A Java nyelv alapjai

A&K AKADÉMIA

MARKOS ANDRÁS

Java alapok

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

Java alapok

Változók

Operátorok

Vezérlő szerkezetek

Változók (variable)

A változók adatok ideiglenes tárolására használhatók (futás időben).

A tárolandó adat típusa (type) és a nevük (azonosítójuk (identifier)) jellemző rájuk.

4 különböző fajta változó van:

- Példány változó
- Osztály változó
- Paraméter változó
- Helyi változó

Példány változó (non-static field)

Példa:

Nem statikus mezőnek is hívják (vagy egyszerűen csak mező)

Minden objektum példányban létezik belőle egy.

Hatóköre: Az adott példányon belül érvényes.

Példány változók

```
class Auto {

    int speed = 0;
    int gear = 0;

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void slowDown(int decrement) {
        speed = speed - decrement;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }
}
```

Osztály változó (static field)

Statikus mezőnek is hívják.

Egy létezik belőle, ami az osztályhoz tartozik.

A `static` kulcsszóval deklarálható.

Hatóköre: Az adott osztályon belül érvényes.

Osztály változó

```
class Auto {

    int speed = 0;
    int gear = 0;
    static int MAX_SPEED = 200;

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void slowDown(int decrement) {
        speed = speed - decrement;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }
}
```

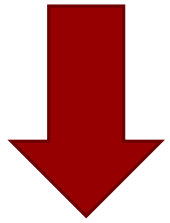
Paraméter változó (parameter)

Metódusok neve után zárójelben adható meg, ez a metódus bemenő adata.

Ugyanúgy van típusa és azonosítója.

Hatóköre: a metódus utasításblokkján belül érvényes.

Paraméter

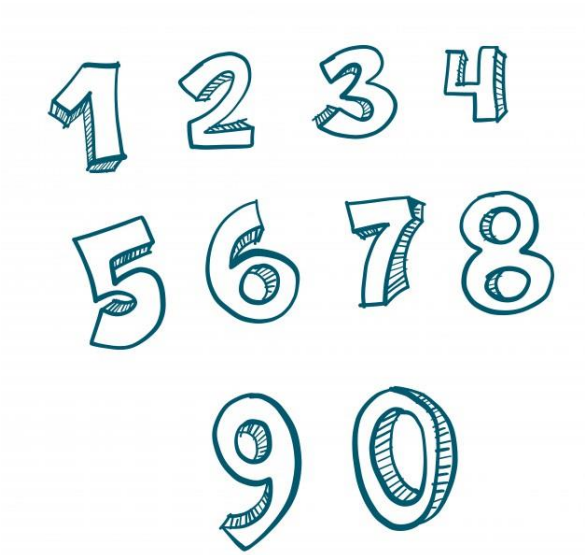


```
class Auto {

    int speed = 0;
    int gear = 0;

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void slowDown(int decrement) {
        speed = speed - decrement;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }
}
```

Helyi változó (local variable)

Metóduson belül deklarálható.

Hatóköre: A deklarált utasításblokkon belül érvényes.



```
void metodus() {
    int szam = 1;
    int i = 100;
    …
}
```

Helyi változó

Változók elnevezési konvenciói

A Java megkülönbözteti a kis- és nagybetűket!

A változó neveket kisbetűvel kezdjük.

Ha összetett szó a változó név, akkor a későbbiekben az összetett szót alkotó szavak kezdőbetűjét nagy betűvel írjuk (CamelCase)

Adjunk beszédes neveket a változóinknak!

A JLS szerint a változónév tetszőleges UNICODE karakterekből állhat, de csak betűvel vagy $ vagy _ szimbólummal kezdődhet. (számmal nem)

Kerüljük a $ és _ szimbólumokat! Java 8 warning-ot ad, Java 9 tiltja az _-et!

Példa:

```
int firstElementIndex;
```

Változók típusai

Java-ban egy változó a következő csoport valamelyikébe tartozik

- Primitív adattípus
- Referencia típus

Primitív adattípusból 8 van, ezek a Java beépített típusai.

Referencia típusból tetszőlegesen sok lehet.

Ami nem primitív típus, az referencia típusú (objektumra mutató referencia).

Primitív típusok

	byte	short	int	long	float	double	boolean	char
Méret	8 bit	16 bit	32 bit	64 bit	32 bit	64 bit	*8 bit (?)*	16 bit
Min	-128	-32768	-2147483648	-9223372036854775808	N/A	N/A	false	0
Max	127	32767	2147483647	9223372036854775807	N/A	N/A	true	65535
Típus	egész	egész	egész	egész	lebegő pontos	lebegő pontos	logikai	UNICODE karakter

- Mezők esetén a következő alapértékekre inicializálódnak a primitív adattípusok ->

Adattípus	Alapérték (default value)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
boolean	false
char	'\u0000'
Referencia típus	null

„God is real, unless declared integer."

Literálok

A primitív adattípusoknál nem használjuk a `new` operátort a létrehozásukhoz.

A primitív adattípusok nem osztályból származnak, hanem a Java nyelvbe beépített speciális típusok.

A **literál** a forráskódban egy fix érték, amely nem igényel számolást.

Példa: `int i = 2;`

Itt a 2-es egy literál.

Példák literálokra

```
int decVal = 26;          // 26 decimálisan

int octVal = 032;         // 26 oktálisan

int hexVal = 0x1a;        // 26 hexadecimálisan

long num01 = 26L;         // 26 mint long

double d1 = 123.4;        // double érték

double d2 = 1.234e2;      // ugyanaz az érték, mint d1,
                          // de normálalakban

double d3 = 3d;           // double érték

float f1 = 123.4f;        // float érték

char c1 = 'A';            // char érték

char c2 = '\u0108';       // UNICODE escape
```

Tömbök (array)

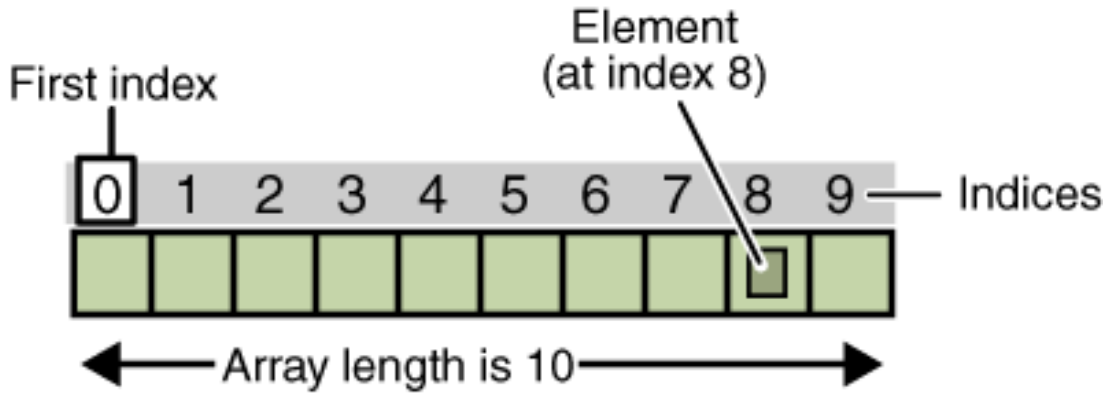
A tömb egy tároló objektum, ami képes több, azonos típusú adat tárolására.

Létrehozásakor kell megadni, hogy hány eleme lehet maximálisan.

Elemei index alapján érhetők el.

A kezdőindex a 0.

```
int[] tomb;           // tömb deklarálása
tomb = new int[10];   // memóriafoglalás 10 int számára
tomb[0] = 42;         // a tömb első elemének értéket adunk
tomb[1] = 100;        // a tömb második eleme 100 lesz
```



Tömbök (array)

Tömbre mutató referencia változó deklarálása:

```
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
int[] anArrayOfInts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
```

Tömbök (array)

Tömb létrehozása

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

VAGY

```
int[] tomb = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Ez utóbbi esetben a megadott értékekre inicializálódnak a tömb elemei, és maximális mérete a megadott értékek száma.

Értékadás a tömb egyes elemeinek

```
tomb[0] = 1900;
tomb[1] = 1999;
```

Érték lekérdezése és kiírása a képernyőre

```
System.out.println(„A tömb első eleme: ” + tomb[0]);
System.out.println(„A tömb második eleme: ” + tomb[1]);
```

Több dimenziós tömbök

Java-ban a több dimenziós tömb egyszerűen tömbök tömbje. (nem úgy mint a C-ben vagy Fortran-ban)

Ennek következménye, hogy a sorok hossza akár el is térhet.

Példa:

```
int[][] ketDimenziosTomb;      // deklarálás
ketDimenziosTomb = new int[10][10]; // helyfoglalás
ketDimenziosTomb[0][0] = 84;
ketDimenziosTomb[0][1] = 72;
…
ketDimenziosTomb[9][9] = 10;
```

Tömbök másolása

A `System` osztálynak létezik egy `arraycopy` metódusa, amely hatékonyan képes tömböket másolni:

```
public static void arraycopy(Object src, int srcPos, Object dest, int
 destPos, int length)
```

src -> másolandó tömb

srcPos -> kezdő pozíció a másolandó tömbben

dest -> új tömb

destPos -> új tömben a kezdőpozíció

length -> ennyi elem kerüljön másolásra

Tömb másolása példa

```
char[] copyFrom = {'s', 'i', 'k', 'e', 'r',
 't', 'e', 'l', 'e', 'n'};

char[] copyTo = new char[5];


System.arraycopy(copyFrom, 0, copyTo, 0, 5);

System.out.println(new String(copyTo));
```

Operátorok (operators)

A változókon műveleteket tudunk végezni operátorok segítségével.

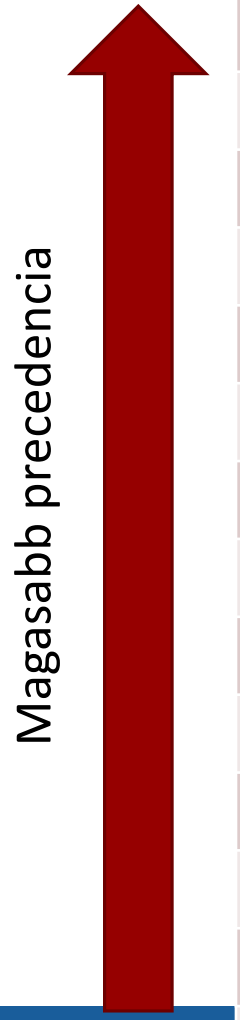
Az operátor egyik fontos tulajdonsága, hogy hány operandusa van.

Az összeadásnak például két operandusa van (2+2).

A logikai negálásnak csak egy operandusa van.

Fontos, hogy egy összetett kifejezésben milyen sorrendben hajtódnak végre az operátorok, ezt hívjuk precedenciának. (a szorzást előbb kell végrehajtani mint az összeadást) (2+3*4).

Operátorok precedenciája



Operátorok	Precedencia
postfix	`kif++  kif--`
egy operandusú	`++kif  --kif  +kif  -kif  ~  !`
multiplikatív	`*  /  %`
additív	`+  -`
shiftelő	`<<  >>  >>>`
relációs	`<  >  <=  >=  instanceof`
egyenlőség	`==  !=`
bitenkénti ÉS	`&`
bitenkénti kizáró VAGY	`^`
bitenkénti VAGY	`|`
logikai ÉS	`&&`
logikai VAGY	`||`
hármas	`? :`
hozzárendelő	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Operátorok (operators)

Példák

```
int a = 1;     // értékadás
int b = 5;     // értékadás
int c;
c = a + b;     // a és b összeadása, az eredmény c-be kerül
System.out.println(c);       // az eredmény kiíratása
```

Egyenlőség vizsgálatra példa

```
int a = 1;     // értékadás
int b = 5;     // értékadás
// egyenlőség vizsgálata
if (a == b) {
    System.out.println(”a és b változó értéke megegyezik.”);
}
```

Operátorok (operators)

Egy kifejezés kiértékelése során az eredmény típusa a kifejezésben résztvevő legtágabb típus lesz

Például:

`int + int` esetén az eredmény is `int`

`int + double` esetén az eredmény `double`

`int / int` eredménye `int` (egész osztás)

`int / double` eredménye `double`

A Java erősen típusos nyelv, ezért figyelnünk kell az eredmény típusára!

Operátorok (operators)

Átlagszámítás rosszul:

```
int[] tomb = {8, 4, 9, 5};
int osszeg = tomb[0] + tomb[1] + tomb[2] + tomb[3];
double atlag;
atlag = osszeg / 4;
System.out.println(atlag);
```

Eredmény:

```
6
```

Pedig 8 + 4 + 9 + 5 = 26

És $\frac{26}{4} = 6{,}5$

Operátorok (operators)

Átlagszámítás jól:

```
int[] tomb = {8, 4, 9, 5};
int osszeg = tomb[0] + tomb[1] + tomb[2] + tomb[3];
double atlag;
atlag = osszeg / 4.0;
System.out.println(atlag);
```

Eredmény:

```
6.5
```

Ha 4.0-val osztunk, akkor `int / double` osztás miatt az eredmény a tágabb `double` típusra bővül és valós osztást végez a Java

Operátorok (operators)

Kisebb relációra példa

```
int a = 1;      // értékadás
int b = 5;      // értékadás
if (a < b) {
    System.out.println(„a kisebb b-nél”);
}
```

Leírás	Kifejezés	Eredmény (a=1, b=5 esetén)
a egyenlő-e b	a==b	false
a nem egyenlő-e b	a!=b	true
a kisebb-e b	a<b	true
a kisebb vagy egyenlő-e mint b	a<=b	true
a nagyobb-e mint b	a>b	false
a nagyobb vagy egyenlő-e mint b	a>=b	false

Java-ban a relációs műveletek eredménye tehát egy igaz-hamis érték.

Operátorok összetett kifejezésben

Tetszőlegesen elbonyolíthatjuk a kifejezéseinket.

Például:

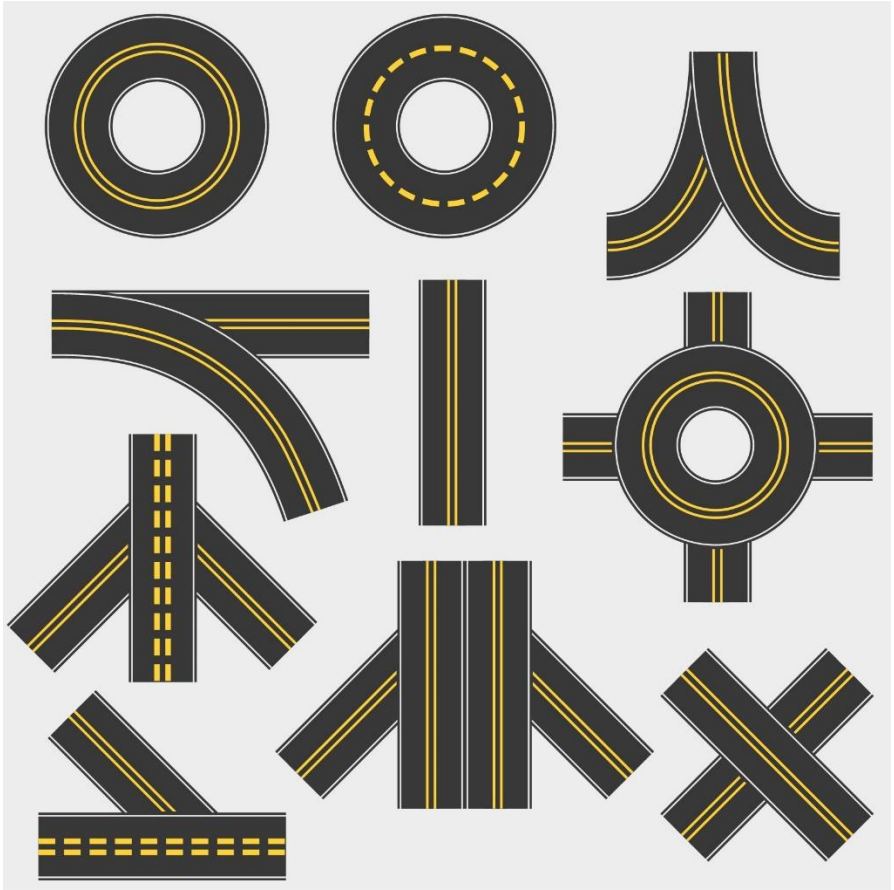
```
int a = 1;      // értékadás
int b = 5;      // értékadás
boolean c = true;
if ((c==false) || ((c == true) && (a < b))){
    System.out.println(„A feltétel most igaz.”);
}
```

Java vezérlő szerkezetek

Feltételes elágazások
- `if-then`
- `if-then-else`
- `switch`

Ciklusszervező utasítások
- `for`
- `while`
- `do while`

Vezérlésátadó utasítások
- `break`
- `continue`
- `return`



Utasításblokk

```
{
    utasítás1;
    utasítás2;
    …
    utasításN;
}
```

Az utasításblokkban található utasítások szekvenciálisan hajtódnak végre.

A végrehajtást megszakíthatja `break`, `continue`, `return` illetve kivételek (ld. később).

if-then

Általános alak:

```
if (kifejezés) {
    utasítás1;
    utasítás2;
    …
    utasításN;
}
```

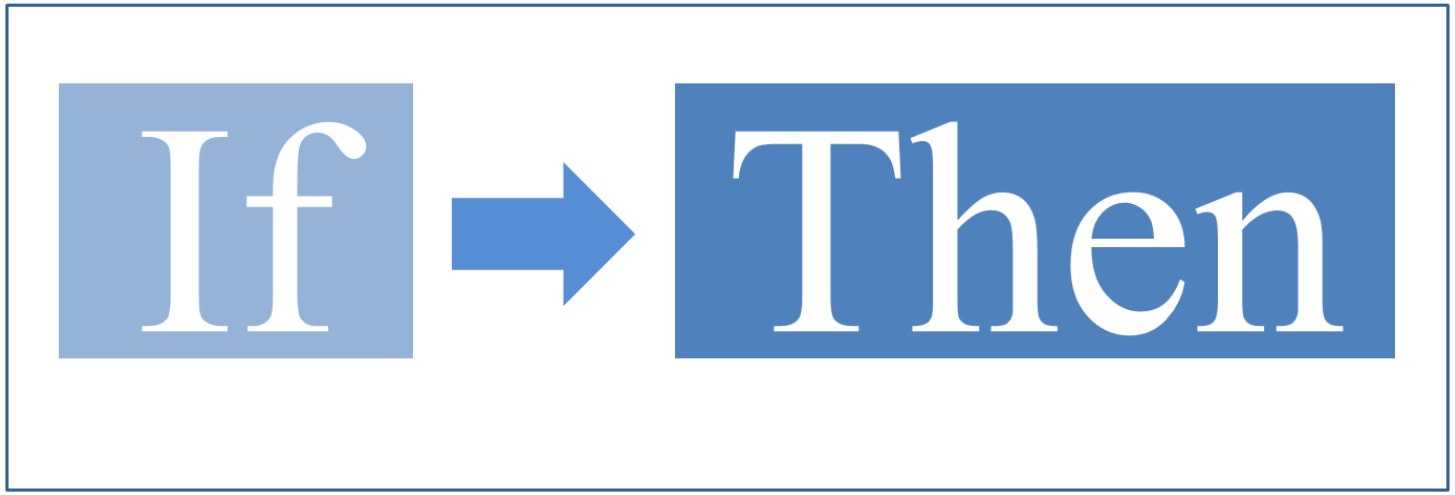
Ha a `kifejezés` értéke igaz, akkor végrehajtódik az `if` utasítás utáni utasításblokk, ha hamis, akkor nem.

if-then

Példa:

```
if (a > 5) {
    System.out.println(„Az a változó nagyobb mint 5.”);
}
```

Ha `a` 5-nél nagyobb, akkor a standard kimenetre kiírásra kerül a fenti szöveg.



if-then-else

Általános alak:

```
if (kifejezés) {
    utasítás1;
    utasítás2;
    …
    utasításN;
} else {
    utasítás1;
    utasítás2;
    …
    utasításN;
}
```

Ha a `kifejezés` értéke igaz, akkor végrehajtódik az `if` utasítás utáni utasításblokk, ha hamis, akkor az `else` utáni utasításblokk.

if-then-else

Példa:

```
boolean sikerult = true;
if (sikerult == true) {
    System.out.println(„A művelet sikeresen befejeződött.”);
} else {
    System.out.println(„A művelet sikertelen.”);
}
```

switch

```
switch (kifejezés) {
    case konstans1: …
    case konstans2: …
    …
    case konstansN: …
    default: …
}
```

A `switch` egy kifejezés alapján többfelé képes elágazni.

A `kifejezés` és a konstansok csak egész vagy `boolean` típusúak lehetnek.

A `case`-ek belépési pontokat jelölnek, a végrehajtás nem áll meg a következő `case` blokknál.

Ha a kifejezés egyik `case` címkével sem egyezik meg, akkor a `default` címke hajtódik végre.

switch

```
switch (eredmeny) {
    case 1: System.out.println(„Egy.”); break;
    case 2: System.out.println(„Kettő.”); break;
    case 3: System.out.println(„Három.”); break;
    default: System.out.println(eredmeny);
}
```

Ha `eredmeny` 1, 2 vagy 3, akkor ezen számok betűvel kiírt változatai kerülnek kiírásra a standard kimenetre, különben maga a szám.

for

Számlálásos ciklus

Általános alak:

```
for (kifejezés1; kifejezés2; kifejezés3) {
    utasítás1;
    utasítás2;
    …
    utasításN;
}
```

`kifejezes1` a ciklus indulásakor értékelődik ki, rendszerint a ciklusváltozót inicializálja

`kifejezes2` a ciklus minden futása előtt kiértékelődik, ha igaz az értéke, akkor a ciklus magja lefut

`kifejezes3` a ciklus minden futása után kiértékelődik, rendszerint a ciklusváltozót lépteti

for

Példa:

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

A program kiírja egymás alá új sorba 0-tól 9-ig az egész számokat.

A ciklus magja 10-szer fut le.

`i`-t itt egy helyi változóként deklaráltuk, csak a ciklusmagon belül érvényes, a ciklus lefutása után érvényét veszti.

`i++` megnöveli eggyel az `i` változó értékét minden ciklus lefutás után

for

Példa:

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

A ciklus során tehát a következők történnek:

1. `int i=0;` -> inicializálódik a ciklusváltozó 0-ra
2. `i < 10` kiértékelődik, ha `true`, akkor a ciklus magja lefut (ha `false` lenne, akkor a ciklus kilépne, és a ciklus utáni első utasításra ugrana)
3. `System.out.println(i);` lefut, ennek hatására kiíródik `i` változó tartalma
4. `i++` kiértékelődik, majd ugrunk a 2. lépésre

while

Elöltesztelős ciklus

Általános alak:

```
while (kifejezés) {
    utasítás1;
    utasítás2;
    …
    utasításN;
}
```

A `kifejezes` a ciklus bennmaradási feltétele, ha igaz, akkor a `while` utáni utasításblokk lefut, ha hamis, akkor a `while` utasításblokkja utáni első utasításra ugrik.

while

Példa:

```
int szam = 3;
while (szam < 10) {
    System.out.println(szam);
    szam++;
}
```

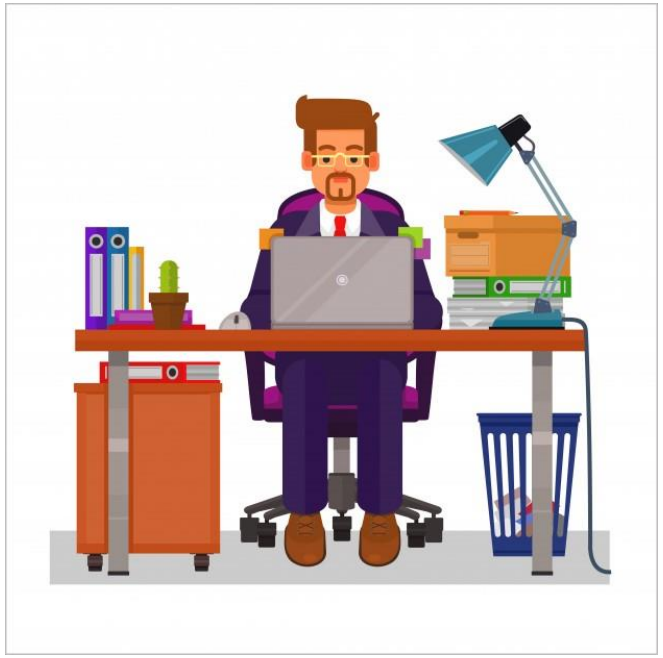
1. A `szam` változót beállítjuk 3-ra.
2. Kiértékelődik a `while` ciklus kifejezése (`szam < 10`), ami `true`, ezért végrehajtjuk a ciklusmagot.
3. Kiírjuk a szabványos kimenetre a `szam` változó értékét (3).
4. Megnöveljük eggyel a `szam` értékét.
5. Ugrunk a 2. lépésre.

do while

Hátultesztelős ciklus

Általános alak:

```
do {
    utasítás1;
    utasítás2;
    …
    utasításN;
} while (kifejezés);
```



A `kifejezes` a ciklus bennmaradási feltétele, ha igaz, akkor a ciklusmag újra lefut, ha hamis, akkor a ciklus utáni első utasításra ugrik.

A ciklusmag egyszer mindenképp lefut.

do while

Példa:

```
int szam = 3;
do {
    System.out.println(szam);
    szam++;
} while (szam < 5);
```

1. A `szam` változót beállítjuk 3-ra.
2. Kiírjuk a szabványos kimenetre a `szam` változó értékét (3).
3. Megnöveljük eggyel a `szam` értékét.
4. Kiértékeljük a `szam < 5` kifejezést, ha igaz, akkor ugrunk a 2. lépésre, különben kilépünk a ciklusból.

break

Két változata van, a **címkézett** és a **címkézetlen**.

Arra használható, hogy kilépjünk egy `switch`, `for`, `while` vagy `do while` utasításblokkjából.

A címkézetlen változat a legbelső utasításblokkból lép ki.

A címkézettel megadható, hogy melyikből lépjen ki.

címkézetlen break példa

```
int[] arrayOfInts = {32, 87, 3, 589, 12, 1076};
int searchfor = 12;
int i;
boolean foundIt = false;

for (i = 0; i < arrayOfInts.length; i++) {
    if (arrayOfInts[i] == searchfor) {
        foundIt = true;
        break;
    }
}

if (foundIt) {
    System.out.println("Found " + searchfor + " at index " + i);
} else {
    System.out.println(searchfor + " not in the array");
}
```

címkézett break példa

```
int[][] arrayOfInts = {{32, 87, 3, 589}, {12, 1076, 2000, 8}, {622, 127, 77, 955}};
int searchfor = 12;
int i;
int j = 0;
boolean foundIt = false;

search:
for (i = 0; i < arrayOfInts.length; i++) {
    for (j = 0; j < arrayOfInts[i].length; j++) {
        if (arrayOfInts[i][j] == searchfor) {
            foundIt = true;
            break search;
        }
    }
}

if (foundIt) {
    System.out.println("Found " + searchfor + " at " + i + ", " + j);
} else {
    System.out.println(searchfor + " is not in the array");
}
```

continue

Ennek is létezik **címkézetlen** és **címkézett** változata.

Ezen utasítás hatására a ciklus a ciklusmag maradék részét átugorja és a következő iterációt kezdi el. (ha a ciklusfeltétel ezt még megengedi)

Címkézetlen változata a legbelső ciklus aktuális iterációját ugorja át, a címkézett az adott címkével ellátott ciklusmagot ugorja át.

címkézetlen continue példa

```
String searchMe = "peter piper picked a peck of pickled peppers";
int max = searchMe.length();
int numPs = 0;

for (int i = 0; i < max; i++) {
    // interested only in p's
    if (searchMe.charAt(i) != 'p') {
        continue;
    }

    // process p's
    numPs++;
}

System.out.println("Found " + numPs + " p's in the string.");
```

címkézett continue példa

```
String searchMe = ”Look for a substring in me”;
String substring = ”sub”;
boolean foundIt = false;
int max = searchMe.length() - substring.length();

test:
for (int i = 0; i <= max; i++) {
    int n = substring.length();
    int j = i;
    int k = 0;
    while (n-- != 0) {
        if (searchMe.charAt(j++) != substring.charAt(k++)) {
            continue test;
        }
    }
    foundIt = true;
    break test;
}

System.out.println(foundIt ? "Found it" : "Didn't find it");
```

return

A `return` utasítás kilép az aktuális metódushívásból és visszatér a hívás helyére.

Két változata létezik

- Van visszatérési értéke
- Visszatérési érték nélküli

return visszatérési érték nélkül

```
static void szamol(int i) {
    if (i == 10) {
        System.out.println(”i pontosan tíz.”);
        return;
    } else {
        System.out.println(”i nem tíz.”);
    }
    System.out.println(”Vége a szamol metódusnak.”);
}

public static void main(String[] args) {
    System.out.println(”A program elindult.”);
    szamol(42);
    System.out.println(”A program lefutott.”);
}
```

return visszatérési értékkel

```
static int szamol(int i) {
    if (i % 2 == 0) {
        System.out.println(„i páros, ezért elosztom kettővel.”);
        return i/2;
    } else {
        System.out.println(„i páratlan, ezért megszorzom kettővel.”);
        return i*2;
    }
}

public static void main(String[] args) {
    System.out.println(„A program elindult.”);
    System.out.println(„A szamol metódus visszatérési értéke: ” + szamol(42));
    System.out.println(„A program lefutott.”);
}
```

Konstansok a Java-ban

Konstans: olyan változó, amelynek az értéke a futás során nem változtatható meg.

Java-ban nincs kifejezetten konstans típus.

Ha egy változót a következőképp deklarálunk, akkor mégis egy konstansszerű változót kapunk:

```
public static final int KONSTANS = 314;
```

`public` -> bárhonnan elérhető (erről később részletesen lesz szó)

`static` -> statikus mező, az osztályhoz tartozik

`final` -> ha egyszer értéket kap, akkor többé nem lehet megváltoztatni (és itt alapból értéket adunk neki)

Az ilyen változókat Java-ban **fordítás idejű konstans**oknak nevezzük.

Enum

Enumeration -> felsorolás

Az enum típus konstansok fix méretű halmaza.

Például a hét napjai, vagy az iránytű irányai:

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}


public enum CompassDirection {
    NORTH, EAST, SOUTH, WEST
}
```

Mindig célszerű használni, amikor fordítási időben tudjuk az összes választási lehetőséget.

Például főmenü menüpontjai…

Enum példa 1/2

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

static void joNapE(Day day) {
    switch (day) {
        case MONDAY:   System.out.println("Van jobb is."); break;
        case TUESDAY:  System.out.println("Nem olyan rossz."); break;
        case WEDNESDAY:System.out.println("A hét már félig letelt."); break;
        case THURSDAY: System.out.println("Mindjárt péntek."); break;
        case FRIDAY:   System.out.println("Végre péntek. Jó!"); break;
        case SATURDAY: System.out.println("Jobb!"); break;
        case SUNDAY:   System.out.println("Legjobb!"); break;
    }
}
```

Enum példa 2/2

```
public static void main(String[] args) {
        joNapE(Day.MONDAY);
        joNapE(Day.TUESDAY);
        joNapE(Day.WEDNESDAY);
        joNapE(Day.THURSDAY);
        joNapE(Day.FRIDAY);
        joNapE(Day.SATURDAY);
        joNapE(Day.SUNDAY);
}
```

Eredmény:

```
Van jobb is.
Nem olyan rossz.
A hét már félig letelt.
Mindjárt péntek.
Végre péntek. Jó!
Jobb!
Legjobb!
```

Enum

Az enum ennél sokkal többet tud!

Egy enum deklaráció egy osztályt definiál, vagyis lehetnek mezői és metódusai is.

Elsőként kell a konstansokat felsorolni vesszővel elválasztva, és a végén pontosvesszővel.

Utána jöhetnek a mezők és metódusok.

Enum példa

```
public enum Atvalto {
    EUR(0.004),
    HUF(1.000),
    USD(0.007),
    GBP(0.003);
    private final double atvaltasiArany;

    Atvalto(double arany) {
        atvaltasiArany = arany;
    }

    double atvalt(double penz) {
        return penz * atvaltasiArany;
    }

    public static void main(String[] args) {
        double atvaltando = 100000.0;
        for (Atvalto penz : Atvalto.values()) {
            System.out.printf("%f forint ennyi %s-et ér: %f\n", atvaltando, penz,
 penz.atvalt(atvaltando));
        }
    }
}
```