**Debreceni Egyetem**
**Informatikai Kar**

# A Java nyelv tanítása középiskolában

**Témavezető:**
Kósa Márk Szabolcs
Egyetemi Tanársegéd

**Készítette:**
Horváth László András
Informatika tanár szak

Debrecen
2009.

# Tartalomjegyzék

# Előszó

„Napjainkra a számítógép (és különösen a világhálózat) univerzális szellemi munkaeszközzé vált. Az írás kialakulása óta a számítógép a legnagyobb találmány. Közel az idő, amikor az informatikai tudás éppen olyan fontos lesz, mint az írni-olvasni tudás. Az információs társadalom küszöbén állunk. Az iskola feladata, hogy felkészítsen az információs társadalomban való életre.

A munkába állás esélyei azok számára nagyobbak, akik számítástechnikai ismeretekkel rendelkeznek. Ha valóban esélyegyenlőséget akarunk biztosítani a gyerekek számára, akkor az informatika ismeretek megszerzését is lehetővé kell tenni az általános képzés keretein belül. …”

Korunkban három nagy tudásterület alapozza meg a kommunikációt és a megismerést. Ezek a következők:

- beszéd, írás, olvasás (idegen nyelven is);
- matematika (és logikus gondolkodás);
- korszerű informatika (számítógépes és kommunikációs technológiák)

E három terület azért hasonlít egymáshoz, mert mindegyikük általános eszközöket nyújt az adat, az információ és a tudás leírására, feldolgozására és a kommunikációra. Az informatika gyakorlat és tudomány, amely olyan eszközöket ad a kezünkbe, amelyek minden tudományban használhatóak, sőt majdnem minden szellemi munkában. Kicsit hasonlóan ahhoz, ahogyan a matematika vagy a nyelv is használható a különböző tudományokban és a gyakorlati munkában.

A gyakorlatias informatika azzal, hogy információkezelésre, logikus gondolkodásra, problémamegoldásra tanít és praktikus alkalmazói tudást, készséget és képességet fejleszt, korszerű informatikai eszközök alkalmazásával, valamennyi tantárgy tanulását segíti.[16]

A Végh András által megfogalmazott mondatok mottóként is szolgálhatnak dolgozatom elején. Lépést kell tartani a fejlődéssel az élet minden területén, így az oktatásban is.

# Bevezetés

A középiskolai oktatásban az 1990-es évek első felétől megjelent már a NAT-ban (Nemzeti Alaptanterv) a keret, és egyéb tantervekben a számítástechnika oktatása is.

A középiskolák többségében a Turbo Pascal vagy Basic, C vagy C++ programnyelvet oktatják. A szakdolgozatomban megpróbálom bemutatni, hogy egy fiatal programozási nyelvet is be kellene iktatni az oktatott programnyelvek sorába, ez a Java. Azokban az iskolákban, ahol a C++ nyelvet tanítják könnyebb az integrálás, mivel a Java nyelv ebből fejlődött ki. A Java elsajátításához nem szükséges a C vagy C++ programozási előképzettség, csupán ajánlott.

A Java nyelv egyszerű, objektum-orientált, biztonságos, és nem utolsó sorban hordozható programozási nyelv. Elterjedésében nagy szerepet játszott az internet rohamos fejlődése. A Java az internetes honlapok készítéséhez kiváló eszköz, de önmagában is teljes értékű programozási nyelv. Előnye, hogy könnyen megtanulható, korszerű, a számítástechnikai alkalmazások egyre keresettebb programnyelve.

Ma a leggyakoribb felhasználási területe a mobil alkalmazások és a szerver oldali szoftverfejlesztés, de az utóbbi években a Java alapú DeskTop alkalmazások is egyre nagyobb számban és egyre jobb minőségben fejlődnek. Ez indokolja, hogy a tananyag frissítés kapcsán beépüljön a középiskolai tananyagba ennek a programnyelvnek a tanítása.

A középiskolában oktatott tananyag természetesen nem nyújthat minden részletre kiterjedő ismeretet, de segítséget adhat a programnyelv megismeréséhez, az alapok elsajátításához.

A szakdolgozatom két nagy egységre bontható: Az első rész a pedagógiai rész. A tananyag elhelyezése az informatika oktatásában. A második egységben a tanterv és tanmenetek figyelembevételével a szakmai részre fektetem a hangsúlyt. Ebben a Java programozás oktatásában fő szerepet kapó anyagrészeket mutatom be, illetve ellenőrző feladatokat írok le.

A szakmai részben található példa programokkal és programrészekkel próbálom bemutatni a Java nyelv szépségeit, előnyeit.

# A NAT-ban megfogalmazott célok és feladatok

**Az informatika tárgy oktatásának a NAT szerint megfogalmazott céljai, feladatai:**

A tantárgy oktatásának alapfeladata a tanulók digitális kompetenciájának fejlesztése, de informatikai alapismeretek oktatás célrendszere összhangban van a kulcskompetenciákkal, mely magába foglalja:

- A társadalom információs technológiáinak magabiztos és kritikus használatát a munka, a kommunikáció és a szabadidő terén. Ez a következő készségeken, tevékenységeken alapul: az információ felismerése, keresése, értékelése, tárolása, előállítása, feldolgozása, bemutatása és cseréje.
- Az informatikában fontos szerepet kap a matematikai kompetencia fejlesztése, azaz a matematikai gondolkodás és problémamegoldás fejlesztése, az algoritmusok alkalmazásának képessége, mindennapi problémák megoldása és a világ rendszereinek, folyamatainak informatikai (számítástechnikai) modellezése során.
- A természettudományos kompetencia készséget és képességet jelent arra, hogy ismeretek és módszerek sokaságának felhasználásával magyarázzuk, és előre jelezzük a természeti folyamatokat, illetve a rendszerek mozgását, tulajdonságait. A természettudományos és műszaki-technikai kompetencia magában foglalja a kételkedő, kritikus és kíváncsi attitűdöt, a megismerés és konstruálás iránti vágyat, valamint a biztonsággal és fenntarthatósággal kapcsolatos etikai kérdések iránti érdeklődést is.
- Az informatikaoktatás elősegíti a kezdeményezőképesség és a vállalkozói kompetencia tudatos fejlesztését is. A gyakorlati feladatok megoldása során olyan készségeket és képességeket fejleszt, mint a tervezés, szervezés, irányítás, elemzés, kommunikálás, a tapasztalatok értékelése, egyénileg és csapatban történő munkavégzés. A programozás oktatása tág lehetőségeket teremt az innovatív és kreatív problémamegoldásra.
- Az információs termékek alkotása, tervezése és gyártása során fontos szempont a mű, illetve a dokumentum esztétikai megjelenése és kifejezőképessége. Az informatikaoktatás az esztétikai-művészeti tudatosság és kifejezőképesség kompetenciafejlesztésének egyik korszerű területe.

A hétköznapi életből vett, változatos feladatok szövegezése lehetőséget biztosít arra, hogy tanulóinkat az egészséges életmódra, a környezettudatos és kritikus fogyasztói magatartásra neveljük.

A legtöbb középiskolában az informatika tantárgy oktatására heti két óra áll rendelkezésre. Ez az időkeret csak arra elég, hogy ízelítőt kapjanak a diákok az informatika adta lehetőségekből, hogy egy olyan alapot kapjanak, amelynek segítségével a fejlődésük, az új iránti fogékonyságuk lehetségessé válik. Természetesen mind ez feltételezi az általános iskolai kerettanterv alapján elsajátított fogalmaknak és jártasságoknak a minimális körét, amelyek ismeretére építhetünk.

A Java programozási nyelv ismereteinek a megalapozását az alábbi felsorolás szerint építeném be a tantárgy tematikájába:

<u>9. – 10. évfolyamon:</u>

- Java alapok
- Nyelvi elemek
- Alapvető objektumok
- Futtatókörnyezet

<u>11. – 12. évfolyamon:</u>

- Adatszerkezet, generikus strukturálás
- Java I/O

Az ismereteket a jelenlegi rendszerbe építve, azok kiegészítéseként, mint egy modult iktatnám be a tananyagba. Ehhez elkészítettem az óratervezetben az órafelosztást.

## Óratervezetek a 9. évfolyam számára

| | **Témakör** | **Éves óraszám** | **Ebből Jávára fordítható időkeret** |
|---|---|---|---|
| 1. | Az informatikai eszközök használata | 14 | 2 |
| 2. | Információ – kommunikáció | 16 | 6 |
| 3. | Számítógépes grafika | 10 | 3 |
| 4. | Szövegszerkesztés | 10 | 2 |
| 5. | Weblap készítése | 7 | 4 |
| 6. | Prezentáció | 11 | 3 |
| - | Szabadon felhasználható órakeret | 6 | 3 |
| | **Összesen:** | **74** | **23** |

## Óratervezetek a 10. évfolyam számára

| | **Témakörök** | **Éves óraszám** | **Ebből Jávára fordítható időkeret** |
|---|---|---|---|
| 1. | Információs társadalom | 7 | 3 |
| 2. | Táblázatkezelés | 20 | 6 |
| 3. | Adatbázis-kezelés | 25 | 7 |
| 4. | Algoritmusok és adatok | 15 | 4 |
| 5. | Könyvtárhasználat | 3 | 1 |
| - | Szabadon felhasználható órakeret | 4 | 2 |
| | **Összesen:** | **74** | **23** |

## Óratervezetek a 11. évfolyam számára

| | **Témakörök** | **Éves óraszám** | **Ebből Jávára fordítható időkeret** |
|---|---|---|---|
| 1. | Szövegszerkesztés | 20 | 4 |
| 2. | Táblázatkezelés | 20 | 4 |
| 3. | Adatbázis-kezelés | 20 | 6 |
| 4. | Prezentáció és grafika | 11 | 3 |
| 5. | Információs hálózati szolgáltatások | | 2 |
| 6. | Weblapszerkesztés | | 2 |
| - | Szabadon felhasználható órakeret - ismétlés | 3 | 2 |
| | **Összesen:** | **74** | **23** |

## Óratervezetek a 12. évfolyam számára

| | **Témakörök** | **Éves óraszám** | **Ebből Jávára fordítható időkeret** |
|---|---|---|---|
| 1. | Szövegszerkesztés | 10 | 3 |
| 2. | Táblázatkezelés | 20 | 4 |
| 3. | Adatbázis-kezelés | | |
| 4. | Prezentáció és grafika | 11 | 3 |
| 5. | Információs hálózati szolgáltatások | 10 | 5 |
| 6. | Weblapszerkesztés | 15 | 7 |
| - | Szabadon felhasználható órakeret - ismétlés | 3 | 1 |
| | **Összesen:** | **68** | **23** |

**Célok**

A tanulók a tantárgy keretében ismerkedjenek meg a nyelv alapvető elemeinek használatával és legyenek képesek egyszerűbb programozási feladatok megoldására a Java programozási nyelvvel. Ismerjék, és megfelelően tudják alkalmazni a egyes vezérlési struktúrákat, egyedi feladatok elvégzésére tudjanak saját osztályokat és metódusokat írni. A tanulók emellett

ismerjék, és megfelelően tudják használni a Java Fejlesztői Környezetet (NetBeans). Segítség nélkül tudja használni a szakkönyveket és a Java dokumentációját is.

**Követelmények, elvárások**

A tanuló a modul elsajátítása után:

- Meg tud írni egy Java programot, és azt le tudja fordítani, futtatni.
- Tudja, hogy a Java osztálykönyvtár milyen hasznos, osztályokat tartalmaz.
- Fel tudja sorolni, hogy egy Java program milyen programozási egységekből épül fel.
- Ismeri a Java nyelv operátorainak kiértékelési sorrendjét (precedencia).
- Fel tudja sorolni a Java részeit.
- Képes legyen a programjában aritmetikai műveleteket végezni.
- Ismeri az egyszerű adattípusokat, és ezeket használja.
- Tudja felsorolni az elágazások és a ciklusok fajtáit.
- Tudja mire való a „break” és „continue” utasítás
- Ismerje, hogy a metódusok hogyan adhatnak vissza eredményt a hívónak.
- Tud metódusokat írni és alkalmazni.
- Képes egy- és kétdimenziós tömböket deklarálni, inicializálni, és azokat használni.
- Képes legyen karaktersorozatokkal alapvető műveleteket végezni.

**Szükséges eszközök**

A Java nyelv oktatásához szükséges egy számítógép, amelyre installálva van:

- a Java Development Kit (JDK),
- NetBeans vagy Eclipse,

Ezen kívül a tanulónak rendelkeznie kell legalább egy tankönyvvel.

**A Java programozási nyelv oktatásához ajánlott szakirodalmak:**

Tanuljuk meg Java programozási nyelvet 24 óra alatt – Rogers Cadenhead – Kiskapu kft Budapest 2006,

Java programozási nyelv I – Móricz Attila – LSI Oktatóközpont A Mikroelektronika Alkalmazásának Kultúrájáért Alapítvány Budapest 1997,

Java Alapismeretek - Móricz Attila – LSI Oktatóközpont A Mikroelektronika Alkalmazásának Kultúrájáért Alapítvány Budapest 1997.

Az angol nyelv ismerete segíti ennek is, mint minden más programnyelvnek az elsajátítását.

# Programozási nyelvekről általában

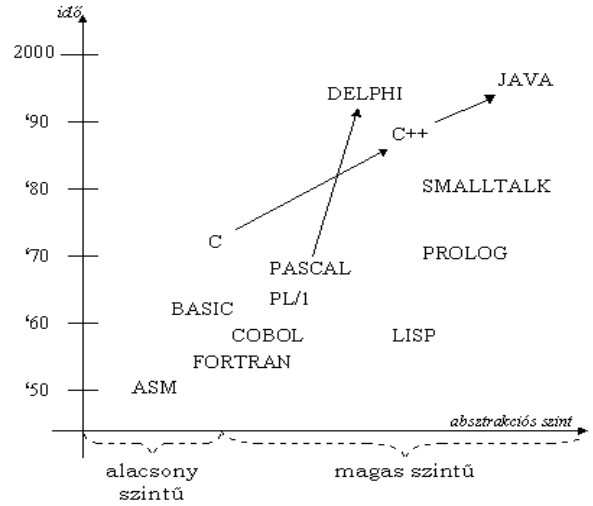
[16] A számítástechnikában használatos nyelveket a nyelv által megoldandó probléma alapján két nagy csoportra oszthatjuk:

- leíró jellegű nyelvek egy állandó állapotot (tényeket) fogalmazhatunk meg bennük, nem pedig cselekvést. pl. HTML, VRML
- (procedurális) programnyelvek algoritmikus cselekvést (tennivalók sorozatát) fogalmazunk meg bennük. Ilyen elven működnek a Neumann elvű számítógépek. Innentől nyelven elsősorban ilyen programnyelvet értünk.

A nyelv absztrakciós szintje alapján:
- alacsony szintű nyelvek nehezebb a programírás, de a lefordítandó program közelebb áll az gép nyelvéhez (a gépi kódhoz), így nehezen hordozható (más típusú gépekre), viszont több beleszólásunk van a gépközeli dolgokba pl. assembly
- magas szintű nyelvek könnyebb programot írni (pontosabban egy emberközeli probléma megfogalmazása könnyebb). A nyelv által nyújtott kényelem és az gép utasításkészlete közötti űrt a fordító/értelmező programnak kell áthidalnia. pl. Java.

## A fontosabb programnyelvek

**Java programozási nyelv megjelenése és elhelyezkedése más programozási nyelvek között**



Az ábra jól szemlélteti az egyes programozási nyelvek fejlődését. A nyilak jelzik azt is, hogy melyik nyelvből, melyik fejlődött ki, például a C nyelv tovább fejlesztett változata a C++, majd ebből született meg a JAVA.

**A ma is élő, legelterjedtebb általános nyelvek és legfontosabb jellemzőik:**

- assembly
  - gépi kódú programozást segítő nyelv
  - a megírt program nehezen hordozható
  - ma már csak az operációs rendszerek készítői, ill. a hardvergyártók programoznak ilyenben
- PASCAL
  - eredetileg a strukturált programozás tanulónyelvének szánták
  - továbbfejlesztett változata, a TURBO PASCAL a modulok, illetve ma már az OOP-t is támogatja
  - a valós életben a C miatt nem tudott érvényesülni, illetve a DELPHI-ben él tovább
- DELPHI
  - alapja az objektum-orientált Turbo Pascal
  - fő erőssége a korszerű és hatékony vizuális fejlesztőfelület
  - fő korlátja, hogy egy céghez (Borland), és egy platformhoz (Windows) kötődik
- C
  - alacsony és magas szintű nyelvként is szokták emlegetni, mert hatékony is, könnyű is programozni
  - minden (fontos) géptípusra van C fordító - (viszonylag) hordozható nyelv
  - régen minden általánosat C-ben írtak, ma a helyét egyre jobban átveszi a C++. Ahol megmarad, az a rendszerprogramozás
- C++
  - a C nyelv objektum-orientált továbbfejlesztése
  - ma már a legtöbb géptípusra van C++ fordító is - elég hordozható nyelv
  - napjainkban minden (általános) programot ebben írnak

- Java
    - egy képzeletbeli (virtuális) Java-gép programnyelve
    - szintaktikája nagyon hasonlít a C++ -éhoz
    - a nyelv szabványos részét képezik a leggyakrabban kellő alapkönyvtárak, mint pl. a grafika, felhasználói felület, hálózat programozás, adatbáziskezelés.
    - Java nyelven lehet appleteket írni, amelyek beszúrhatók HTML oldalakba is
    - tökéletesen hordozható
    - az Internet-programozás fő nyelve

## A Java rövid története

A Sun társalapítója, Bill Joy azt mondta, hogy a Java „tizenöt évnyi erőfeszítés végeredménye, ami arra irányult, hogy jobb, megbízhatóbb módot találjunk a számítógépes programok írására." A dolog azonban ennél kicsit bonyolultabb volt. A Javat a Sun egyik mérnöke, James Gosling fejlesztette ki 1990-ben, és célja egy olyan nyelv létrehozása volt, amelyen intelligens eszközök (interaktív TV-k, mindentudó sütők, katonai műholdak stb.) programját lehet megírni. Gosling a programírást C++ nyelven kezdte, de elégedetlen volt az eredménnyel, ezért bezárkózott az irodájába, és megalkotott egy másik nyelvet, ami jobban megfelelt az igényeinek.

Gosling az új nyelvet Oaknak („tölgy") nevezte el, mert az irodája ablakából éppen egy tölgyfára látott. A nyelv részévé vált a Sun stratégiájának, ami arra irányult, hogy milliókat keressenek az interaktív televíziózás elterjedésével. Ez máig nem következett be bár a TiVo és a ReplayTV kísérleteznek, Gosling új nyelve azonban önálló életre kelt.

A Sun már éppen készen állt arra, hogy beszüntesse az Oak fejlesztését és a rajta dolgozókat a vállalat más területeire irányítsa, amikor a Világháló (World Wide Web) népszerűvé vált.

A körülmények szerencsés összjátéka folytán azok a jellemzők, amelyek Gosling nyelvet különösen alkalmassá tették az okos készülékek programozására, a Weben is hasznosnak bizonyultak. A Sun fejlesztői kidolgozták annak a módját, hogy miként lehet programokat biztonságosan futtatni a weblapokról, és a nyelv új céljához új nevet is választottak: a Javat.

Bár a Java más célokra is alkalmas, a Világhalónak köszönheti, hogy a nemzetközi figyelem rá irányult. Ha egy programozó egy Java programot egy weblapon helyez el, azonnal

elérhetővé teszi bárki számára. Mivel erre először a Java volt képes, a sajtó sztároknak kijáró fogadtatásban részesítette a számítógépes nyelvek között. 1996-ra a nyelv a csúcsra jutott.

A Java nyelvnek eddig hat főbb változata jelent meg:

- 1995 ősze: Java 1.0 - az eredeti kiadás;
- 1997 tavasza: Java 1.1 - frissítés, amelyben javították a felhasználói felületek létrehozásának és kezelésének módján;
- 1998 nyara: Java 2 version 1.2 - a Java 1.0-nál több mint háromszor nagyobb változat, amely a Javat versenyképessé tette az általános célú programozási nyelvek között;
- 2000 ősze: Java 2 version 1.3 - gyorsabban futó Java programokat eredményező és továbbfejlesztett multimédiás szolgáltatásokat tartalmazó kiadás;
- 2002 tavasza: Java 2 version 1.4 - jelentős frissítés, amely kibővítette az internetes szolgáltatásokat, valamint az XML-kezelési és szövegfeldolgozási képességeket;
- 2004 tavasza: Java 2 version 5 - a legújabb kiadás, amely az olyan kiegészítések révén, mint az automatikus adatátalakítás, megbízhatóbbá és könnyebben megírhatóvá teszi a programokat.

## Java szerkezete

[17] A Java programok *osztályok*ból épülnek fel. Egy programot meg lehet írni úgy is, hogy magunk írjuk meg minden részét, a gyakorlatban ez teljesen felesleges, mivel rengeteg előre megírt osztály áll a programozó rendelkezésére. Ide tartoznak az ún. *alapcsomagok*, amelyek a Java alap-osztályait tartalmazzák, és a Java fejlesztői környezet részét alkotják. Ezek az osztályok megkönnyítik a programozást, hiszen rengeteg időt és energiát lehet megspórolni azzal, hogy előre megírt és tesztelt részekből állítjuk elő programunkat. A szoftver-újrafelhasználás egyben az objektum-orientált programozás egyik alapelve. Háromfajta osztályt lehet felhasználni a programozás során, ezek az alapcsomagokat tartalmazó *osztálykönyvtár osztályai*, a programozó által írt saját osztályok, és a mások által írt osztályok.

A Java programok legkisebb önálló egységei az osztályok. Az osztályok metódusokat tartalmaznak, melyek a műveletvégzés legfontosabb eszközei. Az osztályok szintén tartalmazzák az objektumok struktúráját, valamint az objektumok létrehozásának különböző módjait.

Működése során a program példányosítja az osztályokat, vagyis a modellek sémája szerint egy konkrét objektumot hoz létre. Egy ilyen objektum belső állapottal rendelkezik, amely egyrészt befolyásolja a műveletek végrehajtását az objektumon, másrészt a műveletek hatására változhat.

Egy objektum modellje, vagyis osztálydefiníciója két részből áll. Az egyik rész deklarálja azokat a változókat, amelyekkel az objektum állapota leírható, és amelyek értékeiben az azonos típusú osztályok különbözhetnek egymástól. Az osztály minden példánya önálló készlettel rendelkezik ezekből a változókból. A másik rész az objektum viselkedését meghatározó metódusokat tartalmazza. A metódus meghívásakor meg kell adni, hogy az osztály melyik példányra érvényes. A metódus futása közben mindig van egy aktuális példány, amelynek a változóival a metódus dolgozik.

Az objektumorientált programozás a típusokat műveleteikkel együtt tekinti értelmes egységnek. Ebből a szempontból az osztálydefiníció nem más, mint egy típus teljes definíciója. A példányváltozók határozzák meg a típusértékek halmazát, a metódusok pedig a típusműveleteket.

Az adatok és az őket kezelő műveletek szoros összekapcsolása osztályok formájában és az adatok elrejtése más osztályok műveletei elől együttesen az adatabsztrakciót, az objektumorientált programozás egyik alapelvét testesíti meg.

# Tanulási feladatok és útmutatások

## A Java oktatásának egységei

A megtanítandó ismereteket az alábbi témakörökben tervezem megvalósítani:

### Alapfogalmak

Ebben a témakörben a fontosabb programozással, kapcsolatos fogalmakkal ismerkedhet meg a tanuló.

### Forrásszöveg elkészítése

Ebben a témakörben egy forrásszöveg elkészítésével ismerkedhet meg a tanuló

### Operátorok

Operátorok című rész a Java nyelvben szereplő operátorokat mutatja be. Példaprogramon keresztül ismerhetik meg a ++ és a - operátorok használatát. Ez segíti a tanulók számára a könnyebb megértést és elsajátítást.

### Adattípusok

Ebben a témakörben az adatszerkezetek és algoritmusok, egyszerű- és összetett adattípusokról szerezhet ismereteket a tanuló.

### Változók

Itt kiemelten foglalkozunk a változókkal és a programokban történő adat tárolásával és módosításával. Itt jelennek meg az utasítások és kifejezések, és a változok típusának meghatározása is. Ilyenek az egész és lebegőpontos számok, karakterek és karakterláncok, a boolean változótípus. A fejezet egy egyszerű programmal ér véget, mely egész számokat add össze. Ezeknek az elsajátítása minden programozási nyelvnek kulcseleme.

### Elágaztató utasítások

Ebben a fejezetben a tanulók elsajátíthatják a döntéshozással kapcsolatos teendőket példaprogramokon keresztül. Itt tanulják meg a feltételek ellenőrzését különböző utasítások segítségével, ilyen például az `if` és a `switch` utasítás.

**Műveletek ismétlése**

A műveletek ismétlése (ciklusok) fejezetben ismerhetik és sajátíthatják el a tanulók a ciklusok helyes használatát és felépítését. A `for`, a `while`, `do-while` ciklus közti különbségekre és egyezésekre is felhívom a figyelmet. A tanulók megismerhetik a `break` és a `continue` utasítások jelentősségét is.

**Tömbök**

Ez a témakör az adatok tömbökben tárolásának módjait, valamint a szavak, mondatok, karakterek feldolgozási lehetőségeit mutatja be, példákon keresztül.

**Az OO nyelvek**

Itt található az objektumorientált nyelvekkel kapcsolatos ismeretek.

**Objektum - öröklődés**

Ez a rész az objektumokkal kapcsolatos fogalmakat tartalmazza. Betekintést kapnak az objektumok használatáról.

Az öröklődés fejezetben sajátíthatják el az öröklődéssel kapcsolatos fogalmakat és ennek jelentősségét. A viselkedés és öröklődés, az öröklődés létrehozása használatát ismerhetik meg program részleteken keresztül.

**Csomag**

A csomagok című részben rövid áttekintést kapnak a tanulók arról, hogy mi is az a csomag.

**Fájlkezelés**

Ez a rész az adatok kezelési módját mutatja be: adatfolyamok kezelése, adatok fájlból olvasása, adatok fájlba írása.

**Az egységek használata**

Mindegyik témakörnek a célok és követelmények meghatározásával kell kezdődnie. A fejezetek végére tesztkérdéseket terveztem, melyeknek megoldásával ellenőrizhetik tudásukat a tanulók. Az önellenőrzésre és a tanári ellenőrzésre is van feladatsor.

A tanulónak a fejezetek elsajátítása során törekednie kell arra, hogy a feladatokat a legjobb tudása szerint, önállóan oldja meg. Ha valamilyen problémával találja szemben magát, érdemes azt újragondolásra ösztönözni, alternatív megoldások segítségével bátorítani.

**Értékelés, számonkérés**

A fejezetek értékelésénél a hangsúly az elsajátított elméleti alapok gyakorlati alkalmazásán van. A fejezetek végén található feladatok megoldásai beleszámítanak a végső eredménybe. A tanítás során minden fejezet végén érdemes ellenőrző kérdéseket feltenni. A válaszokból megtudjuk, hogy a tanuló megfelelően elsajátította-e a tananyagot.

# A témakörök tartalmának részletezése

## Alapfogalmak

A számítógépek programozására kialakult nyelveknek három szintjét különböztetjük meg:

– gépi nyelv
– assembly szintű nyelv
– magas szintű nyelv

A tananyagok a magas szintű nyelvek eszközeivel, filozófiájával, használatával foglalkoznak, ezeken belül a Jávával. A magas szintű nyelven megírt programot *forrásprogramnak*, vagy *forrásszövegnek* nevezzük. A forrásszöveg összeállítására vonatkozó formai, „nyelvtani” szabályok összességét *szintaktikai* szabályoknak hívjuk. A tartalmi, értelmezési, jelentésbeli szabályok alkotják a *szemantikai* szabályokat. Egy magas szintű programozási nyelvet szintaktikai és szemantikai szabályainak együttese határoz meg.

Minden processzor rendelkezik saját gépi nyelvvel és csak az adott gépi nyelven írt programokat tudja végrehajtani. A magas szintű nyelven megírt forrásszövegből tehát valamilyen módon gépi nyelvű programokhoz kell eljutni. Erre kétféle technika létezik, a *fordítóprogramos* és az *interpreteres*. A fordítóprogram egy speciális szoftver, amely a magas szintű nyelven megírt forrásprogramból gépi kódú *tárgyprogramot* állít elő. A fordítóprogram a teljes forrásprogramot egyetlen egységként kezeli és működése közben a következő lépéseket hajtja végre:

– lexikális elemzés
– szintaktikai elemzés
– szemantikai elemzés
– kódgenerálás

A lexikális elemzés során a forrásszöveget feldarabolja lexikális egységekre, a szintaktikai elemzés folyamán ellenőrzi, hogy teljesülnek-e az adott nyelv szintaktikai szabályai. Tárgyprogramot csak szintaktikailag helyes forrásprogramból lehet előállítani. A tárgyprogram már gépi nyelvű, de még nem futtatható. Belőle futtatható programot a

*szerkesztő* vagy *kapcsolatszerkesztő* készít. A futtatható programot a *betöltő* helyezi el a tárban, és adja át neki a vezérlést. A futó program működését a *futtató rendszer* felügyeli.

A fordítóprogramok általánosabb értelemben tetszőleges nyelvről tetszőleges nyelvre fordítanak. A magas szintű nyelvek között is létezik olyan, amelyben olyan forrásprogramot lehet írni, amely tartalmaz nem nyelvi elemeket is. Ilyenkor egy *előfordító* segítségével először a forrásprogramból egy adott nyelvű forrásprogramot kell generálni, ami aztán már feldolgozható a nyelv fordítójával.

Az interpreteres technika esetén is megvan az első három lépés, de az interpreter nem készít tárgyprogramot. Utasításonként (vagy egyéb nyelvi egységenként) sorra veszi a forrásprogramot, értelmezi azt, és végrehajtja. Rögtön kapjuk az eredményt, úgy, hogy lefut valamilyen gépi kódú rutin.

Az egyes programnyelvek vagy fordítóprogramosak, vagy interpreteresek, vagy együttesen alkalmazzák mindkét technikát.

Minden programnyelvnek megvan a saját szabványa, amit *hivatkozási nyelvnek* hívunk. Ebben pontosan definiálva vannak a szintaktikai és a szemantikai szabályok. A szintaktika leírásához valamilyen formalizmust alkalmaznak, a szemantikát pedig általában természetes emberi nyelven (pl. angolul vagy magyar) adják meg. A hivatkozási nyelv mellett (néha vele szemben) léteznek az *implementációk*. Ezek egy adott platformon (processzor, operációs rendszer) realizált fordítóprogramok vagy interpreterek. Sok van belőlük. Gyakran ugyanazon platformon is létezik több implementáció. A probléma az, hogy az implementációk egymással és a hivatkozási nyelvvel nem kompatibilisek. A magas szintű programozási nyelvek elmúlt 50 éves történetében napjainkig nem sikerült megoldani a *hordozhatóság* (ha egy adott implementációban megírt programot átviszek egy másik implementációba, akkor az ott fut és ugyanazt az eredményt szolgáltatja) problémáját.

Napjainkban a programok írásához grafikus *integrált fejlesztői környezetek* állnak rendelkezésünkre. Ezek tartalmaznak szövegszerkesztőt, fordítót (esetleg interpretert), kapcsolatszerkesztőt, betöltőt, futtató rendszert és belövőt.

## A forrásszöveg összeállításának általános szabályai

A forrásszöveg, mint minden szöveg, sorokból áll. Kérdés, hogy milyen szerepet játszanak a sorok a programnyelvek szempontjából.

*Kötött formátumú nyelvek*: A korai nyelveknél (FORTRAN, COBOL) alapvető szerepet játszott a sor. Egy sorban egy utasítás helyezkedett el, tehát a sorvége jelezte az utasítás végét.

*Szabad formátumú nyelvek*: Ezeknél a nyelveknél a sornak és az utasításnak semmi kapcsolata nincs egymással. Egy sorba akárhány utasítás írható, egy utasítás akárhány sorban elhelyezhető. A sorban tetszőleges helyen jelenhetnek meg az egyes programelemek. A sorvége nem jelenti az utasítás végét. Éppen ezért ezek a nyelvek bevezetik az utasítás végjelet, ez elég általánosan a pontosvessző. Tehát a forrásszövegben két pontosvessző között áll egy utasítás. A Java is ilyen.

Az eljárásorientált nyelvekben a program szövegében a lexikális egységeket alapszóval, standard azonosítóval, valamilyen elhatároló jellel (zárójel, kettőspont, pontosvessző, vessző, stb.), vagy ezek hiányában egy szóközzel el kell választani egymástól. A fordítóprogram ez alapján ismeri föl azokat a lexikális elemzés során. Az eljárásorientált nyelvekben tehát a szóköz általános elhatároló szerepet játszik. A szóköznek nincs kiemelt szerepe a megjegyzésben és a sztring, valamint karakter literálokban. Itt, mint minden karakter, csak önmagát képviseli. Ahol egy szóköz megengedett elhatárolóként, oda akárhányat is írhatunk. Egyéb elhatárolók mellett is állhat szóköz, ez növeli a forrásszöveg olvashatóságát.

## Karakterkészlet

[2] Minden program forrásszövegének legkisebb alkotórészei a *karakterek*. A forrásszöveg összeállításánál alapvető *a karakterkészlet*, ennek elemei jelenhetnek meg az adott nyelv programjaiban. Az eljárásorientált nyelvek esetén ezek a következők:

- lexikális egységek
- szintaktikai egységek
- utasítások
- programegységek
- fordítási egységek

- program

Minden nyelv definiálja a saját karakterkészletét. A karakterkészletek között lényeges eltérések lehetnek, de a programnyelvek általában a karaktereket a következő módon kategorizálják:

- betűk
- számjegyek
- egyéb karakterek

Minden programnyelvben betű az angol ABC 26 nagybetűje. A nyelvek továbbá gyakran betű kategóriájú karakternek tekintik az _ , $ , # , @ karaktereket is. Ez viszont sokszor implementációfüggő. Abban már eltérnek a nyelvek, hogy hogyan kezelik az angol ABC kisbetűit. A nyelvek túlnyomó többsége a nemzeti nyelvi betűket nem sorolja a betű kategóriába, néhány késői nyelv viszont igen. Tehát ezekben például lehet „magyarul” írni a programot. A számjegyeket illetően egységes a nyelvek szemlélete, mindegyik a decimális számjegyeket tekinti számjegy kategóriájú karakternek.

Az egyéb karakterek közé tartoznak a műveleti jelek (pl. +, -, *, /), elhatároló jelek (pl. [, ], ., :, {, }, ’, ", ;), írásjelek (pl. ?, !) és a speciális karakterek (pl. ~). A program szövegében kitüntetett szerepet játszik a szóköz, mint egyéb karakter. A hivatkozási nyelv és az implementációk karakterkészlete eltérő is lehet. Minden implementáció mögött egy-egy konkrét kódtábla (EBCDIC, ASCII, UNICODE) áll. Ez meghatározza egyrészt azt, hogy egy- vagy több-bájtos karakterek kezelése lehetséges-e, másrészt értelmezi a karakterek sorrendjét.

**Lexikális egységek**

A lexikális egységek a program szövegének azon elemei, melyeket a fordító a lexikális elemzés során felismer és tokenizál (közbenső formára hoz). Fajtái a következők: – többkarakteres szimbólum

- szimbolikus nevek
- címke
- megjegyzés
- literálok

### Többkarakteres szimbólumok

Olyan karaktersorozatok, amelyeknek a nyelv tulajdonít jelentést és ezek csak ilyen értelemben használhatók. Nagyon gyakran a nyelvben operátorok, elhatárolók lehetnek. Például a Jávában többkarakteres szimbólumok a következők: ++, --, &&, /*, */.

### Szimbolikus nevek

*Azonosító*: Olyan karaktersorozat, amely betűvel kezdődik, és betűvel vagy számjeggyel folytatódhat. Arra való, hogy a program írója a saját programozói eszközeit megnevezze vele, és ezután ezzel hivatkozzon rá a program szövegében bárhol. A hivatkozási nyelvek általában nem mondanak semmit a hosszáról, az implementációk viszont értelemszerűen korlátozzák azt.

A következők szabályos Java azonosítók (a Jávában az _ betű kategóriájú):

x
almafa
hallgato_azonosito
SzemelyNev
A következő karaktersorozatok viszont nem azonosítók:
x+y a + nem megengedett karakter 123abc betűvel kell kezdődnie

*Kulcsszó (alapszó, fenntartott szó, védett szó, foglalt szó)*: Olyan karaktersorozat (általában azonosító jellegű felépítéssel), amelynek az adott nyelv tulajdonít jelentést, és ez a jelentés a programozó által nem megváltoztatható. Az utasítások általában egy-egy jellegzetes kulcsszóval kezdődnek, a szakmai szleng az utasítást gyakran ezzel nevezi meg (pl. `if`-utasítás). Minden nyelvre nagyon jellemzőek a kulcsszavai. Ezek gyakran hétköznapi angol szavak, vagy rövidítések. Az alapszavak soha nem használhatók azonosítóként.

A Jávában alapszavak a következők:

| | | | | |
|---|---|---|---|---|
| abstract | else | int | return | void |
| boolean | extends | interface | short | volatile |
| break | false | long | static | while |
| byte | final | native | super | |
| case | finally | new | switch | |
| cast* | float | null | synchornized | |
| catch | for | operator | this | |
| class | generic | outer | throw | |
| const | if | package | throws | |
| continue | implements | private | transient | |
| default | import | protected | true | |
| do | inner | public | try | |
| double | instanceof | rest | var | |

**A Java karakterkészlete**

A nyelv ismeri az UNICODE karakterkészletet, a forráskódban használhatnánk bármilyen nemzetközi karaktert. Azonban mivel nem minden platform ismeri az UNICODE-ot, ajánlatos a -már C-ben is - megszokott ASCII formátumot használni.

*Címke:* olyan azonosító, melynek segítségével végrehajtható utasítások címkézhetők

<címke>:<utasítás> formában. GOTO utasítás nincs, más vezérlési utasítások használják.

*Primitív típusok:* atomi, eljárásorientált értelemben vett típusok

- boolean: logikai típus
- char: karakteres típus
- byte: előjel nélküli egész típus
- short: rövid-egész típus
- int: egész típus
- long: hosszú-egész típus
- float: egyszeres pontosságú lebegőpontos típus
- double: dupla pontosságú lebegőpontos típus

| **Primitív típusok** | **Méret** | **Minimum érték** | **Maximum érték** | **Típus név** |
|---|---|---|---|---|
| **char** | 16-bit | Unicode 0 | Unicode $2^{16}$-1 | Character |
| **byte** | 8-bit | -128 | +127 | Byte |
| **short** | 16-bit | $-2^{15}$ *(-32,768)* | $+2^{15}$-1 *(32,767)* | Short |
| **int** | 32-bit | $-2^{31}$ *(-2,147,483,648)* | $+2^{31}$-1 *(2,147,483,647)* | Integer |
| **long** | 64-bit | $-2^{63}$ *(-9,223,372,036,854,775,808)* | $+2^{63}$-1 *(9,223,372,036,854,775,807)* | Long |
| **float** | 32-bit | 32-bit IEEE 754 lebegőpontos szám | | Float |
| **double** | 64-bit | 64-bit IEEE 754 lebegőpontos számú | | Double |
| **boolean** | 1-bit | `true` vagy `false` | | Boolean |

Ezek a szokásos módon használhatók; deklarációs utasításokban típuskomponens megadásánál alkalmazhatók

## Operátorok

- Unáris postfix és prefix operátorok
  - [ ] tömbképző
  - . minősítő
  - ( ) metódus képző
  - || vagy
  - && és
  - new példányosító
  - (*típus*)*kifejezés* típuskényszerítő
  - +, - előjel
  - ++, -- léptető, pl. i++ vagy ++i mindkettő növeli i értékét, de az első értéke i eredeti, míg a második i megnövelt értéke lesz.

A következő példa a ++ és a -- operátorok alkalmazása és különbségére. A különbség jól látható az egyes esetekben.

```
1:  class Osszead {
2:          public static void main ( String[] arguments){
3:                  System.out.print (”Összeadok két számot”);
4:                  int szam1 = 1;
5:                      int plusz_plusz1 = szam1++;
6:                      int plusz_plusz2 = ++szam1;
7:                      int minusz_minusz = szam1--;
8:                      int mínusz_minusz = --szam1;
9:                  System.out.println (szam1 +” ++ esetén az érték: ” + plusz_plusz1);
10:                 System.out.println (”++”+szam1 +” esetén az érték: ” + plusz_plusz2);
11:                 System.out.println (szam1 +” -- esetén az érték: ” + minusz_minusz1);
12:                 System.out.println (”--”+szam1 +” esetén az érték: ” +
                                        minusz_minusz1);
13:         }
```

14: }

**Ellenőrző feladatok**

1. Helyes-e a kijelentés?

A változó értéke növelhető a ++ operátor használatával a változó név után.

a) igaz
b) hamis

**Válasz:** a.) X++ ez az utasítás egyet ad a változóhoz. Pl.: ha a változó értéke 2 volt, akkor ezt az utasítást kiadva 3 lesz.

2. Melyik operátor az és?

a) &&
b) [ ]
c) ++

**Válasz:** a.) a && operátor.

3. Írj egy program részletett, melyben összefűzés történik!

**Válasz:**

System.out.println(”A nevem:” +nev );

Vagy egyéb olyan megoldás melyben összefűzés szerepel.

## Megjegyzés

[2] A megjegyzés egy olyan programozási eszköz, amely segítségével a programban olyan karaktersorozat helyezhető el, amely nem a fordítónak szól, hanem a program szövegét olvasó embernek. Ez olyan magyarázó szöveg, amely a program használatát segíti, működéséről, megírásának körülményeiről, a felhasznált algoritmusról, az alkalmazott megoldásokról ad információt. A megjegyzést a lexikális elemzés során a fordító ignorálja. A megjegyzésben a karakterkészlet bármely karaktere előfordulhat és minden karakter egyenértékű, csak önmagát képviseli, a karakter-kategóriáknak nincs jelentősége.

*Megjegyzések típusai:*

// karakterszimbólumtól a sor végéig tartó megjegyzések

/* és */ karakterszimbólumok között álló megjegyzések

/** és */ karakterszimbólumok között álló úgynevezett dokumentációs megjegyzések, melyek Java programok dokumentálását szolgálják. A Java fordítók a 90-es évek közepének megfelelő technológiát hordozzák, tehát dokumentálásra is alkalmasak, és ki tudják szűrni a dokumentációs megjegyzéseket, és sokkminden mást, amit korábbi fordítók nem tudtak.

## Adattípusok

Az adatabsztrakció első megjelenési formája az adattípus a programozási nyelvekben. Az adattípus maga egy *absztrakt* programozási eszköz, amely mindig más, *konkrét* programozási eszköz egy *komponenseként* jelenik meg. Az adattípusnak *neve* van, ami egy azonosító. A programozási nyelvek egy része ismeri ezt az eszközt, más része nem. Ennek megfelelően beszélünk *típusos* és *nem típusos* nyelvekről. Az eljárásorientált nyelvek típusosak.

Egy adattípust három dolog határoz meg, ezek:

- tartomány
- műveletek
- reprezentáció

Az adattípusok tartománya azokat az elemeket tartalmazza, amelyeket az adott típusú konkrét programozási eszköz fölvehet értékként.

Az adattípushoz hozzátartoznak azok a műveletek, amelyeket a tartomány elemein végre tudunk hajtani.

Minden adattípus mögött van egy megfelelő belső ábrázolási mód. A reprezentáció az egyes típusok tartományába tartozó értékek tárban való megjelenését határozza meg, tehát azt, hogy az egyes elemek hány bájtra és milyen bitkombinációra képződnek le.

Minden típusos nyelv rendelkezik beépített (standard) típusokkal.

Egyes nyelvek lehetővé teszik azt, hogy a programozó is definiálhasson típusokat. A saját típus definiálási lehetőség az adatabsztrakciónak egy magasabb szintjét jelenti, segítségével a valós világ egyedeinek tulajdonságait jobban tudjuk modellezni.

A saját típus definiálása általában szorosan kötődik az absztrakt adatszerkezetekhez.

**Adatszerkezetek és algoritmusok**

[6] Saját típust úgy tudunk létrehozni, hogy megadjuk a tartományát, a műveleteit és a reprezentációját. Szokásos, hogy saját típust a beépített és a már korábban definiált saját típusok segítségével adjuk meg. Általános, hogy a reprezentáció megadásánál így járunk el.

Csak nagyon kevés nyelvben lehet saját reprezentációt megadni (pl. ilyen az Ada).

Az egyes adattípusok, mint programozási eszközök önállóak, egymástól különböznek. Van azonban egy speciális eset, amikor egy típusból (ez az *alaptípus*) úgy tudunk származtatni egy másik típust, hogy leszűkítjük annak tartományát, változatlanul hagyva műveleteit és reprezentációját. Az alaptípus és az altípus tehát nem különböző típusok.

Az adattípusoknak két nagy csoportjuk van:

A *skalár* vagy *egyszerű* adattípus tartománya atomi értékeket tartalmaz, minden érték egyedi, közvetlenül nyelvi eszközökkel tovább nem bontható. A skalár típusok tartományaiból vett értékek jelenhetnek meg literállként a program szövegében.

A *strukturált* vagy *összetett* adattípusok tartományának elemei maguk is valamilyen típussal rendelkeznek. Az elemek egy-egy értékcsoportot képviselnek, nem atomiak, az értékcsoport elemeihez külön-külön is hozzáférhetünk. Általában valamilyen absztrakt adatszerkezet programnyelvi megfelelői.

**Egyszerű típusok**

Minden nyelvben létezik az *egész* típus, sőt általában egész típusok. Ezek belső ábrázolása fixpontos. Az egyes egész típusok az ábrázoláshoz szükséges bájtok számában térnek el és nyilván ez határozza meg a tartományukat is. Néhány nyelv ismeri az *előjel nélküli egész* típust, ennek belső ábrázolása előjel nélküli (direkt).

Alapvetőek a *valós* típusok, belső ábrázolásuk általában lebegőpontos. A tartomány itt is az alkalmazott ábrázolás függvénye, ez viszont általában implementációfüggő.

Az egész és valós típusokra közös néven mint *numerikus* típusokra hivatkozunk. A numerikus típusok értékein a numerikus és hasonlító műveletek hajthatók végre.

A *karakteres* típus tartományának elemei karakterek, a *karakterlánc* vagy *sztring* típuséi pedig karaktersorozatok.

Egyes nyelvek ismerik a *logikai* típust. Ennek tartománya a hamis és igaz értékekből áll, műveletei a logikai és hasonlító műveletek, belső ábrázolása logikai.

Speciális egyszerű típus a *felsorolásos* típus. A felsorolásos típust saját típusként kell létrehozni. A típus definiálása úgy történik, hogy megadjuk a tartomány elemeit. Ezek *azonosítók* lehetnek. Az elemekre alkalmazhatók a hasonlító műveletek.

Egyes nyelvek értelmezik az egyszerű típusok egy speciális csoportját a *sorszámozott* típust. Ebbe a csoportba tartoznak általában az egész, karakteres, logikai és felsorolásos típusok. A sorszámozott típus tartományának elemei listát alkotnak, azaz van első és utolsó elem, minden elemnek van megelőzője (kivéve az elsőt) és minden elemnek van rákövetkezője (kivéve az utolsót). Tehát az elemek között egyértelmű sorrend értelmezett. A tartomány elemeihez kölcsönösen egyértelműen hozzá vannak rendelve a 0, 1, 2, ... sorszámok. Ez alól kivételt képeznek az egész típusok, ahol a tartomány minden eleméhez önmaga mint sorszám van hozzárendelve.

Egy sorszámozott típus esetén mindig értelmezhetők a következő műveletek:

- ha adott egy érték, meg kell tudni mondani a sorszámát és viszont
- bármely értékhez meg kell tudni mondani a megelőzőjét és a rákövetkezőjét

A sorszámozott típus az egész típus egyfajta általánosításának tekinthető. Egy sorszámozott típus altípusaként lehet származtatni az *intervallum* típust.

**Összetett típusok**

A tömb *statikus* és *homogén* összetett típus, vagyis tartományának elemei olyan értékcsoportok, amelyekben az elemek száma ugyanannyi és az elemek azonos típusúak. A tömböt, mint típust meghatározza:

- dimenzióinak száma
- indexkészletének típusa és tartománya
- elemeinek a típusa

Egyes nyelvek (pl. Java) nem ismerik a többdimenziós tömböket. Ezek a nyelvek a többdimenziós tömböket olyan egydimenziós tömbökként kezelik, amelyek elemei egydimenziós tömbök.

Többdimenziós tömbök reprezentációja lehet sor- vagy oszlopfolytonos. Ez általában implementációfüggő, a sorfolytonos a gyakoribb.

Ha van egy tömb típusú programozási eszközünk, akkor a nevével az összes elemre együtt, mint egy értékcsoportra tudunk hivatkozni, az elemek sorrendjét a reprezentáció határozza meg. Az értékcsoport egyes elemeire a programozási eszköz neve után megadott indexek segítségével hivatkozunk. A Jávában az indexek szögletes zárójeleket használunk.

A tömb típus alapvető szerepet játszik az absztrakt adatszerkezetek folytonos ábrázolását megvalósító implementációknál.

## Változók

A változó olyan programozási eszköz, amelynek négy komponense van:

- név
- attribútumok
- cím
- érték

A név egy azonosító. A program szövegében a változó mindig a nevével jelenik meg, az viszont bármely komponenst jelentheti. Szemlélhetjük úgy a dolgokat, hogy a másik három komponenst a névhez rendeljük hozzá.

Az attribútumok olyan jellemzők, amelyek a változó futás közbeni viselkedését határozzák meg. Az eljárásorientált nyelvekben (általában a típusos nyelvekben) a legfőbb attribútum a típus, amely a változó által felvehető értékek körét határolja be. Változóhoz attribútumok deklaráció segítségével rendelődnek. A deklarációnak különböző fajtáit ismerjük.

*Explicit deklaráció*: A programozó végzi explicit deklarációs utasítás segítségével. A változó teljes nevéhez kell az attribútumokat megadni. A nyelvek általában megengedik, hogy több változónévhez ugyanazokat az attribútumokat rendeljük hozzá.

Az eljárásorientált nyelvek mindegyike ismeri az explicit deklarációt, és egyesek csak azt ismerik. Az utóbbiak általánosságban azt mondják, hogy minden névvel rendelkező programozói eszközt explicit módon deklarálni kell.

A változó címkomponense a tárnak azt a részét határozza meg, ahol a változó értéke elhelyezkedik. A futási idő azon részét, amikor egy változó rendelkezik címkomponenssel, a változó *élettartamának* hívjuk.

*Egy változóhoz cím rendelhető az alábbi módokon.*

*Statikus tárkiosztás*: A futás előtt eldől a változó címe és a futás alatt az nem változik. Amikor a program betöltődik a tárba, a statikus tárkiosztású változók fix tárhelyre kerülnek.

*Dinamikus tárkiosztás*: A cím hozzárendelését a futtató rendszer végzi. A változó akkor kap címkomponenst, amikor aktivizálódik az a programegység, amelynek ő lokális változója, és a

címkomponens megszűnik, ha az adott programegység befejezi a működését. A címkomponens a futás során változhat, sőt vannak olyan időintervallumok, amikor a változónak nincs is címkomponense.

*A programozó által vezérelt tárkiosztás*: A változóhoz a programozó rendel címkomponenst futási időben. A címkomponens változhat, és az is elképzelhető, hogy bizonyos időintervallumokban nincs is címkomponens. Három alapesete van:

– A programozó abszolút címet rendel a változóhoz, konkrétan megadja, hogy hol helyezkedjen el.

– Egy már korábban a tárban elhelyezett programozási eszköz címéhez képest mondja meg, hogy hol legyen a változó elhelyezve, vagyis relatív címet ad meg. Lehet, hogy a programozó az abszolút címet nem is ismeri.

– A programozó csak azt adja meg, hogy mely időpillanattól kezdve legyen az adott változónak címkomponense, az elhelyezést a futtató rendszer végzi. A programozó nem ismeri az abszolút címet.

Mindhárom esetben lennie kell olyan eszköznek, amivel a programozó megszüntetheti a címkomponenst.

A programozási nyelvek általában többféle címhozzárendelést ismernek, az eljárásorientált nyelveknél általános a dinamikus tárkiosztás. A változók címkomponensével kapcsolatos a *többszörös tárhivatkozás* esete. Erről akkor beszélünk, amikor két különböző névvel, esetleg különböző attribútumokkal rendelkező változónak a futási idő egy adott pillanatában azonos a címkomponense és így értelemszerűen az értékkomponense is. Így ha az egyik változó értékét módosítjuk, akkor a másiké is megváltozik.

A változó értékkomponense mindig a címen elhelyezett bitkombinációként jelenik meg. A bitkombináció felépítését a típus által meghatározott reprezentáció dönti el.

Egy változó értékkomponensének meghatározására a következő lehetőségek állnak rendelkezésünkre:

*Értékadó utasítás*: Az eljárásorientált nyelvek leggyakoribb utasítása, az algoritmusok kódolásánál alapvető.

**Jávában:**

változónév = kifejezés;
int szam = 4;

Az értékadó utasítás bal oldalán a változó neve általában a címkomponenst, kifejezésben az értékkomponenst jelenti. Az értékadó utasítás esetén a műveletek elvégzésének sorrendje implementációfüggő. Általában a baloldali változó címkomponense dől el először. A típus-egyenértékűséget valló nyelvekben a kifejezés típusának azonosnak kell lennie a változó típusával, a típuskényszerítést valló nyelveknél pedig mindig a kifejezés típusa konvertálódik a változó típusára.

*Input*: A változó értékkomponensét egy perifériáról kapott adat határozza meg.

*Kezdőértékadás*: Két fajtája van. *Explicit kezdőértékadásnál* a programozó explicit deklarációs utasításban a változó értékkomponensét is megadja. Amikor a változó címkomponenst kap, akkor egyben az értéket reprezentáló bitsorozat is beállítódik Megadható az értékkomponens literál vagy kifejezés segítségével.

A hivatkozási nyelvek egy része azt mondja, hogy mindaddig, amíg a programozó valamilyen módon nem határozza meg egy változó értékkomponensét, az *határozatlan*, tehát nem használható föl. Ez azért van, mert amikor egy változó címkomponenst kap, akkor az adott memóriaterületen tetszőleges bitkombináció állhat, amivel nem lehet mit kezdeni.

Van olyan hivatkozási nyelv, amely az *automatikus kezdőértékadás* elvét vallja. Ezeknél a nyelveknél, ha a programozó nem adott explicit kezdőértéket, akkor a címkomponens hozzárendelésekor a hivatkozási nyelv által meghatározott bitkombináció kerül beállításra. A hivatkozási nyelvek harmadik része nem mond semmit erről a dologról. Viszont az implementációk túlnyomó része megvalósítja az automatikus kezdőértékadást, akár még a hivatkozási nyelv ellenében is.

Változó: hagyományos eljárásorientált változó, melynek van neve, attribútuma, címe, értéke. Változó deklarálható primitív típusokkal és referenciatípusokkal is. Az eljárásorientált változó

mindig a programegység lokális változója, fogalmilag teljesen eljárásorientált. A lokális változó deklarálandó, ez a deklaráció C-szerü: típusnév és változónév ill. Változónevek megadásával; lehet explicit kezdőértéket megadni, lokális változóknak automatikus kezdőértékük nincs.

**Egész számok összeadás**

A következő program összead egy előre, már változóban deklarált két számot.

```
1:  class Osszead {
2:          public static void main ( String[] arguments){
3:                  System.out.print (”Összeadok két számot”);
4:                  int szam1 = 10;
5:                  int szam2 = 5;
6:                  int osszeg = 0;
7:                  osszeg = szam1 + szam2;
8:                  System.out.print (”10+5=” + osszeg);
9:          }
10: }
```

Mi is történt most itt? A 4. és 5. sornál létrehozunk egy-egy int típusú változót melynek azonnal adunk értéket is. A 6. sornál deklaráljuk az „osszeg” nevű változót, mely szintén int típusú lesz, és kezdő értéknek nullát adunk meg. A 7. sorban történik az összeadás. A 8 sorban a kiíratás. A (+) jel segítségével hozzáfűzés történik. Az összeadott két szám értékét melyet az „osszeg” nevű változóban tárolunk, írjuk képernyőre.

A következő példa kiírja a változók értékét és az összeget is.

```
1:  class Osszead {
2:          public static void main ( String[] arguments){
3:                  System.out.print (”Összeadok két számot”);
4:                  int szam1 = 10;
5:                  int szam2 = 5;
6:                  int osszeg = 0;
7:                  osszeg = szam1 + szam2;
```

```
8:              System.out.print (”összeadásunk eredménye: ” + szam1 + ” + ” +
                                  szam2 + ” = ” + osszeg);
9:          }
10: }
```

Ennek a példának a segítségével az összefűzést és a tartalom kiíratást lehet gyakoroltatni.

**Ellenőrző feladatok**

1. Milyen karakterrel nem kezdődhet változó név? Több válasz is lehetséges.

    a.) $
    b.) //
    c.) /*
    d.) Betűvel

**Válasz:** b.) c.) dollárjellel ($), aláhúzással (_) vagy betűvel kezdődhet. Ha két perjellel (//) vagy perjellel és csillaggal (/*) nem.

2. Igaz-e az állítás?

A boolean típusú változó true és false értékeket tárol.

    a) igaz
    b) hamis

**Válasz**: a.) A boolean csak igaz (true) vagy hamis (false) értékeket tárol.

3. Helyes-e a felírás?

```
int a = 3;
```

    a) igen
    b) nem

**Válasz**: a.) Helyes, mert így definiálunk egy változót és ezzel értéket is adunk neki.

4. Milyen értéket tárolhatunk String típusú változóban?

   b) karakterláncokat
   c) numerikus értékeket
   d) logikai értékeket

**Válasz:** a.) String – ben csak karakterláncokat tárolhatunk.

5. Sorold fel a változó komponenseit!

**Válasz:** név, attribútumok, cím, érték

## Vezérlő utasítások a Jávában

A Jávában három vezérlő utasítás van még az eddigiekben tárgyalt végrehajtható utasításokon kívül:

`continue`; Ciklus magjában alkalmazható. A ciklus magjának hátralévő utasításait nem hajtja végre, hanem az ismétlődés feltételeit vizsgálja meg és vagy újabb ciklusléptésbe kezd, vagy befejezi a ciklust.

`break`; Ciklus magjában, vagy többszörös elágaztató utasítás `case`-ágában helyezhető el. A ciklust szabályosan befejezteti, illetőleg kilép a többszörös elágaztató utasításból.

`return` [ kifejezés]; Szabályosan befejezteti a függvényt és visszaadja a vezérlést a hívónak

## Elágaztató utasítások

### Kétirányú elágaztató utasítás (feltételes utasítás)

[1] A kétirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján két tevékenység közül válasszunk, illetve egy adott utasítást végrehajtsunk vagy sem.

A feltételes utasítás szerkezete:

`if` (feltétel) utasítás [`else` utasítás]

A feltétel egy logikai (vagy annak megfelelő típusú) kifejezés.

Kérdés, hogy egy nyelv mit mond az utasítások megadásáról. Egyes nyelvek (pl. Pascal) szerint itt csak egyetlen végrehajtható utasítás állhat. Ha viszont az utasítás olyan összetett, hogy csak több utasítással tudjuk leírni, akkor *utasítás zárójeleket* kell alkalmazni. Az utasítás zárójel a Jáva esetén { }. Egy ilyen módon bezárójelezett utasítássorozatot hívunk *utasítás csoportnak*. Az utasítás csoport formálisan egyetlen utasításnak tekintendő. A kétirányú elágaztató utasításnál beszélünk rövid (nem szerepel az `else`-ág) és hosszú (szerepel az `else`-ág) alakról.

A kétirányú elágaztató utasítás szemantikája a következő:

Kiértékelődik a feltétel. Ha az igaz, akkor végrehajtódik az utasítás, és a program az `if`-utasítást követő utasításon folytatódik. Ha a feltétel értéke hamis, és van `else`-ág, akkor az ott megadott utasítás hajtódik végre, majd a program az `if`-utasítást követő utasításon folytatódik, végül ha nincs `else`-ág, akkor ez egy üres utasítás.

Az `if`-utasítások tetszőlegesen egymásba ágyazhatók és természetesen akár az `if`-ágban, akár az `else`-ágban újabb feltételes utasítás állhat. Ilyenkor felmerülhet a „csellengő `else`” problémája, amikoris a kérdés a következő: egy

`if` ... (feltétel) `if` ... (feltétel) ... `else` ...

konstrukciójú feltételes utasítás esetén melyik `if`-hez tartozik az `else` - ág? Vagyis itt egy olyan rövid `if` - utasítás áll, amelybe be van ágyazva egy hosszú, vagy pedig egy hosszú IF – utasítás ágában szerepel egy rövid?

A válasz többféle lehet:

a. A „csellengő `else`” probléma kiküszöbölhető, ha mindig hosszú `if`-utasítást írunk fel, úgy, hogy abban az ágban, amelyet elhagytunk volna, üres utasítás szerepel.

b. A nyelv szintaktikája olyan, hogy egyértelmű a skatulyázás.

A Jávában a feltétel zárójelek között szerepel és nincs `then` alapszó.

A következő példában a program bekér egy karaktert, majd kiírja és a hozzá tartozó ASCII kódtáblában szereplő decimális kódot. A bekért karakter típusa lehet vezérlő karakter, számjegy, angol ábécé nagybetűje és kisbetűje.

```
public class Kodtabla {
public static String tipus(char karakter) {
 String s;
 if(karakter<(char)32) s="vezerlo karakter";
 else if('0'<=karakter && karakter<='9') s="szamjegy";
 else if('A'<=karakter && karakter<='Z') s="angol nagybetu";
 else if('a'<=karakter && karakter<='z') s="angol kisbetu";
 else if(karakter>(char)127) s="valoszinuleg ekezetes betu";
```

```
 else s="egyeb karakter";
 return s;
 }
public static void main(String[] args) {
 if(args.length==0) {
  System.out.println("keves parameter!");
  System.exit(1);
  }
 char c=args[0].charAt(0);
 System.out.println("A beirt karakter: "+c);
 System.out.println("A beirt karakter kodja: "+(int)c);
 System.out.println("A beirt karakter tipusa: "+tipus(c));
 }
}
```

**Többirányú elágaztató utasítás**

A többirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján egymást kölcsönösen kizáró akárhány utasítás közül egyet (vagy esetleg egyet sem) végrehajtsunk. Az utasítások közötti választást egy kifejezés értékei szerint tehetjük meg. Szintaktikája és szemantikája nyelvenként különböző.

**Java**

`switch` (kifejezés) **{**
`case` egész_konstans_kifejezés : [ utasítás ]
[ `case` egész_konstans_kifejezés : [ utasítás ]]…
[ `default`: utasítás]
**}**;

A kifejezés típusának egészre konvertálhatónak kell lennie. A `case`-ágak értékei nem lehetnek azonosak. Az utasításnak végrehajthatóknak kell lennie, vagy blokk lehet. A `default`-ág akárhol elhelyezkedhet.

Működése:

Kiértékelődik a kifejezés, majd értéke a felírás sorrendjében hasonlításra kerül a `case`-ágak értékeivel. Ha van egyezés, akkor végrehajtódik az adott ágban megadott utasítás, majd a program a következő ágakban megadott utasításokat is végrehajtja, ha még van ág. Ha nincs egyezés, de van `default`-ág, akkor az ott megadott utasítás hajtódik végre, majd a program a következő ágakban megadott utasításokat is végrehajtja, ha még van ág. Ha nincs `default`-ág, akkor ez egy üres utasítás. Tehát a Jávában külön utasítás kell ahhoz, hogy kilépjünk a `switch`-utasításból, ha egy ág tevékenységét végrehajtottuk (`break`-utasítás).

A következő példaprogram bekér egy osztályzatot és kiírja azt betűkkel.

```
public class osztalyzat {
 public static void main(String[] args) {
  if(args.length<1) {
   System.out.println("Kevés parameter!");
   System.exit(1);
   }
  String s;
  int j=Integer.parseInt(args[0]);
  switch(j) {
   case 1:s="elegtelen";break;
   case 2:s="elegseges";break;
   case 3:s="kozepes";break;
   case 4:s="jo";break;
   case 5:s="jeles";break;
   default:s="???";
   }
  System.out.println("A megadott jegy: "+j+" ("+s+")");
  }
 }
```

**Ellenőrző feladatok**

1. Mit fog kiírni a következő program?

```
1:  class valami {
2:          public static void main (String [] args ){
3:                  int benzin = 1;
4:                  if (benzin = = 0) {
5:                          System.out.println (”üres tank”);
6:                  }
7:                  else{
8:                          System.out.println (”tele a tank”);
9:                  }
10:         }
11: }
```

**Válasz:** Tele a tank felirat jelenik meg.

2. Mit csinál a `break` utasítás?

   a) sort tör
   b) kilépés
   c) semmit

**Válasz:** b.) Kilép a többszörös elágaztató utasításból

3. Hol helyezhető el `switch` blokkban a `default`-ág?

**Válasz:** a.) A `default`-ág akárhol elhelyezkedhet.

## Műveletek ismétlése

### Ciklusszervező utasítások

A ciklusszervező utasítások lehetővé teszik, hogy a program egy adott pontján egy bizonyos tevékenységet akárhányszor megismételjünk.

Egy ciklus általános felépítése a következő:

- fej
- mag
- vég

Az ismétlésre vonatkozó információk vagy a fejben vagy a végben szerepelnek.

A mag az ismétlendő végrehajtható utasításokat tartalmazza.

A ciklusok működésénél megkülönböztetünk két szélsőséges esetet. Az egyik az, amikor a mag egyszer sem fut le, ezt hívjuk *üres ciklusnak*. A másik az, amikor az ismétlődés soha nem áll le, ez a *végtelen ciklus*. A működés szerinti végtelen ciklus a programban nyilván szemantikai hibát jelent, hiszen az sohasem fejeződik be.

A programozási nyelvekben a következő ciklusfajtákat különböztetjük meg: *feltételes, előírt lépésszámú (Jávában ilyen nincs), felsorolásos*, *végtelen* és *összetett* ciklus.

### Feltételes ciklus

Ennél a ciklusnál az ismétlődést egy feltétel igaz vagy hamis értéke szabályozza. A feltétel maga vagy a fejben vagy a végben szerepel. Szemantikájuk alapján beszélünk kezdőfeltételes és végfeltételes ciklusról.

*Kezdőfeltételes ciklus:*
A feltétel a fejben jelenik meg.

Működése:
Kiértékelődik a feltétel. Ha igaz, végrehajtódik a ciklusmag, majd újra kiértékelődik a feltétel, és a ciklusmag mindaddig újra és újra lefut, amíg a feltétel hamissá nem válik. Tehát, ha

egyszer beléptünk a magba, akkor ott kell valamikor olyan utasításnak végrehajtódnia, amely megváltoztatja a feltétel értékét.

A kezdőfeltételes ciklus lehet üres ciklus, ha a feltétel a legelső esetben hamis. Lehet végtelen ciklus is, ha a feltétel a legelső esetben igaz és mindig igaz is marad.

Kezdőfeltételes ciklus szerkezete:

`while` (feltétel) végrehajtható_utasítás;

Példa kezdőfeltételes ciklusra:
A példaprogram kiírja az első 10 páros számot.

```
public class Paros{
public static void main(String[] args) {
  int i=1,j=2,n=10;
  System.out.println("Az elso "+n+" paros szam kiirasa:");
  while(i<=n) {
   System.out.println(i+". paros szam: "+j);
   j=j+2;
   i=i+1;
   }
  }
 }
```

*Végfeltételes ciklus:*

A feltétel általában a végben van, de vannak nyelvek, amelyekben a fej tartalmazza azt.

Működése:
Először végrehajtódik a mag, majd ezután kiértékelődik a feltétel. Általában, ha a feltétel hamis, újra végrehajtódik a mag. Tehát az ismétlődés mindaddig tart, míg a feltétel igazzá nem válik. Vannak viszont olyan nyelvek, amelyek igazra ismételnek. Nyilván itt is a magban kell gondoskodni a feltétel értékének megváltoztatásáról.

A végfeltételes ciklus soha nem lehet üres ciklus, a mag egyszer mindenféleképpen lefut. Végtelen ciklus viszont lehet, ha a feltétel értéke a második ismétlés után nem változik meg.

Végfeltételes ciklus szerkezete:

`do` végrehajtható_utasítás `while` (feltétel);

Példa végfeltételes ciklusra:

A példaprogram kiírja az első 10 hárommal osztható számot.

```
public class Harommal {
public static void main(String[] args) {
  int i=1,j=3,n=10;
  System.out.println("Az elso "+n+" harommal oszthato szam kiirasa:");
  do {
   System.out.println(i+". szam: "+j);
   j+=3;
   i+=1;
   } while(i<=n);
  }
 }
```

### `for` ciklus

`for`-ciklus szerkezete:

`for`([kifejezés1]; [kifejezés2]; [kifejezés3]) végrehajtható_utasítás;

Ez a ciklus megfelel a következő kódnak:

kifejezés1;

`while`(kifejezés2) {végrehajtható_utasítás; kifejezés3;}

A kifejezés1 inicializáló kifejezés, kifejezés-utasításként a ciklus működése előtt értékelődik ki. A kifejezés2 felelős a ciklus befejezéséért, feltételként értelmeződik A kifejezés3 kifejezés-utasításként minden ciklusléps végén kiértékelődik. Ha a kifejezés2 nem szerepel, akkor végtelen ciklus lesz. A végtelen ciklusból szabályosan a `break` utasítással tudunk kilépni.

Példa `for`-ciklusra:

A példaprogram kiírja kettő első 10 hatványát.

```
public class Kettohatvanya {
public static void main(String[] args) {
  int i,j=2,n=10;
  System.out.println("2 elso "+n+" hatvanyanak kiirasa:");
  for(i=1;i<=n;i++) {
   System.out.println("2^"+i+" = "+j);
   j*=2;
   }
  }
 }
```

**Végtelen ciklus**

A végtelen ciklus az a ciklusfajta, ahol sem a fejben, sem a végben nincs információ az ismétlődésre vonatkozóan. Működését tekintve definíció szerint végtelen ciklus, üres ciklus nem lehet. Használatánál a magban kell olyan utasítást alkalmazni, amelyik befejezteti a ciklust. Nagyon hatékony lehet eseményvezérelt alkalmazások implementálásánál.

**Összetett ciklus**

Az előző ciklusfajták kombinációiból áll össze. A ciklusfejben tetszőlegesen sok ismétlődésre vonatkozó információ sorolható fel, szemantikájuk pedig szuperponálódik. Nagyon bonyolult működésű ciklusok építhetők fel a segítségével.

A nyelvek egy részében vannak olyan vezérlésátadó utasítások, amelyeket bármilyen fajta ciklus magjában kiadva, a ciklus szabályos befejezését eredményezik. Végtelen ciklus szabályosan csak így fejeztethető be.

Minden ciklusfajta kombinálható az összes többivel, így jönnek létre az összetett ciklusok.

**Ellenőrző feladatok**

1. Mivel választhatjuk el a `for`-ciklus fejében szereplő kifejezéseket?

    a) vesszővel
    b) pontosvesszővel
    c) ponttal

**Válasz:** b.) Pontosvesszővel.

2. Mi a különbség a `break` és a `continue` között?

**Válasz:** `continue`; Ciklus magjában alkalmazható. A ciklus magjának hátralévő utasításait nem hajtja végre, hanem az ismétlődés feltételeit vizsgálja meg és vagy újabb ciklusléptésbe kezd, vagy befejezi a ciklust.

`break`; Ciklus magjában, vagy többszörös elágaztató utasítás `case`-ágában helyezhető el. A ciklust szabályosan befejezteti, illetőleg kilép a többszörös elágaztató utasításból..

3. Mi a különbség a `do-while` és a `while` ciklus között?

**Válasz:** A `while` ciklus esetében a feltétel vizsgálata van előbb, aztán az utasítás részé. Előfordulhat, ha a feltétel nem teljesül, akkor az utasítás nem hajtódik végre egyszer sem. A `do-while` ciklus esetében az utasítás blokkban szereplő utasítás vagy utasítások legalább egyszer végrehajthatóak.

# Tömbök

## Információk tárolása tömbökben

A számítógép feltalálásának legnagyobb nyertese a Mikulás. Őt az emberiség évszázadokon át óriási mértékű információ összegyűjtésére és feldolgozására kényszeríttette.

- Rossz gyerek
- Jó gyerek
- A kért ajándékok
- Szörnyű kéményekkel rendelkező otthonok”[9]

„Az Északi-sarkon nagyszerű társ a számítógép. Nagy segítség az információk tárolásában, osztályozásában és tanulmányozásában.”[9]

Az adatok tárolása a számítógépen többféle módon történhet, a legegyszerűbb mód, ha egy változóba tesszük őket. Az adatok tárolásának ez a módja korlátozott felhasználást tesz lehetővé.

„Ha a Mikulásnak minden gyerekhez egy változónevet kellene használnia, akkor legalább 12 karácsonyon át lenne kénytelen dolgozni a programon, nem beszélve arról, milyen hatással lenne ez a kedélyére. A rossz gyerekek listája a hasonló információk összegyűjtésének egy példája. Minden gyerek neve egy karakterlánc vagy a Mikulás Információs Rendszer egy azonosítószáma. Az ilyen listák tárolására *tömböket* használhatunk.”[9]

A tömbök azonos típusú információk (változók) tárolására alkalmasak. Minden információ tárolására használhatunk többöket is. A tömb összetett információk tárolására is alkalmas.

## Tömbök létrehozása

A változókhoz hasonlóan a tömböket is hasonlóan deklaráljuk, a tömb nevének megadásával és típusával hozzuk létre. A különbség a szögletes zárójelek, a [ ] használatában van.

Bármilyen változó tárolható tömbben. Karakterlánc `String[]` vagy egész számok `int[]`.

int[] szam;

A Java rugalmasan kezeli a szögletes zárójeleket egy tömb deklarálásánál. A zárójel a változó típusa helyett a változó neve mögé is helyezhető.

int szam[];

A Java mind a két megoldást engedi, de a könnyebb átláthatóság céljából célszerű ragaszkodni egy fajtához.

A tömbök kezdőérték adásához a `new` utasítást használhatjuk a változó típusával együtt, vagy adhatunk a tömbnek értéket a { } jelek közt. A tömb méretét is meg kell adnunk. A tömbökben tárolt adatokat a tömb elemeinek nevezzük.

int[] szam = new int[10];

Az előbbi kódrészlet egész számokat tárol egy `szam` nevű és `int` típusú tömbben. A tömb tíz elemet tartalmaz.

Ha a `new` utasítással segítségével hozunk létre egy tömböt, akkor meg kell adnunk az elemek számát. Az így létrehozott tömb minden eleme rendelkezik kezdőértékkel. A `numerikus tömb` esetében ez az érték 0, a `char` tömböknél '\0', `boolean` tömböknél pedig hamis `false`. A string tömböknél és minden más objektumnál ez null értékkel jön létre.

A tömb elemeit megadhatunk kapcsos zárójelek között.

```
String[] honapok = { "Január", "Február", "Március", "Április", "Május", "Junius", "Július",
                     "Augusztus", "Szeptember", "Október", "November", "December" };
```

A tömb elemeinek számát nem kell megadni, mert felsoroltuk azokat egy listában. A tömb listájában szereplő elemek egyforma típusúaknak kell lenniük. Az előbbi példában a hónapok vannak felsorolva melyeknek karakterlánc `String` típusuk van.

Ha a tömb létrejött, akkor több elemet már nem tudunk hozzáadni.

**A tömbök használata**

A tömböket ugyanúgy használjuk a programozás során, mint bármely más változót. A különbség, hogy a tömb neve után szögletes zárójeleket használunk és az elemek számában

van. Az elemekre egész számok `indexek` segítségével hivatkozunk. Java-ban az elemek indexe nullával kezdődik. Ez azt jelenti, hogy a legmagasabb sorszám eggyel kisebb, mint amire valójában számítanánk. Ha az elemre index segítségével hivatkozunk onnantól kezdve, mint egy változót használhatjuk.

Példa az elemek indexelésére:

```
String[] honap = new String[12];
```

Ez az utasítás egy karakterlánc `String` típusú tömböt hoz létre, amelyek nullától tizenegyig vannak indexelve.

Ha egy program futása közben ellenőrizni szeretnénk a tömb felső indexét, hogy elkerüljük annak túllépését (túlcsordulás), a `length` változót használhatjuk. Minden létrehozott tömbhöz hozzá van rendelve. A `length` egy egész típusú változó, a tömb lehetséges elemeinek számat tartalmazza.

A következő példában létrehozunk egy tömböt, és kiíratjuk az elemeinek a számát. A hónapokat tartalmazó program részletett egészítjük ki.

```
String[] honapok = { "Január", "Február", "Március", "Április", "Május", "Junius", "Július",
                     "Augusztus", "Szeptember", "Október", "November", "December" };

System.out.println(”A hónapokat tartalmazó tömb hossza: ” + honapok.length);
```

A példában a `honapok.length` értéke tizenkettő, ami azt jelenti, hogy a legmagasabb meghatározható indexű elem a tizenegyes.

A Jávában kétféle mód van a szövegkezelésre, kezelhetjük őket karakterláncként `String` vagy karakterekből `Char` álló tömbként. Ha karakterláncokkal dolgozunk, hasznosa karakterlánc minden karakterét egy tömb egy elemébe helyezni. A megvalósításához a `tocharArray()` tagfüggvényt kell meghívnunk, amely a karakterlánc hosszával megegyező elemszámú `char` típusú tömböt hoz létre.

A Java tagfüggvények segítségével old meg nagyon sok problémát. Sajnos magyar nyelvű dokumentáció nem áll rendelkezésünkre, ezért ajánlott az angolul nyelv alapszíntű tudása. A

Java dokumentáció elérhető az interneten a következő oldalon:
http://java.sun.com/javase/6/docs/api/

A következő alkalmazás egy karakterláncot ír ki úgy, hogy minden szóközt (' ') pontra (' . ') cserél ki.

```
1:  class Nincsszokoz {
2:          public static void main(String[] arguments) {
3:                  String szoveg = "Mindenki szereti a nyarat, de nekem a tavasz is nagyon
                                    tetszik!";
4:                  char[] karakter = szoveg.toCharArray();
5:                  for (int i = 0; i < karakter.length; i++) {
6:                          char mostani = karakter[i];
7:                          if (mostani != ' ') {
8:                                  System.out.print(mostani);
9:                          } else {
10:                                 System.out.print('.');
11:                         }
12:                 }
13:         }
14: }
```

A program kimenete a következő lesz:

Mindenki.szereti.a.nyarat, .de. nekem.a.tavasz.is.nagyon.tetszik!

A `Nincsszokoz` alkalmazás két helyen tárolja a „Mindenki szereti a nyarat, de nekem a tavasz is nagyon tetszik!” szöveget a `szoveg` nevű karakterláncban (`String`) és a „karakter” nevű karaktertömbben. A tömb létrehozása a 4. sorban, a `szoveg.toCharArray()` tagfüggvény meghívásával történik, ami feltölti a tömb elemeit a szöveg karaktereivel. A „M” karakter a nulladik indexű elem, az „i” az első indexű elem, és így tovább, egészen a az utolsó elemig.

A `for` ciklus végignézi a `karakter` tömb minden elemét. Ha a karakter nem szóköz, akkor megjeleníti, ha pedig szóköz, akkor helyette egy pontot ír ki.

**Többdimenziós tömbök**

Többdimenziós tömb nincs a Jávában.

**Tömbök rendezése**

A hasonló adatokat egy csoportba gyűjtjük össze. Ha ez egy tömbben történik, akkor az egyik legegyszerűbb dolog, ha ezek átrendezzük. A következő utasítások megcserélik egy `szam` nevű, egész típusú tömb két elemének értékeit.

```
int atmenet = szam[4];
szam[4] = szam[5];
szam[4] = atmenet;
```

Az utasítások felcserélik a `szam[ 5 ]` és a `szam[ 6 ]` érteket. A `atmenet` nevű, egész típusú változó átmeneti taroló. Az elemek átrendezésének egyik leggyakoribb oka, hogy a tömbben az elemeket egy adott sorrendbe szeretnénk tenni.

A *rendezés* olyan eljárás, amely során egy bizonyos szempont szerint összefüggő adatokat tartalmazó listát átrendezzük.

A Jávában a rendezés egyszerű, mert az `Arrays` osztály elvégzi helyettünk a munkát. Az `Arrays`, amely a `java.uti1` osztálycsoport része, minden típusú változót, karakterláncot és karaktert képes rendezni. Ha az `Arrays` osztályt akarjuk használni, akkor importálni kell ezt az osztályt `import java.util.*;` (minden a java `util` csomagjában szereplő osztályt importálunk) vagy `import java.util.Arrays`; (csak az `Arrays` osztályt importáljuk).

A változótömböket az `Arrays` osztály emelkedő számsorrendbe rendezi. A karaktereket tartalmazó tömböket pedig (ábécé) sorrendbe rendezi.

A következő példában a hónapokat tartalmazó tömböt használjuk újra, és ezt fogjuk rendezni. Oda kell figyelni, hogy az `Arrays` osztályt importálnunk kell.

```
1:  import java.util.*;
2:  class Rendezes {
3:              public static void main(String[] arguments) {
```

```
4:          String[] honapok = { "Január", "Február", "Március", "Április",
                                 "Május", "Junius", "Július", "Augusztus",
                                 "Szeptember", "Október", "November",
                                 "December" };
5:          System.out.println("Az eredeti tömb tartalma: ");
6:          for (int i = 0; i < honapok.length; i++) {
7:              System. out. print ln(i + ": "+ honapok[i]);
8:          }
9:          Arrays.sort(honapok);
10:         System.out.println("Rendezett tömb tartalma:");
11:         for (int i = 0; i < honapok.length; i++) {
12:             System. out. print ln(i +": " + honapok[i]);
13:         }
14:     }
15: }
```

A futási eredmény a következő lesz:

| Az eredeti tömb tartalma: | Rendezett tömb tartalma: |
|---|---|
| Január | Augusztus |
| Február | Április |
| Március | December |
| Április | Február |
| Május | Január |
| Június | Július |
| Július | Június |
| Augusztus | Május |
| Szeptember | Március |
| Október | November |
| November | Október |
| December | Szeptember |

„A számítógépprogramokhoz számos rendezési módszert fejlesztettek ki. Az `Arrays` osztály által használtat összehangolt gyorsrendezésnek nevezik, és Jon L. Bentley és M. Douglas Mcllroy írta le 1993-as kiadású, *Software-Practice and Experience* című könyvében. Az egyes módszereknek megvan a saját nevük: van például *kupacrendezés, farendezés* vagy *buborékos rendezés*. Jason Harrisonnek van egy weboldala, amely a Java nyelvet használja az egyes rendezési módszerek sebességének vizuális demonstrálására. Az oldal megtekinthető a következő címen: *http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html*”[9]

**Ellenőrző feladatok**

1. Beállíthatjuk a `length` változó értékét, hogy így csökkentsük vagy növeljük a tömb hosszát?

**Válasz**: A tömb létrehozása után annak hosszát semmilyen módon nem lehet megváltoztatni; a `length` változót kizárólag a tömb felső határának lekérdezésére használhatjuk.

2. Melyik változót használjuk a tömbök felső határának ellenőrzésére?

   a) top
   b) length
   c) max

**Válasz:** A tömb elemeinek a számát a `length` segítségével határozzuk meg.

3. Hozzál létre egy numerikus adatokat tartalmazó tömböt!

**Válasz**: int[] tomb;

4. Minek a segítségével azonosíthatjuk, a tömb egy elemét?

   a) magával az elemmel
   b) a tömb nevével
   c) indexszel

**Válasz:** c.) A tömb elemeit indexek segítségével azonosíthatjuk. Jávában a tömb indexelése nullával kezdődik

## Az Objektumorientált nyelvek

[1] Az OO nyelveknek két nagy csoportjuk van: a hibrid és a tisztán OO nyelvek.

A hibrid nyelvek azok a nyelvek, amelyek egy más paradigma mentén épülnek fel, és emellé eszközként épül a nyelvbe egy OO-rendszer. Ha kiemeljük a programból ezt az OO eszközrendszert, a másik paradigmához tartozó nyelv eszközei megmaradnak.

A hibrid nyelvekben nincs nyelvi szinten beépített osztályhierarchia, hanem az implementációk adnak hierarchiákat, illetve a programozó ezekben a nyelvekben saját hierarchiát hozhat létre. (pl.: C++)

A hibrid nyelvek előnye: a programozó kétféle paradigma mentén programozik, kihasználja mindkettő előnyeit. Hátrányuk: a kétféle paradigma hátrányai összeadódnak. Tisztán OO-nyelvek azok a nyelvek, melyek az OO paradigma mentén épülnek fel. Ha kiemeljük a programból az OO eszközrendszert, akkor nem marad semmi. A nyelven túlmenően elválaszthatatlanul tartalmaznak egy fejlesztői környezetet is. A nyelvi rendszer egy teljes osztályhierarchiát alkot.

Program lényege: saját osztályok definiálása, beillesztése, osztályhierarchia, örököltetés, példányosítás (ettől kezdve a példányok olyanok, mint az osztályok). Példányosítás után az objektumok viselkedésmódjuknak megfelelően tevékenykednek.

Alapkérdése az egységesség. Azon nyelvek tisztán OO-ak, melyek az egységességet vallják, azaz csak objektumok vannak benne.

Majdnem tiszta OO nyelvként jön létre, de vannak nem OO elemei is. A Java tekinthető javított C++-nak, amelyből bizonyos dolgokat elhagytak és bizonyos új eszközöket beépítenek.

A Java nyelv létrehozásakor alapvető célkitűzés a biztonságos programozás lehetőségének megteremtése. A rendszer nem engedi, hogy a programozó azt tegyen, amit akar. Ezzel lehetővé teszi az eseményvezérelt programozást. Az ötletgazda a *Sun* volt.

A Java egy hordozhatónak szánt fordítóprogramos nyelv. A fordítóprogram nem gépi kódba fordít, hanem egy közbenső, hordozható ún. bájtkódot hoz létre. Ezt a bájtkódot a `Java`

`Virtual Machine` (továbbiakban JVM) interpretálja. A JVM egy absztrakt számítógépnek tekinthető, amelyet megvalósítanak minden platformon, biztosítva ezzel a platformfüggetlenséget.

A JVM

- célhardver, melynek gépi kódja a bájtkód;
- a bájtkódot az adott platform gépikódjává alakítja;
- fordítóként funkcionál az adott hardveren;

A bájtkód interpreteres futtatására, vagy a teljes bájtkód fordítására szolgál.
A Java problémái:

- bár hordozhatónak szánták, a különböző platformokra implementált JVM-ek nem teljesen kompatíbilisek egymással.

## Objektumok – öröklődés

### Általános fogalmak (az OO paradigma filozófiai szintű jellemzése)

*Osztály:* Ez a paradigma legalapvetőbb fogalma (az imperatív nyelvek elnevezése nem a legszerencsésebb). Az eljárás-orientált nyelvekben használt adattípus fogalom megfelelője; olyan az eszköz, mely egyesíti magában azt az eszközrendszert, amelyben együtt kezeljük az adat- és funkcionális modellt. A valós világ magasabb absztrakciós szinten való ábrázolást teszi lehetővé.

Egy osztálynak vannak attribútumai és metódusai.

*Attribútumok:* segítségükkel kezelhető az adatmodell, és tetszőleges bonyolultságú absztrakt adatszerkezet ábrázolható. Az attribútumokat az osztály létrehozója határozza meg.

*Módszerek, metódusok:* megadják a funkcionális modellt, az objektumosztály viselkedésmódját definiálják. Fogalmilag megfelelnek az eljárás-orientált nyelvek alapprogramjainak. A metódusok valójában eljárások, melyekben az objektumosztály viselkedésének algoritmusát írjuk le.

*Bezárás:* az OO paradigma általánosítja a bezárás fogalmát. Az osztályokhoz tartozó olyan eszközrendszer, amelynek segítségével egy elem (attribútum, metódus) láthatóságát szabályozhatjuk. OO rendszerek többszintű (általában minimum háromszintű) bezárást

valósítanak meg. Bezárás csak osztályszinten van, az osztály elemeire vonatkozik, és az egyes elemek láthatósága külön-külön beállítható.

*Objektum:* egy adott osztályból származtatva, példányosítással létrejövő konkrét eszköz. (eljárás-orientált nyelvek változó fogalmának megfelelője).

Osztályon belül az attribútumok és módszerek lehetnek osztály-, illetve példányszintűek. Az objektumoknak is vannak attribútumai és módszerei. Egy adott objektumosztályból származó objektumok attribútumai és módszerei azonosak, példányosításkor az osztály példányszintű attribútumait és módszereit kapja.

*Objektum állapota:* ez adja az objektum konkrétságát; minden objektumnak van egy saját, konkrét tárterülete. Példányosításkor annyi bájt tárterület kerül lefoglalásra, amilyen hosszú bitkombináció az objektum állapotának leírásához szükséges. Ez a tárterület rendelődik hozzá az objektumhoz.

*Objektum kezdőállapota:* az objektum azon állapotát tükröző bitkombináció, amely létrehozásakor, példányosításkor állítódik be.

Az objektum élettartama a példányosítással indul.

A nyelvek általában nem nevezik meg az objektumot, nincs nevük, viszont van *öntudatuk*; meg tudják különböztetni önmagukat az összes többi objektumtól, tudják, hogy melyik osztályból származnak. Az objektum maga is viselkedik, van viselkedésmódja, aktív, képes saját maga is cselekvésre.

Minden objektum önálló, rendelkezik ún. objektumazonosítóval (OID = `Object IDentifier`). Az OID-ek kiosztása, megvalósítása rendszerfüggő. Általában a rendszerek nem biztosítják az azonosítók teljeskörű egyediségét. Ha egy objektum megszűnik, OID-je nem adható másik objektumnak.

Az objektumok aktívak, ez az aktivitás abban mutatkozik meg, hogy az objektumok üzenetek (maguk is objektumok) segítségével kommunikálnak egymással (tehát az OO paradigma üzenet alapú). Egyik objektum üzenettel szólítja meg a másikat; kér tőle valamilyen szolgáltatást, esetleg elküldi a szükséges paramétereket. Innentől kezdve nem tudja, hogy az üzenettel mi történik (absztrakció). Az objektumok interfészen keresztül kommunikálnak,

csak azt tudják, hogy a másik objektumnak milyen üzenet küldhető és milyen választ kaphat rá. A mai informatikai rendszerek alapja az üzenetalapú OO-rendszerek. Azonban nem minden OO-nyelv ilyen.

Az objektumok módszerei

*Osztály példányszintű módszerei:* ezek mindig egy konkrét objektumon hajthatók végre, ezért mindig meg kell adni egy aktuális példányt.

*Osztály-attribútumok:* osztályhoz kapcsolódnak, azok bizonyos jellemzőit írják le adatokkal. Ezek csak egy példányban léteznek, és az osztálymódszerek módosítják őket.

*Osztály osztályszintű módszerei:* ezek nem konkrét objektumon hajthatók végre, ezért nincs aktuális példány.

*Osztály kiterjedtsége:* az osztályból származtatott példányok száma az adott időpillanatban.

Osztály objektummódszerei:

a) *beállító módszerek:* egy adott objektum vonatkozásában az objektumok állapotát változtatják meg; eljárásjellegűek.

b) *lekérdező módszerek:* segítségükkel egy objektum állapota kérdezhető le.

**Öröklődés:** az újrafelhasználás eszköze, osztályok közötti viszony, köztük értelmezett kapcsolat. Ez a viszony aszimmetrikus.

*Létezik az ősosztály.* Az alosztály örökli ősosztályának azon attribútumait és metódusait, melyet a bezárás nem tilt le, felhasználhatja ezeket, új attribútumokat és módszereket vezethet be, átnevezheti ezeket, megváltoztathatja a bezárást, újradefiniálhatja az attribútumokat és a módszereket. Bizonyos OO-rendszerek azt is megengedik, hogy az alosztály „eldobhassa” szülőosztályának egy vagy több attribútumát.

*Helyettesíthetőség*: ez jelenti az osztályok közötti viszony aszimmetriáját. Egy leszármazott osztály minden objektuma objektuma az ősosztályának. Mindenhol, ahol megjelenhet az ősosztály objektuma, ott megjelenhet az alosztály objektuma.

Az osztályok között tetszőleges öröklődési hierarchia építhető fel.

Kérdés: Egy osztálynak hány ősosztálya lehet?

a) *egy:* egyszeres öröklődés; fa hierarchia, melyben van egy kitüntetett elem, a gyökérelem, minden osztálynak ez az ősosztálya, az összes objektum az ő objektuma

b) *több:* többszörös öröklődés; gráf, melyben tiltott a ciklus (hurok)

Példa: grafikai rendszer egyszeres specializálódási öröklődési hierarchiája.

Az egy úton elhelyezkedő osztályok között leszármazott-elől viszony van. Azokat az osztályokat, amelyek között nincs ilyen viszony, *kliensosztályoknak* nevezzük.

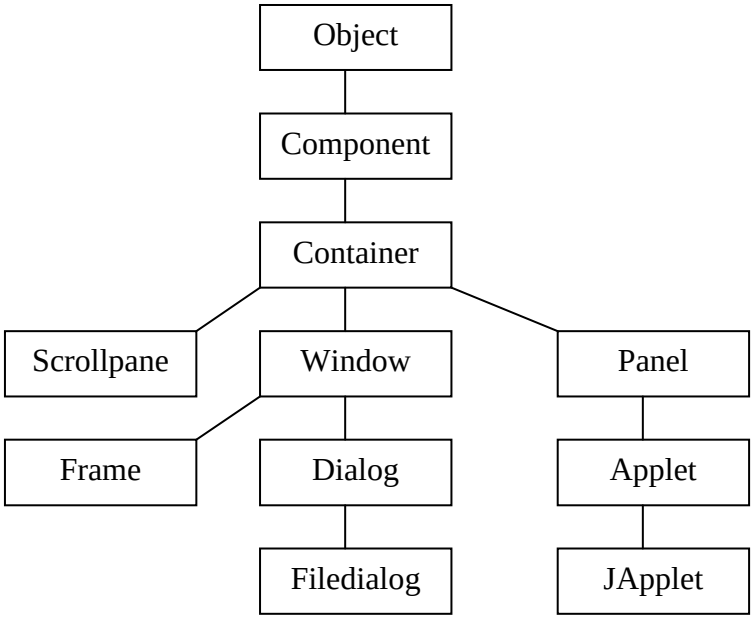
A bezárás főbb szintjei:

1. *privát:* adott eszközt csak az osztály látja (még az alosztályai sem)
2. *publikus*: minden osztály hozzá tud férni (még a kliensosztályok is)
3. csak az alosztályok, örökösök látják (a kliensosztályok nem)
4. fordítási egység

A további szintek az adott nyelvtől függnek.

Már eddig is használtuk az öröklődést még, ha nem is tudtunk róla, a `String`-nél. A Java osztályok egy- piramisszerű felépítést követnek. A piramis csúcsában az „Object” helyezkedik el. Ebből az osztályból örököltetünk. Minden alosztály örökli a felette elhelyezkedő osztály tulajdonságait.

Az öröklődési hiearchiát egy ábrával szemléltetem.




**1. ábra** *Egy osztály hiearchia*

Az ábrán jól látható, hogy például a `Frame` osztálynak négy szuperosztálya van. A `Frame` osztály mind a négy szuperosztályból örökölhet.

Ezt a fajta hiearchiát *családfát* a mindennapi életből vett példának a segítségével könnyebb megérteni. Az emberek családfája is így épül fel, például, a gyerek örökli a szülők és nagyszülők tulajdonságainak és viselkedésének egy részét.

**Az öröklődés létrehozása**

Egy osztályt egy másik osztály gyerekeként (alosztálya) az `extends` utasítással hozhatunk létre.

```
class Alosztaly extends javax.swing.JFrame {
        //Viselkedés és jellemzők
}
```

Az `extends` utasítás létrehozza az *Alosztaly* osztályt, mint a `JFrame` osztály alosztályát, a `javax.swing.JFrame` teljes osztálynév használatával.

Ha egy örökölt tagfüggvényt szeretnénk felülírni, akkor, ezt a következő képen tehetjük meg:

A `Component` osztály `paint()` tagfüggvénye a következő fél képpen kezdődik:

public void paint (Graphics g) {

Ha ezt az *Alosztaly* felül akarja írni akkor, a következőt kell tennünk:

public void paint (Graphics screen) {

A különbség a `Graphics` osztály neve, ez nem számít, ha a tagfüggvény létrehozása ugyan olyan módon történik. A két utasítás megfelel egymásnak, mert a `paint()` tagfüggvény nyilvános, egyik sem ad értéket vissza, mind a kettő egyetlen argumentum egy `Graphics` objektum.

**Ellenőrző feladatok**

1. Milyen utasítással érhetjük el, hogy egy osztálytól örököljön?

    a) new
    b) out
    c) extends

**Válasz:** c.) Az `extends` (bővít) kifejezést használjuk, mert a hierarchiában felette elhelyezkedő szuperosztály jellemzőinek és viselkedésének bővítése.

2. Mi az öröklődés?

**Válasz:** **Öröklődés:** az újrafelhasználás eszköze, osztályok közötti viszony, köztük értelmezett kapcsolat. Ez a viszony aszimmetrikus.

## Csomag

A Jávában fordítási egység az osztály vagy az interfész. (A fordítási egységben kell elhelyezni.) Minden fordítási egységet el kell helyezni egy csomagban. A csomag ilyen értelemben továbbra is csomag: egyrészt egy eszközgyűjtemény, csakhogy az eszköz most egy referenciatípust jelent (osztály vagy interfész), az összes többi eszköz ezen belül van; másrészt hatásköri egység, a láthatóságot szabályozza, a bezárást valósítja meg. Amíg az osztályhierarchia és az interfészgráfok inkább egy logikai kapcsolatot jelentenek, addig a csomagrendszer egy fizikai kapcsolathoz áll közelebb. Csomagrendszer van, a csomagok fákat alkotnak. Tehát a csomag az a könyvtárszerkezetre hasonlít (csomagokon belül alcsomagok, egymásba ágyazhatók tetszőleges szintig). Minden csomagnak saját neve van, a csomagokra való hivatkozás minősített csomagnévvel lehetséges. A csomagrendszerrel átmetsszük az osztályhierarchiát és a ráilleszkedő interfészgráfot. Összesítvén valamilyen összetartozó szempont alapján a referenciatípusokat és a hatáskört definiálván az osztályokra és az interfészekre vonatkozóan. Egy konkrét eszközre való hivatkozás úgy történik, hogy végig kell minősíteni, hogy mely csomagokban van:

csomag.alcsomag.osztály.objektum.eszköznév

Egy fordítási egység a következőképpen épül fel:

[csomagleírás]
[importleírás]
típusdefiníció
csomagleírás: `package` csomagnév;

Ekkor ez az osztály vagy interfész ebbe a csomagba fog bekerülni, ha csomagszintű láthatóság van, akkor erre a csomagra vonatkozik, és innentől kezdve ebben a csomagban érhető el. Amennyiben nem adunk meg csomagleírást, akkor a referenciatípusunk egy névtelen csomagba kerül, amely azért nem egészséges, mert ennek kezelése rendszerfüggő, platformfüggő. De kötelező csomagban elhelyezni, ezért egy alapértelmezett csomagba kerül. Innentől kezdve minden, amit csomagra vonatkozóan mondunk, erre a csomagra fogjuk érteni. Ez mindig egy egyedi csomag. Tehát ez nagyon lényeges, mert ezzel egy csomaghierarchiába bepakoltuk az osztályunkat vagy az interfészünket. Egy osztály vagy

interfész csak egy csomagba kerülhet. Ezért egy fordítási egységben legfeljebb csak egy csomagleírás lehet.

Az importleírás egy programozás-technikai eszköz.
`import` minősített_név;

A minősített név egy csomag csomaghierarchiában értelmezett, minősített neve. Ennek utolsó neve lehet * is. Az importálás arra szolgál, hogy a típusdefinícióban (törzsben) a minősítést ne kelljen mindig kiírni. A minősítéssel vagy adott osztály, vagy adott osztály összes nevét tudjuk hivatkozni, amely az adott csomagban helyezkedik el, azaz importáljuk az adott osztály interfészét (természetesen csak a látható részét). Így a hivatkozásoknál nem kell minősítenünk. Ezt csomagonként kell megtennünk, nem elég a gyökérelemet megadni. Ezért egy fordítási egységben importból sok lehet, mert több csomagból importálhatunk.

Maga a Java nyelv és a Java fejlesztői környezet egy csomagrendszerben van, viszont egy általunk írt alkalmazás is egy csomagrendszerbe kerül bele. Alapvető a java csomag és ennek alcsomagjai. A `java.lang` nevű csomagban vannak az alapvető nyelvi elemek: primitív típusok, kivételosztályok stb. Ennek az elemeit nem kell importálni, mert minden eleme automatikusan importálódik a fordítási egységhez. Minden egyes másik csomagot viszont importálni kell, ha minősítés nélkül használni kívánjuk az eszközeit.

**Ellenőrző feladatok**

1. Mi a csomag?

**Válasz:** a.) A Jávában fordítási egység az osztály vagy az interfész. Minden fordítási egységet el kell helyezni egy csomagban.

2. Helyes-e a felírt programrészlet?

import util . java . * ;

   a) igen
   b) nem

**Válasz:** b.) a helyes felírás: import java.util.*;

# Fájlkezelés

## Adatfolyamok

A Java programokban a tartós adattárolásra mindenképpen kell használnunk legalább egy adatfolyamot. Az adatfolyam, olyan objektum, mely az információt a forrásból máshová viszi.

Kétféle adatfolyam létezik:

- Bemeneti adatfolyam – adatokat olvas
- Kimeneti adatfolyam – adatokat ír

Minden be- és kimeneti adatfolyam bájtokból épül fel. A Java osztályok bináris alakban bájtokban tárolódnak, ezeket *bájtkódoknak* nevezzük. A Java-értelmező futathat más nyelvű bájtkódot is. E tulajdonsága miatt szokták *bájtkód-értelmezőnek* is nevezni.

A Java-ban a fájlokat a `java.io` csomag `File` osztálya kezeli. A `File` objektum már létező vagy még éppen létrehozandó fájlokat jelöl. Egy `File` objektum létrehozásához a fájl nevét és kiterjesztését kell megadni a konstruktornak.

File Iskolanev = new File (”Kossuth.dat”);

Ennek az utasításnak a segítségével egy `Kossuth.dat` nevű fájlt hoztunk létre. Ha nem adunk meg elérési utat, akkor a létrehozás az aktuális könyvtárban történik meg.

File Iskolanev = new File (”c:/Iskola/Egyetem/Kossuth.dat”);

A létező `File` objektumot meghívhatjuk, a következő tagfüggvények segítségével:

- `getName()` – A fájl nevének kezelése String-ként.
- `lenght()` – A fájl mérete long-ként.
- `createNewFile()` – Azonos nevű fájl létrehozása, ha a fájl még nem létezik
- `delete()` – A fájl törlése.
- `renameTo(Fájl)` – A fájl átnevezése.

A `File` objektum könyvtárakat is jelölhet. A könyvtár nevét a a konstruktorban adhatjuk meg (”c:/Iskola/Egyetem/Kossuth.dat”).

**Adatok olvasása**

Bemeneti adatfolyam segítségével adatot olvassunk egy fájlból. A `FileInputStream()` osztály segítségével bájtokat olvashatunk egy fájlból.

Bemeneti adatfolyamot a `FileInputStream()` konstrukor tagfüggvény segítségével hozhatunk létre, mely a fájl nevét vagy egy objektumot adunk át paraméterként. A fájlnak már léteznie kell a bemeneti adatfolyam létrehozása előtt. Ha a fájl még nem létezik és úgy próbálnánk belőle olvasni, akkor `IOException` kivétel lép fel. Ennek a kivételnek az elkerülése érdekében a fájllal kapcsolatos összes utasítást egy `try-catch` blokkba érdemes tenni.

Egy példa erre az utasításra:

```
try {
        File Iskolanev = new File(”kossuth.dat”);
        FileInputStream = new FileInputStream(Iskolanev);
        System.out.println(”A fájl hossza: ” + Iskolanev.length());
    } catch (IOException e) {
        System.out.println(”Nem sikerült a fájl olvasása!”);
}
```

Az adatokat bájtonként olvassák. Egyetlen bájtot a `read()` tagfüggvény paraméter nélküli megadásával olvashatjuk. Ha elértük a fájl végét, akkor a tagfüggvény (-1) értékkel tér vissza.

Ha az adatfolyam néhány bájtját ki akarjuk hagyni, akkor a `skip()` tagfüggvény egyetlen *int* paraméterrel hívjuk meg, mely a kihagyandó bájtok számát adja meg.

```
Iskolanev.skip(100);
```

**Adatok írása fájlba**

A `java.io` csomagban az adatfolyamokkal dolgozó osztályok mindig párosak, az olvasás és írás miatt. Ha bájtfolyamban dolgozunk a `FileInputStream` és a `FileOutputStream`, ha karakterfolyamokkal dolgozunk, akkor `FileReader` és `FileWriter` osztályokat használjuk.

Az adatok olvasásához először létre kell hoznunk egy `File` objektumot. A fájlnak nem kell létező fájlnak lennie, mivel ha nem létezik írásnál hozzuk létre.

Kétféle módon tudunk `FileOutputStream` osztályt létrehozni:

*Hozzáfűzéssel*: ilyenkor két paraméterrel hívjuk meg, ezek a fájl jelölő *File* objektum a másik a `true` logikai változó. A hozzáfűzéssel a bájtok az adatfolyam végére kerülnek. Ha a bájtokat új fájlba akarjuk írni, akkor csak `File` objektum paraméterrel hívjuk meg a konstruktort.

Ha már létezik kimeneti adatfolyam, különböző `write()` tagfüggvényekkel hívhatjuk meg a bájtok írására:

- Egyetlen bájt paraméterrel, ezzel egy adott bájtot írunk bele.
- Egy bájttömbbel, ilyenkor a tömb összes bájtját az adatfolyamba írjuk
- A `write()` tagfüggvényt három paraméterrel is meghívhatjuk:
    - Egy adattömbbel.
    - Egy egész számmal, mely az adatfolyamba írandó tömb első elemét jelöli.
    - Bájtok számával.

A következő példában egy öt elemű numerikus értékeket tartalmazó bájttömböt hozunk létre, és az utolsó két elemet egy fájlba írjuk.

```
File adat = new File(”adat.dat”);
FileOutputStream adatStream = new FileOutputStream(adat);
byte[] adat = new { 1, 2, 3, 4, 5};
adatStream.write (adat, 2, 2);
```

Ha az adatfolyam írása befejeződött, akkor a `close()` tagfüggvény meghívásával le kell zárni.

**Ellenőrző feladatok**

1. A `File` osztály melyik tagfüggvényt használja a fájl méretének a meghatározásár?

   a) read( )
   b) filsizes( )
   c) length( )

**Válasz:** c.) A `length ( )` tagfüggvény egy *long* értéket ad vissza.

2. Milyen adatfolyamatot használunk fájlból való olvasáshoz?

   a) bemeneti adatfolyamatot
   b) kimeneti adatfolyamatot
   c) be- és kimeneti adatfolyamatot

**Válasz:** a.) Bemeneti adatfolyamat egy `File` objektumból hozható létre.

3. Bemeneti adatfolyamatot mely tagfüggvénnyel hozhatunk létre?

   a) write( )
   b) FileInputStream ( )
   c) Read( )

**Válasz:** b.) Bemeneti adatfolyamat a `FileInputStream( )` konstruktor tagfüggvénnyel hozható létre, amelynek a fájlnevét vagy egy `File` objektumot adunk át paraméterként.

4. Milyen csomagot kell importálnunk, ha adatfolyammal akarunk dolgozni?

   a) Java
   b) import
   c) io

**Válasz:** c.) `java.io` csomagot

# Összegzés

„Mindennapi gyakorlatunkban meg növekedett az információ társadalmi szerepe, és felértékelődött az informálódás képessége. Az egyén érdeke, hogy időben hozzájusson a munkájához, az életvitele alakításához szükséges információkhoz, képes legyen azokat céljának megfelelően feldolgozni és alkalmazni. Ehhez el kell sajátítania a megfelelő információszerzési, információfeldolgozási és információátadási technikákat, valamint az információkezelés jogi és etikai szabályait (információk átvétele, bizalmas kezelése stb.).

E gyorsan változó, fejlődő területen nagyfokú az ismeretek elavulása is, ezért különösen fontos, hogy a tanulónak igénye legyen informatikai ismereteinek folyamatos megújítására.”[13]

A NAT felismerte ezt, ezért: „A tanterv az alapelvek és célok közé emelte az információ társadalmi szerepét, az információszerzés, információfeldolgozás, az adattárolás, adatszervezés, és átadási technikák képességének kialakítását, valamint az információkezelés jogi és etikai szabályainak ismeretét.”[14]

„Az utóbbi évtizedekben zajló információs forradalom sokrétűen hat mindennapjainkra. Az elektronikus adatkezelő eszközök fejlődésének és elterjedésének hatására kibővült az ember módszer- és eszköztára, a megoldható problémák köre. Az új eszközök közül sokoldalúságával és hozzáférhetőségével kiemelkedik a számítógép, mely önmagában is, de főleg számítógépes hálózatra kapcsolva újszerű probléma-megoldási lehetőségeket biztosít. E tudás jelentős része napjainkra az alapműveltség részének tekinthető.”[13]

A Java egy objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett a 90-es évek elejétől kezdve napjainkig. Mára a nyelv népszerűvé vált viszonylagos egyszerűsége, és objektumorientáltsága miatt. Jelentőségét abban kell keresni, hogy már kikristályosodott szabványokat, eszközöket foglal egy rendszerbe, ezáltal elérhetővé téve azt bárki számára.

Az informatikai állások esetében kiemelt fontossággal bír az aktuális információtechnológiai ismeretek napra készsége. Ez ugyanúgy igaz a programnyelvekre (Java, C, C++, C#, Net, PHP stb.), mint a hardveres technológiák ismeretére.

Mindez együtt vezetett ahhoz a gondolathoz, hogy ezzel a fiatal programnyelvvel való ismerkedést már a középiskolai tanulóknak el kellene kezdeni. Dolgozatomban kísérletet teszek arra, hogy a Java programnyelv megismeréséhez szükséges alapok oktatását elhelyezzem az informatika tantárgy rendszerében. Az ide kapcsolódó elméleti ismereteket részletezem. Az egyes témakörök elsajátítási szintjének ellenőrzéséhez feladatlapokat készítettem.

Bízom benne, hogy a középiskolai informatika oktatás megújul, lépést tart a fejlődéssel, és ennek során ez a fiatal programnyelv elfoglalja méltó helyét a tantárgy ismeretanyagában.

# Felhasznált irodalom


[1] http://www.inf.unideb.hu/~panovics/Programozas220040330.pdf

[2] http://www.inf.unideb.hu/~panovics/programozas120040519.pdf

[3] http://www.inf.unideb.hu/~bodai/prog/vezerlesi_szerkezetek.html

[4] http://www.inf.unideb.hu/~hajdua/prog1/prog1_jegyzet.pdf

[5] onik.inf.unideb.hu/@api/deki/files/35/=ptjegyz_v02.doc

[6] Adatszerkezetek és algoritmusok órai jegyzet.

[7] Angster Erzsébet: Objektumorientált tervezés és programozás, Java első kötet, 4KÖR Bt. Budapest, 2003

[8] Angster Erzsébet: Objektumorientált tervezés és programozás, Java második kötet, 4KÖR Bt. Budapest, 2004

[9] Rogers Cadenhead: Tanuljuk meg a Java programozási nyelvet 24 óra alatt, Kiskapu Kft. Budapest, 2006

[10] Móricz Attila: Java programozási nyelv I. „java.awt” csomag, Referenciakártya, LSI Oktatóközpont, Budapest, 1997

[11] Móricz Attila: Java Alapismeretek, LSI Oktatóközpont, Budapest, 1997

[12] Móricz Attila: Java programozási nyelv I., LSI Oktatóközpont, Budapest, 1997

[13] http://sagv.gyakg.u-szeged.hu/tantargy/NAT/8informk.htm

[14] http://www.sulinet.hu/tart/fcikk/Kacd/0/17705/2

[15] http://www.fppti.hu/szakteruletek/informatika/tanulmanyok/cikkek/helyitantervekrol.pdf

[16] http://www.umszki.hu/~zsigi/jegyzet.html

[17] http://indy.poliod.hu/program/java/modul.htm

[18] http://java.sun.com

[19] Zipernowsky Károly Műszaki Szakközépiskola – Informatikai alapismeretek tanmenet a 11. évfolyam informatika szakmacsoport számára.

# Köszönetnyilvánítás

Szeretnék köszönetet mondani néhány olyan embernek, akik hozzásegítettek ahhoz, hogy tanulmányaimat befejezhessem, és a diplomamunkám elkészüljön:

Először a témavezetőmnek, aki tanácsaival, ötleteivel terelgette gondolataimat a helyes irányba.

Másodszor a tanszék tanárainak, akik hozzájárultak tudásom gyarapításához.

Köszönöm a munkahelyem vezetőinek, amiért lehetőséget biztosítottak számomra, hogy a konzultációkon részt vehessek, hogy tanulhassak.

Végül de nem utolsó sorban a családomnak köszönöm a türelmet, a megértést, a bíztatást, a támogatást.