E. Fejezet

A Java nyelvről Pascal programozóknak

Könyvünkben több helyen is utaltunk arra, hogy a Java nyelv kifejlesztésekor az alkotók a C++ nyelvet vették alapul. Tehát mindazoknak, akik ismerik ezt a programozási nyelvet, könnyebben elsajátítható lesz a Java nyelv is.

Ebben a fejezetben azoknak szeretnénk segíteni, akik a Pascal programozási nyelvet ismerik, azt alkalmazzák és programjaikat ebben írják. Az első ránézésre talán az ő számukra riasztó és könnyen félrevezető lehet több újszerű megoldás, vagy az, hogy a két nyelv eltérő értelmezésben használ azonosan jelölt fogalmakat. Itt csupán a kezdeti lépéseket szeretnénk némileg megkönnyíteni. Természetesen nem szabad megfeledkezni arról, hogy ez a fejezet csak mankó, „valódi" Java programokat csak a nyelv szemléletének megismerése, megértése után lehet készíteni.

A Pascal nyelv rövid története: 1968-ban készült el Nicklaus Wirth professzor terve, melyet az addigi legfejlettebb nyelvek továbbfejlesztésére készített. 1973-ban látott napvilágot a Revised Report. Ebben definiálták a Standard Pascal nyelvet, mely azóta is a különböző implementációk alapja. Több egyetem is foglalkozott a nyelv különböző változatainak kifejlesztésével. A Borland cég pedig kidolgozta a Borland Pascal néven ismert változatot. A továbbiakban a Borland Pascal 7.0 verziót vesszük alapul, az összehasonlításként megírt programrészleteket ebben írtuk.

A Java és a Pascal nyelv különbségeit három részre bontva tárgyaljuk:

- Először áttekintjük mindazokat a lehetőségeket, melyek mind a két programozási nyelvben megtalálhatóak, majd a meglévő különbségekre kitérve megmutatjuk, hogyan lehet könnyen és gyorsan programjainkat átírni.
- A második részben az objektumorientált programozást támogató elemek különbségeit vesszük sorra, és bemutatjuk a Java nyelv által nyújtott többletlehetőségeket is.
- A harmadik egységben pedig röviden kitérünk arra, milyen – a Standard Pascal nyelv lehetőségein messze túlmutató – nyelvi elemek léteznek a párhuzamosság, illetve a kivételek kezelésére.

Reméljük, e három rész együttes használatával kellő segítséget nyújtunk a kezdeti nehézségek leküzdéséhez és a Java nyelv lehetőségeit kihasználó hatékony programok készítéséhez.

E.1. A nyelvek összehasonlítása

E.1.1. A Java nyelv első látásra

Tekintsük a következő egyszerű Pascal programot, amely bekér két számot, majd kiírja a két szám szorzatát:

```
program elso;
var
   a,b : integer;

begin
  writeln('Első példa program Pascal kód alapján');
  write('A=');
```

```
  readln(a);
  write('B=');
  readln(b);
  writeln('A két szám szorzata : ', a*b);
end.
```

Ugyanez a program Javaban a következőképpen néz ki:

```
import java.io.*;
class Elso {
  public static void main(String[] args) {
    int a=0, b=0;
    BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));

    System.out.println("Első példa program JAVA kód alapján");
    System.out.print("A=");
    try {a=Integer.parseInt(stdin.readLine());
    } catch (Exception e) {
      System.out.println("Hibás értéket adtál a-nak!");
    }
    System.out.print("B=");
    try {b=Integer.parseInt(stdin.readLine());
    } catch (Exception e) {
      System.out.println("Hibás értéket adtál b-nek!");
    }
    System.out.println("A számok szorzata : " + a*b);
  }
}
```

A program import utasítással kezdődik, felsorolva a felhasznált csomagok nevét. Ez nem kötelező, az egyes elemekre való hivatkozásnál is megadhatjuk a csomagnevet. Az import a kapcsolatszerkesztőnek, a betöltőnek szól. Importálásnál csomagokból (package) osztályokat (class) és interfészeket (interface) importálunk, azaz a programunkban felhasználhatóvá tesszük azokat.

Minden Java alkalmazásnak, pontosabban az azt vezérlő osztálynak – mivel a Javaban minden utasítás csak osztályok definíciójában szerepelhet – tartalmaznia kell egy main nevű metódust, amely a következő alakú:
`public static void main(String[] args)`
A public a metódus láthatóságáról nyilatkozik, jelentése: bárhonnan meghívható. A static jelentéséről egyelőre annyit, hogy a metódus csak osztályváltozókat használhat, a void pedig a visszatérési érték típusa: nem ad vissza értéket. Egy Java alkalmazás futtatását a virtuális gép ezen main metódus végrehajtásával kezdi. A metódus argumentuma egy sztringtömb, amelynek egyes elemei a parancssorban megadott paramétereket tartalmazzák.

A szövegek tárolására, kezelésére szolgáló String egy előre definiált osztály, nem pedig karaktertömb. A tömbök indexértékei itt nullától kezdődnek. Az args[0] pedig nem a program nevét adja vissza, hanem ténylegesen az első paramétert.

Az egyes metódusokban természetesen használhatunk lokális változókat, az előbbi programban például:
`int a = 5;`
Lokális változókat a metódusok törzsében tetszőleges helyen deklarálhatunk, sőt az egyes programblokkok – `{ }` zárójelekkel határolva – saját, máshonnan nem látható lokális változókkal rendelkezhetnek.

A példaprogramban felhasznált System osztály, amely az automatikusan importálásra kerülő java.lang csomagban található, a Java programok futtatásához szükséges objektumokat tartalmazza. Ilyen a programok szabványos kimenetét (standard output, pl. konzol

periféria) megvalósító out objektum, vagy ennek párja, a szabványos bemenetet (standard input) megvalósító in objektum. A kimenet kiíró metódusai a print, illetve println, melyeknek csak egyetlen String paraméterük van, de minden objektum automatikusan (a toString metódus segítségével) sztringgé konvertálódik. A println metódus a paraméter kiírása után még egy soremelés karaktert is kiír.

A példaprogram a try-catch-throw utasításcsoportot használja hibakezelésre, ami a standard Pascal nyelvből teljesen hiányzik. Kisebb gondossággal megírt programoknál a programozó hajlamos elfeledkezni arról, hogy legtöbbször a visszatérési érték tesztelésével ellenőrizze, sikerült-e a művelet végrehajtása. Ha erre mégis sor kerül, hiba esetén leggyakrabban egyszerűen befejeződik a program. Ezen segíthetnek a fenti utasítások. Most csak felületesen említjük meg a kivételkezelést, hiszen ezzel részletesen külön fejezet (lásd 7.) foglalkozik.

Egy try { } blokkba zárt utasításcsoporton belül bárhol előforduló kivétel hatására a program normális futása megszakad, és a vezérlés automatikusan a blokkot követő catch blokkra kerül. Itt a catch paramétereként megkapjuk a hiba okát is egy Exception típusú objektummal reprezentálva. A példaprogramunk egyszerűen kiírja a hiba okát.

A Java programváltozat legszembeötlőbb eltérései a következők:

- A blokk struktúra szerkezetében a Pascal begin kulcsszónak a {, míg a lezáró endnek itt a } jel felel meg.
- A főprogram maga is egy metódus.
- A változódeklaráció helye nem kötött és nem vezeti be külön kulcsszó.
- Az I/O műveletek nehézkesebbnek tűnnek.

A példaprogramokban nem látszik, de igen jelentős eltérés a két nyelv között, hogy a Pascal programozási nyelv érzéketlen a kis- és nagybetűkre, míg a Java a C++ hagyományait követve különbséget tesz azok közt.

E.1.2. Megjegyzések használata

A Java nyelvben a megjegyzések elhelyezésére több lehetőség is kínálkozik:

- `/* hosszabb megjegyzések */`
- `// egysoros megjegyzés`
- ```
  /**
  * dokumentációs megjegyzés
  */
  ```

A Pascal nyelv a megjegyzésekre a következő lehetőségeket biztosítja:

- `{ Borland cég fejlesztette Pascal verziókban használható }`
- `(* Standard Pascalban definiált forma *)`

E.1.3. A Java nyelv kulcsszavai

A következő táblázatban összefoglaljuk azokat a kulcsszavakat, amelyek a Pascal nyelvben is megvannak, de eltérő a jelentésük és ezért keveredést okozhatnak, illetve a Pascalban gyakran használatosak és a Java nyelvből teljesen hiányoznak:
  ```

	Java	Pascal
boolean	primitív típus (nem felsorolási)	primitív felsorolási típus
break	címkével mint paraméterrel lehet megadni a folytatás helyét	paraméter nélküli eljárás, csak ciklus törzsén belül használható
byte	8-bites előjeles egész	8-bites pozitív egész
char	16-bites Unicode karakter	8-bites ASCII karakter
dispose	-	new-val foglalt terület felszabadítása
do	csak hátultesztelő ciklus szervezésére használható	csak elöltesztelő ciklus szervezésére használható
else	csak if ág esetén jut szerephez	mind if, mind case esetén használható
goto	nem implementált, de foglalt kulcsszó	feltétlen vezérlésátadás
int	primitív, 32-bites előjeles egész típus	nem létezik, helyette az INTEGER kulcsszót kell használni
interface	új interface deklarációt bevezető kulcsszó	unitok deklarációs részét bevezető kulcsszó
long	primitív, 64-bites előjeles egész típus	nincs, helyette LONGINT
nil	-	nullpointer
null	üres referencia	-

E.1. ábra: Java-Pascal kulcsszavak összehasonlítása

E.1.4. Vezérlési szerkezetek

A Java nyelvben a következő vezérlési szerkezetek találhatóak meg: az elágazásokhoz az if-else és switch; ismétlésekre, ciklusszervezésre a for, while és do-while utasítások adottak. Ezek használata lényegében megegyezik a Pascalban megszokottal, ezért csak egy összefoglaló táblázatban mutatjuk meg a szintaxisukban lévő különbségeket:

Java	Pascal
if (*logikai kifejezés*) *utasítás1*; else *utasítás2*;	if *logikai kifejezés* then *utasítás1* else *utasítás2*;

switch (*egész kifejezés*) {
case *címke1* :
utasítások;
break;
...
default : *utasítások*;
}
	case *diszkrét kifejezés* of
	címke1 : begin
	utasítások;
	end;
	...
	else *utasítás*;
	end;

for (*kezdet*; *ciklusfeltétel*; *továbblépés*) *utasítás*;	for *változó*:=*kezdő* to *végérték* do *utasítás*;
while (*logikai kifejezés*) *utasítás*;	while *logikai kifejezés* do *utasítás*;
do *utasítások*; while (*logikai kifejezés*);	repeat *utasítások*; until *logikai kifejezés*;

Feltétlen vezérlésátadás

A Borland Pascal újításaként a goto utasítás elkerülésére bevezették a break és continue eljárásokat, melyek a standard Pascalban is meglévő exit és halt eljárásokhoz sorolhatók logikailag. Mind a két eljárás paraméter nélküli. A break hatására a program a ciklus utáni első utasításon folytatódik. A continue eljárás hatására a ciklus törzsének utasításai nem hajtódnak végre, hanem új iterációval elölről kezdődik a ciklus. Java nyelv esetén a goto utasítás nem is létezik – a biztonságosprogram elve értelmében –, helyette a continue és break utasítások lehetőségeit azzal bővítették, hogy azokkal egy megadott címkéjű ciklust lehet vezérelni. Ekkor a ciklusnak tartalmaznia kell egy címkét, amint ezt az alábbi program is mutatja:

```
public class CiklusTest {
  public static void main(String args[]) {
    címke:
    for (int i=0; i<3; i++) {
        for (int j=0; j<3; j++) {
            if (i+j == 3) continue címke;
            System.out.println("i = "+i+"; j = "+j);
        }
    }
  }
}
```

A program a következőket írja ki:

```
i = 0; j = 0
i = 0; j = 1
i = 0; j = 2
i = 1; j = 0
i = 1; j = 1
i = 2; j = 0
```

E.1.5. Típusok, deklarációk

Mind a Java, mind a Pascal erősen típusos nyelv, így megkövetelik, hogy minden változót egy – és csak egy – típusba soroljunk be. A típus leszűkíti, meghatározza a változókhoz hozzárendelhető értékek és a velük elvégezhető műveletek halmazát. Ennek nyomán a nyelv minden esetben megvizsgálja, hogy a kifejezésekben összeegyeztethető típusok vannak-e, azaz automatikus típuskonverzióval egyforma típusúra alakíthatóak-e. Ha az ellenőrzés fordítási időben elvégezhető, akkor hiba esetén a fordító megszakítja a fordítást és hibaüzenetet generál.

Konstansok a Java nyelvben

A Java programozási nyelv nem használ a Pascalban megszokott konstansokat, de lehetőséget nyújt konstans értékű változók, illetve adatmezők használatára. Ezt a létrehozott változó/adatmező neve elé kitett final kulcsszóval jelöljük. Példák:

```
Java                          Pascal
final static char SPACE=' ';  const SPACE=' ';
final static String HI="Hi!"; const HI='Hi!';
final int KETTO=2;            const KETTO=2;
```

Változók/adatmezők deklarációja

Az osztályok belsejében adatmezőket, a metódusok belsejében pedig változókat deklarálhatunk. A deklarációt egy-két alapszóval (pl. static, final) módosíthatjuk a következő formában:
módosítók típus változónév [=kezdeti_érték];

A Java nyelv tartalmaz néhány beépített, egyszerű adattípust, amelynek neve és értékkészlete a következő táblázatban található:

egész típusok	byte	8-bit kettes komplemens
	short	16-bit kettes komplemens
	int	32-bit kettes komplemens
	long	64-bit kettes komplemens
valós típusok	float	32-bit IEEE 754 lebegőpontos
	double	64-bit IEEE 754 lebegőpontos
karakter típus	char	16-bites Unicode karakter
logikai	boolean	true vagy false

A Java nyelv megengedi, hogy változódeklarációt egy metóduson belül bárhol elhelyezzünk. Egyetlen feltétel, hogy a változót mindenképpen deklarálni kell a felhasználása előtt. Élve ezzel a lehetőséggel, a változó deklarációja és a felhasználása közel helyezhető el egymáshoz, elkerülve ezzel rengeteg programozási hibát.

Sőt, a Java nyelvben megvan a lehetőségünk arra is, hogy egy ciklusváltozót ne deklaráljunk előre. A Pascal nyelv megköveteli, hogy a globális változókat csak a főprogram vagy unit deklarációs részében deklaráljuk, a lokális változókat pedig csak eljárás vagy függvény elején. Nézzünk erre egy Java példát:

```
public static void main(String[] args) {
  int n = 5;
  for (int i=1; i<=n; i++)
      System.out.println("\n" + i);
}
```

Az alábbi két esetben a deklaráció nem vihető be a kifejezésbe:

```
if (int s==0)        // Hibás !
while (int s==12)    // Hibás !
```

A felhasználó által definiált típusok

Az egyszerű, beépített típusokon túl a Java programozási nyelv csak kétfajta összetett adattípust tartalmaz: a programozó által definiálható osztályt (class), illetve a tömböt. Nincs meg a C++-ban ismert struct, union, typedef, sőt mutatók sincsenek!

A Java nyelvben csaknem minden objektum. Mint korábban láthattuk, egy programban az import utasításokon kívül minden utasítás osztályok belsejében szerepelhet csupán, nincsenek globális változók, globális eljárások. A nyelv ugyan tartalmaz néhány egyszerű adattípust, de ezeken felül minden egyéb adatszerkezet vagy valamilyen osztályba tartozó objektum, vagy tömb, ami ugyancsak egy speciális osztály. Mellesleg a nyelv tartalmaz beépített típusokat becsomagoló osztályokat (wrapper class) is: Boolean, Character, Double, Float, Integer.

Osztályok esetén is megfigyelhető, hogy a deklaráció-definíció sokkal egyszerűbb, mint a Pascal programozási nyelvben. Lássunk erre is egy példát:

```
//Java
class komplex {
  int re;
  int im;
  double abszolutertek() {
    return Math.sqrt(re*re+im*im);
  }
}

{Pascal nyelven}
type komplex = object
       re, im: integer;
       function abszolutertek(): real;
     end;
...

function komplex.abszolutertek(): real;
begin
  abszolutertek:=sqrt(re*re+im*im);
end;
```

Tömbök

A Java nyelv tömbjei is objektumok, ha kissé speciálisak is. Deklarálásuk kétféle szintaxissal megengedett, azaz a következő két deklaráció ekvivalens:

```
int a[] = new int[10];
int[] a = new int[10];
```

Ennek Pascal nyelvi megfelelője:

```
a : array [0..9] of integer;
```

Javaban a tömb legkisebb indexe mindig 0, és a virtuális gép ellenőrzi, hogy az index ne nyúljon túl a tömb határán; ha ez mégis megtörténne, kivétel keletkezik. Minden tömb objektumnak van egy **length** nevű mezője, amely a tömb aktuális méretét adja vissza. Ez azért is fontos, mert a program futása során ugyanaz a tömbváltozó más és más méretű tömbökre hivatkozhat.

Természetesen nemcsak a beépített, egyszerű típusokból képezhetünk tömböket, hanem tetszőleges típusból, beleértve másik tömböt is. A Java többdimenziós tömbjei mint tömbök tömbjei jönnek létre. Példa:

```
int a[][] = new int[10][3];
```

Ennek Pascal megfelelője :

```
a : array [0..9,0..2] of integer;
```

E.1.6. Kifejezések és operátorok

Az operátorok különbözőségét a két nyelv között a következő oldalon egy összehasonlító táblázat mutatja. A Java bitenkénti KIZÁRÓ VAGY operátor jele (^) eléggé félrevezető, mivel ugyanez a jel a Pascalban a Javaból teljesen hiányzó mutató típus kezeléséhez szükséges.

	Java	Pascal
Asszociativitás	balról jobbra	balról jobbra
Legmagasabb precedenciájú operátorok		
metódushívás	()	()
tömbindexelés	[]	[]
névminősítés	.	.
Egyoperandusú operátorok		
logikai tagadás	!	not
bitenkénti negálás	~	—
+/- előjel	+/-	+/-
növelés/csökkentés eggyel	++/–	inc/dec
címképzés	—	@
Multiplikatív operátorok		
szorzás	*	*
valós osztás	/	/
egész osztás	/	div
maradék	%	mod
Additív operátorok		
összeadás/kivonás	+/-	+/-
Összehasonlító operátorok		
kisebb [vagy egyenlő]	<[=]	<[=]
nagyobb [vagy egyenlő]	>[=]	>[=]
eleme-e?	—	in
Egyenlőség		
egyenlő	==	=
nem egyenlő	!=	<>
Logikai műveletek		
bitenkénti ÉS	&	and
bitenkénti kizáró VAGY	^	xor
bitenkénti VAGY	|	or
logikai ÉS	&&	and
logikai VAGY	||	or
Feltételes operátor	? :	—
Értékadás		
egyszerű értékadás	=	:=
értékadás művelettel	*/%+-&^|«»=	—

A new és a nem létező dispose operátorok

A szabad memória dinamikus használata részét képezi legtöbb Pascal nyelven megírt programnak. Ott a mutatótípus segítségével nyílik lehetőségünk arra, hogy dinamikusan használhassuk a rendelkezésünkre álló memóriaterületet. A változókhoz hasonlóan a Pascal nyelvben az objektumok helyfoglalása is történhet dinamikusan a heap-ben, ennek kezelését könnyen és hatékonyan el tudjuk végezni. Objektumok dinamikus létrehozásának menete a következő: deklarálunk egy objektumra mutató pointert, majd a new operátorral helyet foglalunk a számára. Például:

```
var
   pKor: ^Kor;
...
```

```
begin
  ...
  new(pKor);
```

A dinamikusan létrehozott objektumpéldányok helyfoglalásának a felszabadítása a heapből a dispose operátorral történik, például:
`dispose(pKor);`

A Java nyelvben is létező new operátor használata, jelentése merőben eltér a Pascal nyelven megszokottól. A new segítségével egy már meglévő osztályunkhoz új objektumot hozhatunk létre. Például:

```
class  Valami {
  public int i;
  public String s;
}
...

Valami sajat = new Valami();
```

Ez az operátor lefoglalja az új objektumunk számára a szükséges helyet, és erre a területre vonatkozó referenciával tér vissza. Ha az új objektum referencia típusú adattagjainak még nem definiált az értéke, akkor azok inicializálása egy újabb new operátor használatát, vagy meglévő objektumok átadását jelenti.

A Java nyelv dispose operátorral nem rendelkezik, így felmerül a kérdés, hogy a létrehozott objektumokat hogyan lehet felszabadítani. A Javanak semmilyen explicit eszköze sincs arra, hogy egy objektumot megszüntessen, egyszerűen csak nem kell hivatkozni rá. A legegyszerűbb megoldás, hogy az objektumra hivatkozó meglévő referenciáknak null referenciát adunk értékül. A könyv előző fejezeteiben említett úgynevezett automatikus takarítást a szemétgyűjtő eljárás végzi, melynek működtetése teljes egészében a futtató rendszer feladata.

E.2. A Java nyelv mint objektumorientált nyelv

Míg a Pascal nyelv lehetőséget biztosít az objektumorientált módon történő programozásra, addig a Java nyelven csak ezt a programozási stílust követve írhatunk programokat.

Egy példa segítségével nézzük meg, hogyan néz ki egy osztály definíciója Java nyelven:

```
public class Kartya {

// OSZTÁLYSZINTŰ ADATTAGOK
  private static final String[] szinek={"C", "D", "H", "S"};
  private static final String[] ertekek={"A", "2", "3", "4", "5", "6", "7", "8",
                                         "9", "10", "J", "Q", "K"};
  private static final Kartya[]
                       kartyak=new Kartya[szinek.length*ertekek.length];
  private static int kovetkezoKartya;

// OSZTÁLYSZINTŰ METÓDUSOK
  public static void keveres() {
    int masikKartya;
    Kartya temp;

    for (int kartya=0; kartya<kartyak.length; kartya++) {
```

```
        masikKartya=(int)(Math.random()*kartyak.length)%kartyak.length;
        temp=kartyak[masikKartya];
        kartyak[masikKartya]=kartyak[kartya];
        kartyak[kartya]=temp;
    }
    kovetkezoKartya=0;
  }
  public static Kartya deal() {
    return kovetkezoKartya==kartyak.length ? null: kartyak[kovetkezoKartya++];
  }
// STATIKUS INICIALIZÁTOR
// Inicializálja a kártya csomagot
  static {
    int kartyaSzam=0;
    for (int kartyaSzin=0; kartyaSzin<szinek.length; kartyaSzin++)
        for (int kartyaErtek=0; kartyaErtek<ertekek.length; kartyaErtek++)
            kartyak[kartyaSzam++]=new Kartya(kartyaSzin, kartyaErtek);
    keveres();
  }
// EGYEDSZINTŰ ADATTAGOK
// Adatmezők, melyek az aktuális kártya színét és értékét tárolják
  private int ertek;
  private int szin;
// EGYEDSZINTŰ METÓDUSOK
// Olyan metódusok, melyek egy adott kártya objektum színét és értékét
// adják vissza sztringként
  public String mekkoraErtek() {
    return ertekek[ertek];
  }
  public String miaSzine() {
    return szinek[szin];
  }
// metódus, mely megadja, hogy az aktuális kártya, vagy az argumentum kártyának
// nagyobb-e az értéke
  public boolean nagyobbErtek (Kartya masikKartya) {
    return ertek>masikKartya.ertek;
  }
// KONSTRUKTOR
  private Kartya(int ujErtek, int ujSzin) {
    ertek=ujErtek;
    szin=ujSzin;
  }
}
```

A példában a különböző szintű deklarációk sorrendje persze felcserélhető. A Kartya osztály konstruktora minden egyes új kártya objektum létrehozásakor meghívódik. A statikus inicializátor viszont csak egyszer, az osztály betöltésekor hajtódik végre.

E.2.1. Objektumhivatkozások

A Javaban a Pascal nyelvhez teljesen hasonlóan a pont operátorral lehet egy objektum mezőire hivatkozni. Például:

```
kartya1.mekkoraErtek();
kartya2.miaSzine();
```

E.2.2. Egységbezárás

Az adatok és az azokat kezelő alprogramok egységbezárása (encapsulation) fontos sajátossága minden objektumorientált nyelvnek. Általános követelmény továbbá, hogy az objektum adatmezőit csak metódusok használatával érhessük el. Sajnos az ezen elvet megvalósító public, illetve private kulcsszavakat a Pascal nyelvben csak unitokon belül használhatjuk, mivel a főprogramban e két kulcsszót figyelmen kívül hagyja a fordító.

A Java nyelv esetén az elv könnyen megvalósítható, mint ahogy ezt az előző példa is mutatja. A példában egy Kartya objektumnak van szin és ertek mezője, de ezeket nem lehet közvetlenül elérni, kifelé ezek nem láthatóak. Csak a mekkoraErtek(), miaSzine(), nagyobbErtek() metódusok láthatók kívülről.

A Java nyelv további újdonsága a Pascal nyelvhez képest, hogy lehetőséget biztosít névtúlterhelésre (overloading), azaz arra, hogy ugyanolyan névvel lehessen elnevezni különböző metódusokat. Az egyetlen feltétel ezzel kapcsolatban, hogy a metódusok szignatúrája különböző legyen. A megfelelő metódus kiválasztása ugyanis az átadásra kerülő argumentumok alapján történik. Például:

```
class Pelda {
  public static int kettozdMeg(int x) {return 2 * x;}
  public static String kettozdMeg(String x) {return x + x;}
}
```

A példában szereplő mindkét metódusnak azonos a neve, de eltérőek az argumentumaik és a visszatérő értékeik. Használatuk:

```
int egeszEredmeny = Pelda.kettozdMeg(3);
String stringEredmeny = Pelda.kettozdMeg("Ha");
```

Az egeszEredmeny értéke 6 lesz, mivel ilyenkor az első kettozdMeg metódust használjuk, míg a stringEredmeny a második változat alapján a "HaHa" sztringet kapja értékül.

Az objektumorientált programozás kulcsfogalma az öröklődés és a származtatott osztályok tulajdonságainak megváltoztatásából eredő sokalakúság (polimorfizmus). Mivel a Pascal nyelvet csak később tették alkalmassá objektumorientált módon történő programozásra, ezért a polimorfizmus megvalósítása kissé nehézkes. Azokat a metódusokat ugyanis, melyeket polimorfikusan szeretnénk használni, a virtual direktívával külön meg kell jelölni. Ez azonban eléggé memóriaigényes, hiszen a fordító a virtuális metódusok címét osztályonként különálló, nagyméretű, virtuális metódustáblának (VMT) nevezett táblázatban tárolja.

A Java programozási nyelvben nincs szükség semmilyen direktívára, itt minden metódus virtuálisnak tekinthető. A Pascalban meglévő statikus típus szerinti (nem virtuális metódusok esetén) metódushívást a Javaban (objektumszinten) még típuskényszerítéssel sem lehet megvalósítani, mert minden hivatkozás feloldása dinamikusan, a dinamikus típusnak megfelelően történik.

E.2.3. Láthatóság

A Java nyelven a programozók megadhatják, hogy az általuk definiált egyes osztályok, illetve az osztályok adattagjai, metódusai milyen körben használhatók. Erre a célra az ún. hozzáférést specifikáló (access specifier) módosítók használatosak. Leggyakoribbak a nyilvános (public) és a magán (private) módosítószavak. Létezik egy úgynevezett félnyilvános vagy baráti (friendly) hozzáférés is, ez az alapértelmezés, ha nem adjuk meg a hozzáférés módját.

Osztályokra csak a nyilvános vagy a félnyilvános hozzáférés vonatkozhat: nyilvános osztályokat a programunkban bárhol használhatunk, bármelyik csomagból (package) importálhatunk. Félnyilvános osztályokat csak az adott fordítási egységen belül lehet használni.

Egyed-, illetve osztályváltozókra, metódusokra a nyilvános és baráti hozzáférés a fentivel azonos jelentésű, ezen túl a magán változók, metódusok csak az adott osztályon belül láthatóak.

E.3. A Java nyelv új lehetőségei

A Java interfész (interface) fogalma teljesen ismeretlen a Pascal programozók számára. Egy interfész nem más, mint konstansok és metódusok lenyomatának a gyűjteménye, amelyek egy osztályban való egyidejű meglétét a programozó fontosnak tartja. Maga az interfész a metódusok törzsét nem adja meg, de írhatunk olyan osztályokat, amelyek megvalósítják, implementálják az interfészt, azaz konkrétan definiálják az interfészben felsorolt valamennyi metódust.

Ennek nagy haszna, hogy az öröklődési hierarchiában nem rokon osztályok is viselkedhetnek hasonlóan, az objektumorientált terminológia szerint azonos protokoll – metódusgyűjtemény – segítségével kommunikálhatnak.

Ha egy osztály megvalósít egy interfészt, azt a class *A* implements *I* formában lehet kifejezni, de ennek az a következménye, hogy *A*-ban az összes *I*-beli metódust meg kell valósítani. A Javaban, akárcsak a Pascalban többszörös öröklés ugyan nincs, de egy osztály egyidejűleg több interfészt is megvalósíthat. Az interfészben deklarált minden metódus nyilvánosnak és absztraktnak (public abstract) tekintendő, a módosítókat – sem ezeket, sem másokat – nem célszerű kiírni. Szükség esetén egy interfész deklarálhat konstansokat is, ezek mind véglegesek, nyilvánosak és statikusak (final public static). Erről részletesebb ismertetés az Interfészek című fejezetben (lásd 5.) található.

A Java programozási nyelv törekedik a minél biztonságosabb programok létrehozására, ezért lehetővé teszi a hibakezelést, melyet a standard Pascal nyelv egyáltalán nem támogatott. Ez a hibakezelés két dolgot jelent, egyrészt a programot a hiba fellépése után olyan állapotba kell hozni, hogy futását folytatni tudja, vagy ha ez nem lehetséges, akkor néhány dolog végrehajtása után „biztonságos állapotban" kell leállnia. A Java hibakezelése kivételkezelésen alapszik, melynek célja futás közben a fellépő hibák lekezelése. A konkrét megvalósításról részletesebb ismertetés a Kivételkezelés fejezetben (lásd 7.) olvasható.

A Java nyelv lehetőséget biztosít arra is, hogy egy program végrehajtása ne csak egyetlen szálon fusson, hanem párhuzamosan több szál is aktív legyen a programban. Ezt nyelvi szinten támogatja a Thread osztály, a Runnable interfész, a ThreadGroup osztály, illetve a synchronized kulcsszó. A szálaknál is, mint az egyéb párhuzamos tevékenységeknél, a következő alapvető problémák merülnek fel:

- Szálak létrehozása, futtatása;
- Szálak szinkronizációja, futásuk összehangolása;
- Kommunikáció a szálak között;
- Ütemezés, a központi egység erőforrásának felosztása a szálak között.

Ezen problémák megoldásáról, illetve a konkrét használat leírásáról a többszálú programozásról szóló fejezetben (lásd 15.) olvashatunk részletesen.

Reméljük, hogy az előzőekben bemutatott lehetőségek felkeltették az érdeklődést a Java programozási nyelv iránt és a kezdeti nehézségeken is sikerült egy kicsit könnyíteni. Néhány program megírása után, reméljük, hogy a Java szemlélete egységes, logikus rendszerré áll össze.