```
        viszont a Javadoc-ban nem fog megjelenni.
        */

    }
}
```

4.2. Egyszerű és összetett típusok

A programozás során az adatainkat (legyen az csak átmeneti is) ún. változókban tárolunk. Ezeket aprócska vödrökként kell elképzelnünk, melyeknek, ha van egy konkrét értéke, a vödörbe akkor lesz valamilyen tartalom. Viszont több ilyen típusú kis vödör létezik és nem mindegy, hogy melyik vödröcskébe mit pakolunk. Egy változó típusa megadja azt, hogy a változó milyen értékeket vehet fel.

Sok nyelvben, mint a Java – ban is megkülönböztetünk különböző típusokat, melyeket két fő csoportra oszthatunk:

1. *egyszerű adattípusok:* olyan adattípusok, mely nem bonthatók további részekre. Pl.: egész és valós szám típusok, karaktertípusok, stb…

2. *összetett adattípusok:* olyan adattípusok, adatszerkezetek, melyeknek az elemei mind – mind egyszerű adattípussal rendelkeznek. Pl.: tömbök, listák, vektorok, stb…

Az összetett adattípusok könyvünkben külön – külön fejezetet kaptak, így ezek mélyrehatóbb tárgyalása azon fejezetek feladata. Sokkal inkább lényegesebb most az azokat alkotó egyszerű adattípusok megismerése.

Egyszerű adattípusok:

1. **Egész számok**

   a. **byte:** előjeles egész — mérete 8 bit (1 byte) — *(-128 – tól + 127 - ig)*

   b. **short:** előjeles egész — mérete 16 bit (2 byte) — *(-32768 – tól + 32767 - ig)*

   c. **int:** előjeles egész — méret 32 bit (4 byte) — $(-2 * 10^9 + 2*10^9)$

   d. **long:** előjeles egész — méret 64 bit (8 byte) — $(-10^{19} + 10^{19})$

GYAKORLATIAS JAVA MINTA

2. **Valós számok**

   a. **float:** lebegőpontos mérete 32 bit (4 byte) *(IEEE 754)*

   b. **double:** lebegőpontos mérete 64 bit (8 byte) *(IEEE 754)*

3. **Karakter típus**

   a. **char:** karakter mérete 16 bit (2 byte) *(UNICODE)*

   A char típust áttekintve megemlítjük a gyakran alkalmazott String típust, mint összetett típust, mely tulajdonképpen felbontható char típusú karakterek sorozatává, azaz a String típusú változó egy karaktertömb.

4. **Logikai típus**

   a. **boolean:** logikai *TRUE/FALSE*

A típusok megadásának példáit a következő, *2.1.4.2* fejezet tartalmazza.

4.3. Azonosítók

Olyan karaktersorozat, amely betűvel kezdődik és betűvel, vagy számjeggyel folytatódhat. Arra használjuk, hogy az előző részben említett vödröcskéket egyedi módon elnevezzük vele és később ezen a néven hivatkozzunk rá. Azaz az azonosító arra szolgál, hogy a meghatározhassunk tetszőleges neveket változóinknak, függvényeinknek, osztályainknak, stb…

4.4. Változók

A programok adatokon végeznek különféle manipulációkat, ezek az adatok a program futása alatt változtatják értéküket. A kezdeti adatokat a program eltárolja a változókba, az értékek az algoritmusnak megfelelően módosulnak a program futása folyamán. Az algoritmus végén a változó a végeredmény értékével rendelkezik. A változó olyan adatelem, amely azonosítóval van ellátva. Az egyik legfontosabb programozási eszközünk.

GYAKORLATIAS JAVA MINTA

Nézzük is meg a következő részt, ahol a típusok, azonosítók és a változó együttes alkalmazásával értelmet nyer azok használata.

4.5. Változó deklaráció

Minden változó biztos, hogy rendelkezik valamilyen adattípussal, legyen az egyszerű, vagy összetett adattípus. A változó adattípusa az, ami meghatározza, hogy a létrehozott, deklarált változó milyen értékeket vehet fel és milyen műveletek végezhetők rajta.

Egy programban, amikor beírjuk a változó típusát és azonosítóját, akkor tulajdonképpen **deklaráljuk** a változót. Ennek szabálya, hogy először megadjuk a változó típusát, hogy abba milyen típusú elemeket szeretnénk tárolni, majd a változó tetszőlegesen választott azonosítóját, betartva a konvenciókat, végül pedig az utasítást pontosvesszővel zárjuk. Általános konvenció, hogy a változó neve:

- nem kezdődhet számmal (de a változó neve tartalmazhat számot az első karaktert követően)
- kisbetűvel kell kezdődnie
- nem lehet fenntartott kulcsszó
- szóösszetételek mentén az új szót nagybetűvel kezdjük
- nem túl hosszú, mégis beszédes
- lehet benne $ karakter és _ karakter

GYAKORLATIAS JAVA MINTA

A következő példában láthatjuk a deklarálás példáit az összes típusnak megfelelően.

```
package deklaracio;

public class Deklaracio {
  public static void main(String[] args) {
    //egész szám típusú változók deklarációja
    byte egeszSzam1;
    short egeszSzam2;
    int egeszSzam3;
    long egeszSzam4;

    //valós szám típusú változók deklarációja
    float valosSzam1;
    double valosSzam2;

    //karakter típusú változók deklarációja
```

De sokkal szemléletesebb, ha erre is mind nézünk egy-egy példát:

```
package logikaiop;

public class LogikaiOperator {

  public static void main(String[] args) {
    //kisebb
    System.out.println(10 < 7); //false
    //Magyarázat: Mivel a 10 nem kisebb mint 7, így ez false

    //nagyobb
    System.out.println(10 > 7); //true
    //Magyarázat: Mivel a 10 nagyobb mint 7, így ez true

    //kisebb, vagy egyenlő
    System.out.println(10 <= 15); //true
    //Magyarázat: Mivel a 10 kisebb, vagy egyenlő mint 15, így ez true

    //nagyobb, vagy egyenlő
    System.out.println(10 >= 10); //true
    //Magyarázat: Mivel a 10 nagyobb, vagy egyenlő mint 10, így ez true

    //egyenlő
    System.out.println(10 == 10); //true
    //Magyarázat: Mivel a 10 egyenlő 10 - el, így ez true

    //nem egyenlő
    System.out.println(11 != 10); //true
    //Magyarázat: Mivel a 11 nem egyenlő 10 - el, így ez true

    //negálás
    System.out.println(!(5<7)); //false
//Magyarázat: Mivel az 5 az kisebb mint 7, az true,
//annak a negáltja pedig false

    //logikai ÉS
    int szam1 = 15;
    System.out.println(szam1 > 10 && szam1 < 20); //true
    //Magyarázat: Mivel a 15 nagyobb mint 10 ÉS 15 kisebb mint 20, ezért
true

    //logikai VAGY
    int szam2 = 20;
    System.out.println(szam2 < 0 || szam2 > 10); //true
//Magyarázat: Mivel a 20 nem kisebb mint 0 DE a 20 nagyobb, mint 10,
//ezért true
  }

}
```

GYAKORLATIAS JAVA MINTA

Forrásfájl helye: Alapveto_java_programozasi_ismeretek / 01 / LogikaiOperatorok

- Értékadó operátorok

Az értékadó operátorok lényege, hogy valamilyen változónak adnak értéket.

Formája: változó_neve = kifejezés;

Az értékadás bal oldalán mindenképpen egy változónak kell szerepelnie, jobb oldalon pedig egy literál, változó, vagy olyan kifejezés (operátorok és operandusok összessége), amely egy értéket határoz meg, amit eltárolunk az értékadás bal oldalán lévő változóban. Az értékadó kifejezésben maga a bal oldali változó is szerepelhet, de csak akkor, ha a változót korábban inicializálták.

Operátor	Példa	Jelentés
+=	`int i = 0;` `i += 5;`	`int i = 0;` `i = i + 5;`
-=	`int i = 0;` `i -= 5;`	`int i = 0;` `i = i - 5;`
*=	`int i = 0;` `i *= 5;`	`int i = 0;` `i = i * 5;`
/=	`int i = 0;` `i /= 5;`	`int i = 0;` `i = i / 5;`
%=	`int i = 0;` `i %= 5;`	`int i = 0;` `i = i % 5;`

GYAKORLATIAS JAVA MINTA

Látható, hogy ott már a tanult aritmetikai operátorok rövidített formájának használatáról van szó. Ilyen operátor használatakor kiértékelésre kerül a jobb oldal, és a bal oldali változó értékét az értékadáshoz kapcsolt műveletnek megfelelően végzi el.

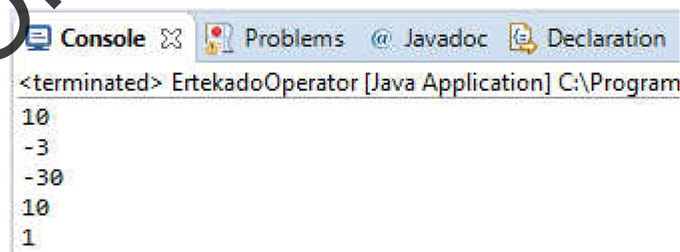
Fontos, hogy a kezdőérték megadásakor (inicializáció) ennek a rövidített formának a használata nem lehetséges, mert addig nem használható fel egy változó egy kifejezésben, amíg

nincs kezdőértéke! A táblázatban ennek a szemléltetése miatt szerepel mindenütt az i változó deklarációja és inicializációja.

De jöjjenek itt is a példák:

```
package ertekadoop;

public class ErtekadoOperator {

  public static void main(String[] args) {
    int szam1 = 3;
    int szam2 = 7;

    // +=  --> szam1 értéke 7+3=10 lesz
    System.out.println(szam1 += szam2);

    // -=  --> szam2 értéke 7-10=-3 lesz
    System.out.println(szam2 -= szam1);

    // *=  --> szam1 értéke 10*-3=-30 lesz
    System.out.println(szam1 *= szam2);


    // /= -->  szam1 értéke -30/-3=10 lesz
    System.out.println(szam1 /= szam2);

    // %= -->  szam1 értéke 10%-3=1 lesz
    System.out.println(szam1 %= szam2);
  }

}
```

A kimenet:



15. ábra: Értékadó operátorok példáinak kimenete

Forrásfájl helye: Alapveto_java_programozasi_ismeretek / 01 / ErtekadoOperatorok

GYAKORLATIAS JAVA MINTA

- Prefix és postfix operátorok

Ez a két speciális operátor nagyon egyszerűen teszi lehetővé az inkrementálás (1 – el való növelés) és a dekrementálás (1 – el való csökkentés) műveletét. Nagyon gyakori eset programozás során, hogy egy változó aktuális értékét 1 – el kell növelni, vagy 1 – el kell csökkenteni.

Az 1 – el való növelést inkrementálásnak, az 1 – el való csökkentést dekrementálásnak nevezzük. Mivel ezek olyan gyakran fordulnak elő, a Java – ban erre külön operátorok léteznek, **az inkrementálásra a ++ (plusz plusz)**, **a dekrementálásra a -- (mínusz mínusz)**.

Mindkét operátor használható **prefix operátorként (ilyenkor a változó neve elé kerül) és postfix operátorként (ilyenkor a változó neve után kerül)**. Amikor **prefix** operátorként alkalmazzuk őket (pl.: ++valt), akkor **előbb módosul a változó értéke és csak utána értékelődik ki a kifejezés értéke**, mely már az új értéket fogja felhasználni. **Postfix** operátor esetén **előbb értékelődik ki a kifejezés értéke**, mely még a változó eredeti értékét használja fel, majd **csak ezután módosul annak az értéke** (mely már a kifejezés értékére nincs kihatással).

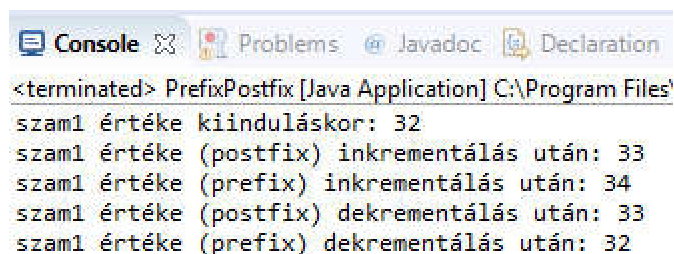
```
int valt = 0;
valt++;  //inkrementálás, egyenértékű ezzel: valt = valt + 1;
valt--;  //dekrementálás, egyenértékű ezzel: valt = valt - 1;
```

Először, mint önálló utasításként használva nézzünk erre példákat:

```
package prefixpostfix;
public class PrefixPostfix {
  public static void main(String[] args) {
    //Önálló utasításként való alkalmazás
    int szam1 = 32;
    System.out.println("szam1 értéke kiinduláskor: " + szam1);   //32

    szam1++;  //33
    System.out.println("szam1 értéke (postfix) inkrementálás után: " +
szam1);
    ++szam1;  //34
    System.out.println("szam1 értéke (prefix) inkrementálás után: " +
szam1);
    szam1--;  //33
    System.out.println("szam1 értéke (postfix) dekrementálás után: " +
szam1);

    --szam1;  //32
```

GYAKORLATIAS JAVA MINTA

```
     System.out.println("szam1 értéke (prefix) dekrementálás után: " +
szam1);
   }
}
```



16. ábra: Az inkrementálás és dekrementálás alkalmazása önálló utasításként

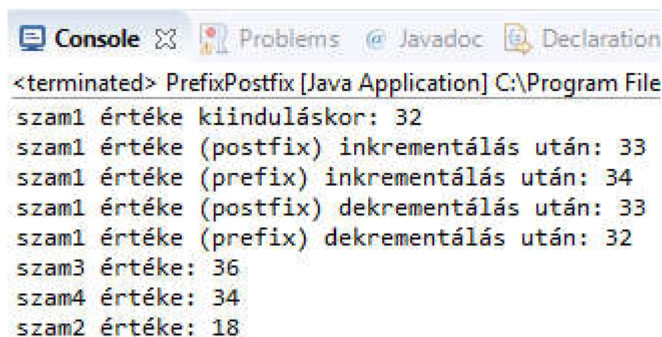
Forrásfájl helye: Alapveto_java_programozasi_ismeretek / 01 / PrefixPostfix

A fenti példákban látható, hogy önálló utasításként használva a prefix és postfix operátorokat tulajdonképpen ugyanaz a hatás. De teljesen más működést lehet előidézni, ha az inkrementálás, vagy dekrementálás egy kifejezés belsejében történik meg.

Az előző példát folytatva:

```
//Kifejezés belsejében való alkalmazás
int szam2 = 17;
int szam3 = ++szam2 * 2; //szam2 = 18, így szam3=36 lesz
System.out.println("szam3 értéke: " + szam3);

szam2 = 17;  //ugyebár előzőleg 18 lett, msot visszaállítjuk 17-re
int szam4 = szam2++ * 2;
//az eredmény 34 lesz, majd az eredmény kiszámítása után a szam2 18 lesz

System.out.println("szam4 értéke: " + szam4 + "\nszam2 értéke: " +
szam2);
```

És a kimenet futtatás után:



17. ábra: Az inkrementáló és dekrementáló operátorok használata önálló kifejezésként és utasítás belsejében

GYAKORLATIAS JAVA MINTA

Forrásfájl helye: Alapveto_java_programozasi_ismeretek / 01 / PrefixPostfix

Inkrementáló és dekrementáló utasítás prefix és postfix operátorokkal bárhol használható a programban, ahova egyébként változót írnánk.

GYAKORLATIAS JAVA MINTA

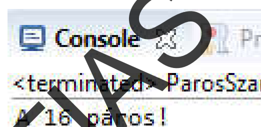
5.2. Elágazások

5.2.1. Az IF utasítás

Az egyszerű IF utasítás az egyik legalapvetőbb elágazást kezelő utasítás. Feladata, hogy adott kód részt csak akkor hajtson végre, ha a megadott feltételes utasítás értéke igaz. Nézzünk egy példát: a 16 – os számról döntsük el, hogy az páros-e és ha igen, akkor az írja ki a program.

```
package parosszamok;

public class ParosSzam {
   public static void main (String args[]) {
      int szam = 16;
      if (szam % 2 == 0) {
         System.out.println("A " + szam + " páros!");
      }
      //ha a szam változó páratlan, akkor ide ugrik a vezérlés
   }
}
```

Eredmény:



Forrásfájl helye: Alapveto_java_programozasi_ismeretek / 02 / ParosSzamok

A fenti program megvizsgálja, hogy a `szam` változó mod 2 az nulla-e. Ha nulla (tehát, ha nincs a `16 / 2` osztásnak maradéka), akkor a `szam` az páros. És mivel igaz a feltétel, így a vezérlés az IF utasítás blokkjában folytatódik. **Ha a `szam` változó értéke esetleg nem lenne páros, akkor az utasítás végrehajtás az IF utasítás záró kapcsos zárójele utáni sorra ugrik.**

SZINTAKTIKA

```
      if (felteteles_utasitas) {
         //ha a felteteles_utasitas igaz értékű,
         //akkor a vezérlés ebbe a blokkba kerül,
         //az itt levő utasítások végrehajtódnak
         utasitas1;
         utasitas2;
         ……
         utasitasN;
```

GYAKORLATIAS JAVA MINTA

```
    }
    //Ha a felteteles_utasitas hamis értékű, akkor a programvezérlés itt
    //folytatódik úgy, hogy a { és } közötti utasítás nem hajtódik végre.
```

Fontos megemlíteni, hogy az IF utasításnál a nyitó és kapcsos zárójel elhagyható, de csak akkor, ha a feltételes utasítás igaz eredménye esetén csak EGY utasítást szeretnénk végrehajtani. Tehát a szintaktika egy utasítás végrehajtása esetén:

```
    if (felteteles_utasitas)
      //ha a felteteles_utasitas igaz értékű,
      //akkor az itt levő EGY utasítás végrehajtódik
      utasitas1;
```

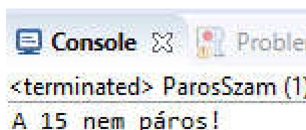
Egyébként ezt a rövidebb formulát nem célszerű alkalmazni és általában nem is szoktuk, ugyanis sokkal átláthatóbb a kód, ha minden vezérlési szerkezet szemmel látható blokkokban van elkülönítve.

5.2.2. Az IF – ELSE utasítás

Ez az elágazás az előző kiegészítése azzal, hogy lekezeli azt az esetet is, ha a feltételes utasítás hamis értékű. Nézzük újra az előző példát, de most a 15 – ös számról döntsük el, hogy az páros-e és ha igen, akkor az írja ki a program, ha nem páros azt is jelezzük.

```
package parosszamok;

public class ParosSzam {
  public static void main(String[] args) {
    int szam = 15;
    if (szam % 2 == 0) {
      System.out.println("A " + szam + " páros!");
    } else {
      System.out.println("A " + szam + " nem páros!");
    }
  }
}
```

Eredmény:



Forrásfájl helye: Alapveto_java_programozasi_ismeretek / 02 / ParosSzamok2

GYAKORLATIAS JAVA MINTA

A fenti program megvizsgálja, hogy a `szam` változó mod 2 az nulla-e. Ha nulla (tehát, ha nincs a `15 / 2` osztásnak maradéka), akkor a `szam` az páros, egyébként páratlan. És mivel most hamis a feltétel, így a vezérlés a hamis ágra kerül, azaz az else utasítás blokkjában folytatódik a vezérlés. **Ha a `szam` változó értéke esetleg páros lenne, akkor az utasítás végrehajtás természetesen az IF utasítás igaz blokkjába kerülne.**

SZINTAKTIKA

```
if (felteteles_utasitas) {
  //ha a felteteles_utasitas igaz értékű,
  //akkor a vezérlés ebbe a blokkba kerül,
  //az itt levő utasítások végrehajtódnak
  utasitas1;
  utasitas2;
  ……
  utasitasN;
} else {
  //ha a felteteles_utasitas hamis értékű,
  //akkor a vezérlés ebbe a blokkba kerül,
  //az itt levő utasítások végrehajtódnak
  utasitas1;
  utasitas2;
  ……
  utasitasN;
}
```

Fontos megemlíteni itt is, hogy az IF - ELSE szerkezetnél is a nyitó és kapcsos zárójel elhagyható, de csak akkor, ha a feltételes utasítás igaz eredménye esetén csak EGY utasítást szeretnénk végrehajtani, illetve a hamis ág esetén is csak EGY utasítást. Tehát a szintaktika egy utasítás végrehajtása esetén:

```
if (felteteles_utasitas)
  //ha a felteteles_utasitas igaz értékű,
  //akkor az itt levő EGY utasítás végrehajtódik
  utasitas1;
else
  //ha a felteteles_utasitas hamis értékű,
  //akkor az itt levő EGY utasítás végrehajtódik
  utasitas1;
```

Ezt a rövidebb formulát sem célszerű alkalmazni és általában nem is szoktuk, ugyanis sokkal átláthatóbb a kód, ha minden vezérlési szerkezet szemmel látható blokkokban van elkülönítve.

GYAKORLATIAS JAVA MINTA

6. PROGRAMOZÁSI TÉTELEK

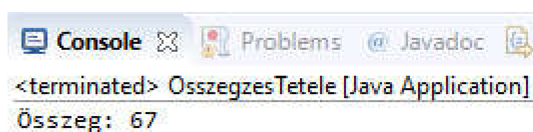
Egy olyan fejezethez értünk, melyet a további fejezetek feldolgozásához, illetve az előző fejezetek elmélyítéséhez mindenképpen szükséges megértenünk és készség szinten begyakorolnunk. Akármilyen hihetetlennek tűnik, nagy ipari alkalmazások egyes funkciói is mind – mind visszavezethetők valamelyik programozási tételre. A programozási tételek tulajdonképpen a leggyakrabban alkalmazott algoritmusokat jelentik. A programozási tételeket a legtöbb esetben valamilyen sorozaton hajtunk végre, ez a sorozat általában egy tömb elemeit foglalja magába. A következőkben a leggyakrabban alkalmazott programozási tételeket tekintjük át (de ez nem azt jelenti, hogy az itt leírtakon kívül nincs több, viszont funkcióiban a többinek ugyanaz a célja, pl.: többféle rendezési algoritmus van: buborékrendezés, minimum kiválasztásos rendezés, shell rendezés, stb...).

6.1. Összegzés

Egy meghatározott sorozat elemeinek összegét határozza meg. Példaként számítsuk ki egy tetszőleges számokat tartalmazó tömb elemeinek összegét.

```
package programozasitetelek;

public class OsszegzesTetele {

  public static void main(String[] args) {
    /* 1. Összegzés tétele:
     * egy tömb elemeinek összegzése
     */
    int[] szamok = {3,1,7,4,9,2,7,11,23};
    int osszeg = 0;

    for (int i = 0; i < szamok.length; i++) {
      osszeg += szamok[i];
    }

    System.out.println("Összeg: " + osszeg);
  }
}
```

Eredmény:



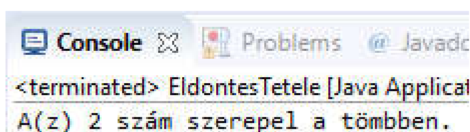
GYAKORLATIAS JAVA MINTA

6.2. Eldöntés

Az eldöntés tétel lényege, hogy az algoritmus eldönti, hogy van – e olyan elem a sorozatban, mely rendelkezik egy bizonyos tulajdonsággal (amelyet a feladat megkíván). Ha talál egy olyan elemet, ami megfelel a feltételnek, a ciklus megáll. Ha a ciklus úgy áll le, hogy nem volt benne olyan elem, mely a feladat által meghatározott, akkor nincs a sorozatban olyan elem, mely megfelelne a kritériumnak. A következő példában döntsük el, hogy egy tetszőleges számú tömbben szerepel – e a 2 – es szám.

```
package eldontestetele;

public class EldontesTetele {

  public static void main(String[] args) {
    /* 2. Eldöntés tétele:
     * adott tulajdonságú elem létezésének vizsgálata
     */
    int[] szamok = {3,1,7,4,9,2,7,11,23};
    int keresendo = 2;

    int i = 0;
    while (i < szamok.length && szamok[i] != keresendo) {
      i++;
    }

    if (i < szamok.length) {
      System.out.println("A(z) " + keresendo + " szám szerepel a tömbben.");
    } else {
      System.out.println("A(z) " + keresendo + " szám nem szerepel a tömb-
ben.");
    }
  }

}
```

Eredmény:



A fenti példában nagyon fontos a ciklusfeltétel sorrendje, valamint érdemes még megjegyezni, hogy a 2. feltétel (`szamok[i] != keresendo`) biztosítja azt, hogy a ciklus ne fusson tovább, ha a feladat által meghatározott feltétel (van – e 2 – es szám a tömbben) teljesül.

GYAKORLATIAS JAVA MINTA

Érezhető a metódus és függvény szorossága, mely teljes mértékben jogos is, de egy, a prog.hu oldalon írt hozzászólás jó megfogalmazása szerint a két fogalom között részhalmaz reláció van, azaz **minden metódus függvény, de csak az a függvény metódus, ami szorosan az objektumhoz kötött.**

Nézzük egy szemléltető példát: kapunk egy olyan feladatot, hogy számítsuk ki egy diák tanulmányi átlagát, majd számítsuk ki az osztály tanulmányi átlagát is. Egyből látni, hogy átlagot kétszer kell számolnunk a programunkban, így célszerű azt kiemelni egy függvényben és az átlagszámításnál a függvényt meghívni az aktuális értékekkel. (Majd, ha az átlagokat el szeretnénk menteni egy objektum jellemzőjeként, akkor meghívjuk az objektum átlagra vonatkozó beállító, azaz ún. setter metódusát, erről részletesen a II. kötetben!).

7.1. Saját függvények, paraméterátadás

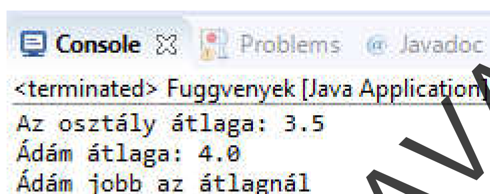
Gyorsan írjunk egy példakódot, hogy ez a rengeteg elmélet ne vegye el a kedvünk. A feladat: van egy 10 fős csoport, akiknek Informatika jegyei év végén: 4, 3, 4, 5, 2, 3, 3, 5, 3, 3. Ádámnak az Informatika jegyei az év során az alábbiak szerint alakultak: 5, 3, 2, 4, 5, 5. Számítsa ki egy program, hogy mi az osztály Informatika átlaga, illetve nézzük meg, hogy Ádámnak milyen az átlaga Informatikából. Ha jobb átlaga van Ádámnak, akkor írjuk ki, hogy Ádám jobb az átlagnál, ellenkező esetben írjuk ki, hogy Ádám rosszabb az átlagnál.

```
package fuggvenyek;

public class Fuggvenyek {

  public static void main(String[] args) {
    int[] erdemjegyekOsztaly = new int[] {4, 3, 4, 5, 2, 3, 3, 5, 3, 3};
    int[] erdemjegyekAdam = new int[] {5, 3, 2, 4, 5, 5};

    double osztalyAtlaga = atlagSzamitas(erdemjegyekOsztaly);
    double adamAtlaga = atlagSzamitas(erdemjegyekAdam);

    System.out.println("Az osztály átlaga: " + osztalyAtlaga);
    System.out.println("Ádám átlaga: " + adamAtlaga);

    if (adamAtlaga > osztalyAtlaga) {
      System.out.println("Ádám jobb az átlagnál");
    } else {
      System.out.println("Ádám rosszabb az átlagnál");
    }
  }

  public static double atlagSzamitas(int[] szamok) {
    double atlag = 0;
```

GYAKORLATIAS JAVA MINTA

```
        int darab = 0;

        if (szamok != null && szamok.length > 0) {
            int szamokOsszege = 0;
            darab = szamok.length;
            for (int szam : szamok) {
                szamokOsszege += szam;
            }
            atlag = (double)szamokOsszege / darab;
        }

        return atlag;
    }
}
```

Eredmény:



Először nézzük meg egy függvény szintaktikáját:

```
[lathatosag] [modosito] [vissz_ertek_tipusa] fuggvenyNev([formalis_param]) {

    [return vissz_ertek]
}
```

A szögletes zárójelben levő adatok megadása opcionális. Jelen példánknál maradva elemezzük ki a függvényünk definiálását: `public static double atlagSzamitas(int[] szamok)` rögtön két olyan kulcsszóval kezdődik, amit még nem tudhatunk (a `public static` rész), ezekre részletesen kitérünk a II. kötetben. A `static` egyébként azért kell, mert a `main` metódus kötelezően statikus és a Java nem engedi használni statikus környezetben a nem statikus függvényeket. A `public` a függvény láthatóságát jelöli. Most a feladatok megoldásánál csak fogadjuk el, hogy egyelőre ezzel kell kezdeni új függvények megadását.

A következő a `double` kulcsszó, amivel azt mondom meg, hogy ez az átlagszámító függvény eredményül egy `double` típusú értéket fog adni. A visszatérési érték típusa tulajdonképpen bármi lehet, azt a feladat leírása fogja meghatározni. Van egy speciális kulcsszó, a `void`, ami azt jelenti, hogy a függvénynek nem lesz visszatérési értéke, az nem ad vissza egy konkrét eredményt. Egyébként az olyan függvény, aminek nincs visszatérési értéke, azt eljárásnak is szokás nevezni.

GYAKORLATIAS JAVA MINTA

Nézzük tovább, most jön a függvény neve, ami tetszőleges lehet, de általában kisbetűvel kezdjük, valamint a szóösszetételek mentén nagybetűt írunk (`atlagSzamitas`). A függvény neve után mindenképpen ki kell tennünk egy nyitó és záró gömbölyű zárójelet, ez kötelező. A zárójelbe viszont már nem kötelező megadni paraméter(eke)t (ezekre a paraméterekre azonnal kitérünk). Ezt követően pedig egy nyitó és záró kapcsos zárójel közé jöhet a függvény kódja.

Ha a függvény létrehozásakor adtunk meg visszatérési érték típust (tehát az nem `void`), akkor a függvény kapcsos zárójelbeli utolsó utasítása biztosan egy `return` utasítás lesz, amit pedig a visszatérési érték követ. Ha `void` a függvény visszatérési értéke, akkor a `return` utasítást nem tehetjük ki. Egyszerűbben: ha `void` van megadva, akkor nem kell a `return` a függvény végén, ha nem `void` van megadva a visszatérési érték típusának a függvénydefinícióban, akkor mindenképp kell majd a `return` kulcsszó és utána a visszatérési érték.

Nézzük ezt egy ábrán szemléltetve:

Egyéb módosítók
Függvény neve
Formális paraméterlista
Visszatérési érték tipusa

```
public static double atlagSzamitas(int[] szamok) {
    double atlag = 0;
    int darab = 0;

    if (szamok != null && szamok.length > 0) {
        int szamokOsszege = 0;
        darab = szamok.length;
        for (int szam : szamok) {
            szamokOsszege += szam;
        }
        atlag = (double)szamokOsszege / darab;
    }

    return atlag;
}
```

A visszatérési értéknek double van megadva, igy gondoskodni kell róla, hogy a visszatérési értéket tároló változó tipusa is double legyen.

Ugyebár a vissz. érték nev void, igy kell a return és maga az érték.

GYAKORLATIAS JAVA MINTA

A függvény hívás menete lépésről lépésre:

```
public static void main(String[] args) {
    ...
    ...
    double osztalyAtlaga = atlagSzamitas(erdemjegyekOsztaly);
    double adamAtlaga = atlagSzamitas(erdemjegyekAdam);
    ...
    ...
}
```

1. Függvényhívás

2. a függvény kiszámolta az osztály átlagát és letárolta a változóba

3. Most Ádám átlagát számítja ki, majd menti le a változóba

4.

A 4. pontban a vezérlés újra a main függvényben van, de már úgy, hogy a program kiszámolta a szükséges átlagokat.

```
public static double atlagSzamitas(int[] szamok) {
    double atlag = ...

    ...

    return atlag;
}
```

7.1.1. Paraméterátadás

Még egy dologról nem beszéltünk, a formális és aktuális paraméterlistáról. A formális paraméterlista az, amit a függvény definíció tartalmaz, az aktuális paraméterlista pedig az, amit a függvény meghívásakor megadunk a zárójelben. Fontos, hogy az aktuális paraméterlistának illeszkednie kell a formális paraméterlistára, azaz a példánkban az aktuális paraméterlistának mindenképp egy Integer értékeket tartalmazó tömböt kell átadnunk. Egy függvénynek tetszőleges számú és típusú formális paraméter listája lehet. Nézzünk erről is egy ábrát:

GYAKORLATIAS JAVA MINTA

A harmadik lépésben kezdjük el a Debug – ot. Kattintsunk a kód ablakba, a kódsorba, hogy ott legyen a kurzor (tetszőleges helyen lehet). Látjuk, hogy az elhelyezett ún. break point – nál megszakadt a program végrehajtása és a programozó interakcióját várja a debugger (zöld háttér jelöli a programegszakítás helyét).

Kezdjük el lassan nyomkodni az F6 billentyűt (soronként lépteti a kódvégrehajtás menetét), de közben figyeljük a variables ablakban, hogy mi történik. Egyszer megnyomva, látjuk, hogy létrejön a 10 elemet tartalmazó tömbünk, melyek mind 0 értékkel inicializálódnak:

```
3  public class DebugExample {
4
5      public static void main(String[] args) {
6          int[] parosSzamok = new int[10];
7          for (int i = 0; i < 10; i++) {
8              if (i % 2 == 0) {
9                  parosSzamok[i] = i;
10             }
11         }
```

Nyomjuk meg mégegyszer az F6 – ot, ekkor az IF vizsgálatra ugrik a vezérlés, mivel a 0 páros, így a következő F6 –ra a program felülírja a tömb 0. indexének aktuális értékét, jelent esetben 0 – val. Kezdjük el nyomkodni az F6 – ot szépen lassan és figyeljük a Variables ablak változásait. Látható, hogy a vezérlés nem ugrik bele az if törzsébe, ha az i változó értéke nem páros, azaz nem írja felül a tömb i. indexű értékét, az marad nulla. Szépen nyomkodva az F6 – ot, a debugger kiemeli az éppen aktuális indexen levő érték változtatását:

GYAKORLATIAS JAVA MINTA