1. fejezet - Java könyv

Tartalom

A következő bekezdésekben a Java 2 útikalauz programozóknak 5.0 első kötetének élménybeszámolóját olvashatják: A Java programozási nyelv hasonló a C nyelvhez, azonban újabb és több lehetőséggel áll rendelkezésünkre. Nyelvünk objektumorientált, ami annyit tesz, hogy egy program objektumokra, azon belül osztályokra van felbontva, míg a C és C++ nyelvek eljárásorientáltak. Az osztályokon belül használunk változókat, illetve metódusokat. Ez utóbbi felel azért, hogy milyen műveletet hajtunk végre az adatokon. Míg a C-nél mutatókat használunk, a Java-ban már referenciákat. Változókat épp úgy adhatunk meg mint C vagy C++ nyelveknél, először a változó típusa (int, string, double, stb…), majd „=” operátor és az érték amit meg szeretnénk adni. Amikor Java kódot írunk érdemes megjegyzéseket hozzá tenni a programunkhoz (épp úgy mint bármelyik másik nyelvnél), ezt a „//” egyetlen sor esetében, ha több sorban szeretnénk megjegyzést írni akkor a „/*” nyitó részként, és a „*/” záró részként szolgáló jelöléssel tehetjük meg. Osztályokat a class kulcsszóval vezetünk be, amiken belül tetszőleges sorrendben vehetjük fel annak metódusait, és adatait. Fontos eldönteni, hogy kinek a számára legyen látható az adott elem. Így pl.: egy public osztály mindenki számára látható, míg private csak az osztályon belülre vonatkozik. Osztályon belül a new operátorral hozhatunk létre újabb objektumokat. Ügyelnünk kell a kivételkezelésre (bizonyos programok esetében pl.: nullával való osztás), ezt a try és catch segítségével oldhatjuk meg. A Java sok kétdimenziós grafikai elemmel rendelkezik, ilyen pl az awt, amivel saját magunk készíthetünk két dimenziós alakzatokat. A legelterjedtebb karakterkészlet az ASCII kódrendszer, ami 8 biten ábrázolja a karaktereinket. Ebből azonban sok nemzet által gyakran használt karakterek hiányoznak (nálunk pl.: ű és ő betűk). Az ASCII-tól kétszer annyi biten ábrázoló Unicode azonban már rendelkezik a számunkra szükséges karakterekkel. Java-ban az azonosítók betűvel kell, hogy kezdődjenek és betűvel vagy számmal kell folytatódniuk. Tetszőleges hosszúak lehetnek, de nem tartalmazhatják a nyelv kulcsszavait (pl.: int, bolean, for, return, stb…). Az egyszerű típusokat és objektumokat literálokkal inicializáljuk, ezek pedig: objektum, logikai érték, egész szám, lebegőpontos szám, karakter, szöveg, és végül osztály. Változódeklarációnál kell hogy megadjunk egyetlen típust és legalább egy változónevet. Tömböket hasonlóan adunk meg mint C-ben, itt viszont a tömb igazi típus lesz, nem pedig mutató. Ha struktúrákat szeretnénk elérni, azokra ponttal hivatkozhatunk. Az utasítások két fajtája: kifejezés-utasítás és deklaráció utasítás. Elágazásokat az if szerkezettel, összetettebbeket a switch-el oldhatunk meg. A Java-ban a következő ciklusok ismertek: elöltesztelő (while), hátultesztelő (do, while), léptető (for, while), és bejáró (for). Egy ciklusból a break parancs segítségével léphetünk ki, ilyenkor a program automatikusan fut tovább a következő kódsorra. Hasonlóan a continue parancs is ezt teszi, azonban ezt már használhatjuk metódusokban és inicializáló blokkokban. Visszatérési értéket C-hez hasonlóan a return parancs fog nekünk adni, viszont a Java-ból kikerült a goto utasítás, a biztonság növelése érdekében. Java-ban az osztályok a legkisebb önálló egységek. Egy osztály egy adott tulajdonságú halmaz elemeit tartalmazza (pl.: emberek, tárgyak). Itt figyelnünk kell, hogy egy adott osztálynak csak egy célja legyen, és ne terheljük túl az osztályokat. Osztályon belüli változódeklarációnál ügyelnünk kell arra, hogy mekkora legyen a változó „hatósugara”, hogy más osztályok ne használhassák egymás változóit. Osztályon belüli metódusokat metódusdefiníciók árják le, amiknek fej és törzs része van. Metódushíváskor nem elég csak a nevet megadni, annak paramétereit is megkell. Figyelnünk kell a metódustúlterhelésre is. Ugyanis egy osztályon belül lehet több azonos nevű metódus is. Ilyenkor a fordítóprogram a paraméterek száma és típusa alapján választja ki a számunkra megfelelő metódust. Objektumokat a new operátorral hozunk létre, a new után pedig megkell adnunk, hogy melyik osztályt példányosítjuk. Példányosításkor memóriát foglalunk le, ahol az objektum változói lesznek tárolva, és visszaadja a kezdőcímét. Programozáskor problémát szokott jelenteni, hogy az objektumok feleslegesen foglalnak memóriát, mert egyszerűen nem hivatkozik rájuk semmi. Java-ban ez nem jelent problémát, mert a rendszer automatikus felszabadítja azokat helyeket, amire nincs hivatkozás. Más nyelveknél ezt a programozónak magának kell megtennie. A metódusokra nagyban hasonlítanak a konstruktorok. Ezek végrehajtása a példányosításkor azonnal megtörténik. Egy konstruktornak meg

kell egyeznie az osztály nevével. Csak példányosításon keresztül meghívhatóak. Néhány esetben nem árt értesülni egy-egy objektum megsemmisüléséről. Ez osztály szinten a classFinalize, metódus szinten simán finalize metódus (ez egy destruktor) fogja nekünk megmondani. Polimorfizmusnak nevezzük amikor egy változó olyan módon van deklarálva, hogy a leszármazottak is hivatkozhatnak rá. Polimorfizmusnál megkülönböztetünk statikus és dinamikus változókat. A statikus változó típusa változatlan, még a dinamikus változhat a változó által hivatkozott tényleges típusra. Az osztályok mellett másik fontos építőköve a nyelvnek az interfész, ami olyan referenciatípus, amely absztrakt metódusok deklarációjának és konstans értékeknek az összege. Valódi használata az implementációján keresztül történik, így egy absztrakt program konkréttá válik. Interfészek között is van öröklődés. Deklarálni az interface kulcsszóval lehet, hasonlóan mint az osztályoknál a class. Deklarációval egy vázat hozunk létre, és ebben implementációkat helyezünk el, amik osztáylokat helyettesítenek. Fontos a fordítási hiba elkerülése végett, hogy publikus implementációkat adjunk az interfész összes metódusához. Az interfész egy újabb referenciatípus. Úgy használjuk mint egy osztályt. Az interfészek öröklődését kiterjesztésnek nevezzük, és az extend paranccsal tudjuk megvalósítani. Ez szintén úgy működik mint az osztályoknál. Konstansokat a következő képpen vezetünk be: módosítók, konstans típusa, azonosító, inicializáló kifejezés és ; zárjuk le, mint a legtöbb sorunkat. Deklarálásukhoz a módosítókat kell használjunk. Az interfészek használata az őket használó osztályok hívják meg. Ez csak akkor tud megvalósulni, ha az összes implementáló metódus szignatúrája és visszatérési értéke megegyezik az interfészével, különben error. Amikor programozunk törekedjünk a letisztult és átlátható formára. Révén, hogy objektumorientált nyelvről van szó, a programrészeket megfelelően tagoljuk. Erre lesznek segítségünkre a csomagok, amik tartalmazzák a fejlesztői környezetet és a kódunkat. Úgy is mondhatjuk, hogy a csomag a hozzáférési kategóriák használatának az eszköze. A csomagoknál az öröklődési szint hierarchikusan van jelen. Itt a gyermeket alcsomagnak hívjuk. Tartalmuk lehetnek típusok vagy további alcsomagok, melyeket ponttal választunk el. Itt egy fa struktúrát kell elképzeljünk, ahol nincs szorosabb kapcsolat az ős felé, mint bármelyik másik csomag iránt. Célja a programkód átláthatóságának növelése, vagyis a programozó munkájának könnyítése. Csomagokat fájlrendszerben (JDK) vagy adatbázisban tárolunk. A csomagok kódja a fordítási egységekben található. Deklarálásuk package kulcsszóval történik, amik a kód legelején kell hogy legyenek, már deklaráció után nem állhatnak. Továbbá lehet importdeklarációt alkalmazni aminek három fajtája a következő: egyszerű típusimport-deklaráció, igény szerinti típusimport-deklaráció, statikus típusimport-deklaráció (ez utóbbi Java 5-től kezdve van jelen). Ezek segítik a típusok használatát. Csomagot létrehozni úgy tudunk, hogy a elkészítjük a hozzá tartozó fordítási egységeket, ezeket deklaráljuk aszerint, hogy melyik csomaghoz tartoznak. Majd megírjuk a típus~ és importdeklarációkat a fordítási egységekben. Ügyeljünk a megfelelő névválasztásra, ugyanis csoportos projekt esetén egy rosszul megválasztott csomagnév sok fejfájást okozhat. Erre egy jó módszer ha az egész elérési útját adjuk meg névként. Sokszor esünk bele abba a hibába, hogy azt hisszük mindent jól kódoltunk le, mégis hibás a program. Azt tudjuk, hogy bizonyos részek lefutnak, na de hol keressük a bugot? Erre használjuk a kivételkezelést (Java exception), ami Java-ban a hasonlóan történik mint C-ben. Ez egy olyan különleges helyzet, hogy valahol hibát dob a program, de mégis lefut, a hibás részt pedig egy kivételobjektum fogja kezelni, pontosabb információt kapva a hibáról. Ezt nevezzük kivétel kiváltásnak (throwing exception), és szintén három fajtája fordul elő: rendellenes dolog történt (pl.: 0-val való osztás), throw utasítás kivált egy kivételt(csomagokban vagy a kódban), aszinkron hiba (párhuzamos futásnál egyik szál megszakad). A kivétel kezelése oly módon történik, hogy a kivételkezelő megkeresi azt a helyet, ahol a kiváltott kivétel kezelése megtörténhet. A kiváltott típusnak meg kell egyeznie a kivétel típusával, vagy őse az osztályhierarchiában. A kivételkezelő dönti el, hogy mekkora az a blokk amit kivált. Ezeket az utasításokat kapcsoszárójelek közé teszi. A kivételt el is kell kapnunk valahogy, erre a célra a try lesz segítségünkre. Ehhez azonban hozzá kell tennünk még közvetlenül utána egy catch vagy finally blokkot, különben errort kapunk. A generikusok a Java 5. újdonságai, feladatuk az osztályok vagy eljárások típusokkal való paraméterezése, ez által sokkal egyszerűbb megoldani a feladatokat. Céljai az egyszerűség, biztonság növelése és a hatékonyság. Típushelyettesítő forma a wildcard ami az alábbi formák egyike lehet: ?, ? extends T, ? super T, itt T egy típust jelöl. Ügyelnünk kell, hogy a következők kivételével minden referencia-típus lehet generikus: névtelen belső osztály, kivétel típusok (oka, hogy futás közben a generikusok nincsenek tárolva, a throwable osztály márpedig futási időben él), felsorolási típusok (mivel statikus jellegűek ugyan az a probléma mint a kivételnél). A generikusok típustörléssel foglalkoznak, és objektumok terén végrehajtott ezért object típus lesz. Figyelnünk kell az instanceof kifejezésre, ugyanis nem mindig működik generikusokra, a típustörlés következményeként. Gyűjteményeknek nevezzük azokat a típuskonstrukciós eszközöket, melyek feladata az egy vagy több típusba tartozó objektumok

példányainak memóriában történő összefoglaló jellegű tárolása, lekérdezése és manipulálása. A legtöbb programozási nyelvben metalálhatók. Általában objektumok és tömbök mutatóinak vagy referenciáinak összekapcsolása. A Java 2-ben jelent meg, inkább a praktikusság jellemző rá, mintsem a teljesség. Alapvető gyűjtemények és leképezések osztályai szorosan összefüggnek, külön-külön nem használjuk őket. Az interfészeikhez több reprezentációs osztály is tartozhat, ezek közötti váltás egyszerű. A gyűjtemény (collection) egyik leszármazottja a halmaz (set), feladata a halmaz adattípus megvalósítása. Megszorításokat tesz a, nem tesz kiterjesztést a gyűjteményhez képest. Örökli a műveleteket, amik kifejezhetők halmazműveletekként. Másik leszármazottja a lista (list), és a lista adattípust valósítja meg. Itt az elemek duplikáltan is szerepelhetnek, és számít a sorrendjük. A lista ad kiterjesztést néhány gyűjtemény interfészhez. Néhány örökölt műveletet is megváltoztat, ez duplikálás és sorrend miatt fontos (pl.: a remove művelete csak az első elemet távolítja el a listából). A sor adatszerkezet lényege, hogy az első objektumot ami bekerül, azt vegyük ki elsőnek. A leképezés adatszerkezet kulcs-érték párokat tárol, a Map interfészen keresztül. Egy kulcshoz egy érték tartozik. Egy szoftverrendszer felépítését az alapján kell megtervezni, hogy az adatokon milyen tevékenységek lesznek elvégezve. A legáltalánosabb a felülről lefelé haladó eljárás, hiszen gyors, könnyen áttekinthető. Azonban nem minden esetben ez a legmegfelelőbb eljárás. Ezt mindig a programozónak kell eldöntenie, hogy mi lesz a legmegfelelőbb a programhoz. Objektumorientált nylevként először mindig objektumokra bontjuk a feladatot. Ez reprezentálja a modellezendő világ egységeit. Az objektumokat osztályokba soroljuk azok tulajdonságai alapján (pl.: élőlény osztálynak van ember, állat és növény objektuma). Ezen belül lesznek az alosztályok (állatoknál pl.: emlős, madár ,stb…). Amikor egy objektumot nem tudunk egyértelműen besorolni egyetlen osztályba sem, olyankor kapcsolatokkal kötjük őket osztályokhoz. Minden osztály meghatározható egy másik osztály leszűkítésével vagy kiterjesztésével. Így jön létre az örökös és gyermek kapcsolat az osztályok között. Minden gyermek osztály örökli az ősosztály tulajdonságait, fordítva nem igaz. A programtervezés legfőbb célja, hogy jól átlátható kódot készítsünk, erre az objektum orientált nyelv az egyik legjobb módszer a fentebb említett példák miatt. Szintén három lépésre oszthatjuk: analízis (a probléma körvonalazása), rendszertervezés (a körünket részekre osztjuk), osztálytervezés (a rendszert további osztályokra bontjuk). Ha mindezzel megvagyunk akkor megkezdődhet az implementálás. A C++ jelölésrendszeréből sok mindent átvett a Java. Egy C++-ban jártas programozónak nem okoz majd nagy nehézséget megtanulni a Java szintaktikai szabályait, mert nagy átfedés van a két nyelv között, mivel a Java szintaxisa a C és C++-ból fejlődött ki. Felépítésükben viszont eltérést látunk, mivel a Java egy teljesen ″ objektumorientált nyelv. Ez annyit takar, hogy minden változó és metódus egy osztálynak a része. Amikor megírunk egy programot, aztán fordítani és futtatni szeretnénk, akkor a C ++ fordítója a saját kódunkat gépi kóddá konvertálja. A gépi kódot már értelmezni tudja az eszközünk és így a programunk már futtatható. Ennek a módszernek az a hátránya, hogy egy másik számítógépen csak azonos platform mellet lesz garantált ugyanaz az eredmény. Ezt másnéven platform- függőségnek is nevezzük. Java-ban ha a típusokról van szó, akkor primitív és nem primitív típusokra oszthatjuk fel őket. A primitív típusok egy konkrét értéket tárolnak. Ezeket a primitív típusokat helyettesíthetjük csomagoló osztályokkal is. Minden primitív adattípus rendelkezik egy hasonló (vagy vele megegyező) névű, de nagy betűs csomagoló osztállyal. Például az int-et helyettesíthetjük az Integer-rel. Osztályokat úgy mint C++-ban, Java-ban is a class teremti meg. A C++-ban különféle beépített vagy akár saját osztályok eléréséhez header fájlokat include-olása szükséges. Ez Java-ban viszont a header fájlok hiányában nem lehetséges. Helyette csomagokat használunk, amelyeket az import-tal érhetünk el, vagy simán megadhatjuk a csomag elérési útvonalát