessen különböztetni a statikus metódusoktól.

Konstruktorok és destruktorok

Mint már említettük, a konstruktorok speciális metódusok. Minden olyan osztály, amely virtuális vagy dinamikus metódusokkal rendelkezik, kell hogy rendelkezzen konstruktorral is, hisz ez teremti meg a kapcsolatot a VMT-vel és/vagy a DMT-vel, vagyis értéket ad a VMT (DMT) mezőnek. Mivel a konstruktor az első metódus, amelynek, célszerű akár példányosításra, akár az objektumok kezdőállapotának beállítására (inicializálás) használni. Foglaljuk tehát össze a konstruktorokkal kapcsolatos tudnivalókat:

- A konstruktor teremti meg a kapcsolatot a VMT-vel és/vagy DMT-vel. Ebből kifolyólag a konstruktort akkor is meg kell hívni, ha annak a törzse éppen üres (absztrakt konstruktor).
- Az objektumot még az első virtuális, dinamikus metódus hívása előtt a konstruktorral inicializálni kell. Először mindig a kontsruktort hívjuk!
- A konstruktor használható az objektumok példányosítására és az objektumok inicializálására (a kezdőállapot megadására). Emiatt a konstruktorok gyakran osztály-metódusok.
- A konstruktort át lehet örökölni, ebben az esetben a VMT hozzárendelés csak a legfelső szinten történik, amikor a konstruktort példányra hívjuk. Ha a konstruktort konstruktorból hívjuk (**inherited**), akkor az normál metódusként fog működni. Gyakran szoktunk így hívni konstruktorokat, mert ezek elvégzik az örökölt adatok inicializálását.
- Az örökölt konstruktor is úgy viselkedik mint bármely más konstruktor, az aktuális osztályban nem kell feltétlenül felüldefiniálni.
- Elvileg a konstruktorok statikusak, vagyis nem lehetnek a maguk során virtuálisak, hisz ez azt jelentené, hogy a hívása előtt egy konstruktort kellene meghívni, amely felépíti a VMT-t… De a későbbiekben látni fogjuk, hogy bizonyos speciális esetekben a konstruktorok lehetnek virtuálisak is.
- Egy osztálynak több konstruktora is lehet, ha ezeket egymás után hívjuk, az első teremti meg a VMT kapcsolatot, a többi normális metódusként működik. Legalább egy konstruktorra mindig szükség van.
- Némely programozási nyelvben a konstruktort a **constructor** fenntartott szó segítségével lehet deklarálni, más nyelvekben a konstruktor egy olyan metódus, amelynek a neve megegyezik az osztály nevével.

A destruktorok a konstruktorok ellentett műveletei, vagyis leépítik a VMT kapcsolatot. Ezért mindig a destruktorhívás legyen az utolsó metódushívás, akkor, mikor már tényleg nincs szükségünk az objektumra. Emellett a destruktorok még más leépítési műveletet (az objektum belső adatzónáinak a felszabadítását stb.) is elvégezhetnek. Egy osztály több destruktort is tartalmazhat, természetesen az első destruktorhívás után már nem hívhatunk meg újabb destruktort, hisz a példány már le van építve. Ezért a destruktort a példányok leépítésére, megszüntetésére is használjuk. Természetesen a destruktorok is öröklődhetnek, sőt ezek virtuálisak vagy dinamikusak is lehetnek. Némely programozási nyelvben a destruktort a **destructor** fenntartott szóval kell deklarálni, más nyelvekben a *~Osztálynév* konstrukcióval deklaráljuk, vagy ezeknek a működése akár automatikusan is történik. Ha megszüntetünk az objektumra minden referenciát, akkor automatikusan meghívódik a destruktor és a példány leépül (ez inkább interpreter jellegű nyelvekre vonatkozik).

Kovács Lehel