**2. modul**

Programming Basics

Java Basics

oop basics, programming basics, java basics, java core, junit, sql basics

# Szintaxis

Mint minden beszélt nyelvnek, úgy a programozásban használt nyelveknek is vannak szabályai, amelyeket be kell tartani. Ezeket a szabályokat összefoglaló néven szintaxisnak (syntax) nevezünk.

# 

# 

# 

## Lefoglalt kulcsszavak (keywords)



## Adattípusok (data types)

Minden nyelvben, így a JAVA-ban is szabályozva van, hogy milyen típusú értékeket tárolhatunk a memóriában. Ezért ez egy típusos nyelv. Bő, részletes leírás [itt](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html).

### Elemi (primitív) adattípusok (primitive data types)

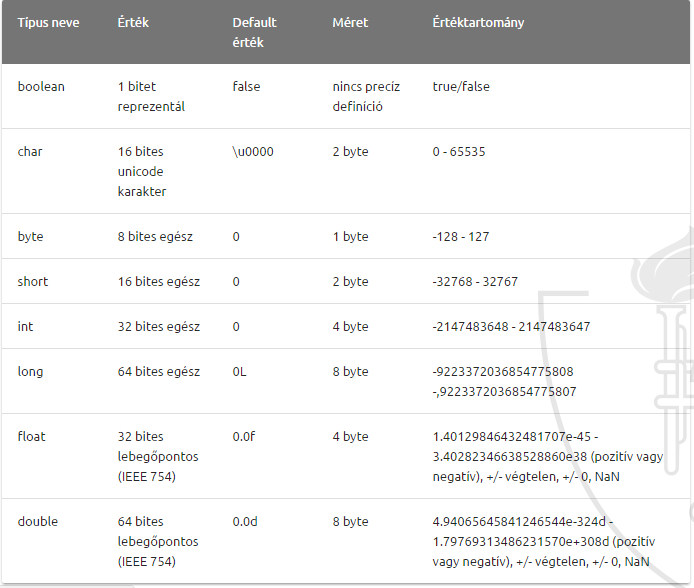
Ilyenek például az egész számok, tizedes számok, logikai változók (értékük igaz, vagy hamis lehet), karakterek és karakter láncok. Amikor java programot írunk kötelező, hogy minden változónak megadjuk, hogy milyen típusú, mert a későbbiekben a fordító ez alapján tudja eldönteni, hogy a program amit írtunk működő képes (szintaktikailag helyes).

**boolean** - logikai igaz/hamis - true/false tárolására

**char -** egy darab karakter tárolására. Pl.: ‘*P*’

**byte, short, int, long -** adott nagyságú, negatív vagy pozitív egész szám tárolására. Pl.: 15

**float, double -** adott nagyságú, negatív vagy pozitív **nem** egész számok tárolására. Pl.: -13.453



### Referencia adattípusok

**String** - adott hosszúságú karakterlánc tárolására. Pl.: “Progmatic”

**tömb, lista, map, osztály, interfész… Ezekkel a Java Collections részben fogjuk megismerni.**

## Osztály (Class)

Az objektum(ok) alapját egy osztály szolgáltatja.

| Osztalynev.java fájl: //fájlnév=osztálynév  class Osztalynev {  utasitas1;  ...  utasitasn; } |
| --- |

**Például**

| **class Learner {  String name;  LocalDate birthday; }** |
| --- |

## Objektum (Object)

Az objektum(oka)t osztályból hozunk létre azaz **példányosítunk/instanciate**. Példányosítás a **new** kulcsszó segítségével történik, ami egy Java operátor..

| new Learner(); |
| --- |

## Metódus

| visszateresiErtek metódusNev() {  utasitas1;  ...  utasitasn; } |
| --- |

**Például**

| int getAge() {  return 65;  } |
| --- |

### Speciális main metódus

**A program futáskori elindulási pontja**

| public static void main(String[] args) {  utasitas1;  …  utasítasn;  } |
| --- |

## Változó

Az objektum az állapotát változókban tárolja. A változó olyan adatelem, amely azonosítóval van ellátva.

Egy változó nevét és típusát egyértelműen meg kell adni a programunkban, ha használni akarjuk azt. A változó típusa meghatározza, hogy milyen értékeket vehet fel a változó, és milyen műveleteket hajthatunk végre rajta.

A változó rendelkezik hatáskörrel (érvényességi tartománnyal) is. A hatáskört a változó deklaráció helye egyértelműen meghatározza. *(lásd láthatóság majd a későbbiekben)*

A változó nevét és típusát a változó deklarációban adjuk meg, ami általában ehhez hasonló:

### Változó létrehozása

| változótípus változónév; |
| --- |

### 

### Változó létrehozása kezdőérték megadásával

| változótípus változónév=kezdőérték; |
| --- |

**Változó használata**

| változónév |
| --- |

### Példa int változó

| int evszam; |
| --- |

### Példa int változó létrehozása kezdőérték megadásával

| int evszam=2022; |
| --- |

**Példa int változó használata**

| évszam |
| --- |

### Példa String változó

| String nev; |
| --- |

### Példa String változó létrehozása kezdőérték megadásával

| String nev=”Sanyiii”; |
| --- |

**Példa String változó használata**

| nev |
| --- |

### Példa char változó

| char kedvencBetu; |
| --- |

### Példa char változó létrehozása kezdőérték megadásával

| char kedvencBetu=’C’; |
| --- |

**Példa char változó használata**

| kedvencBetu |
| --- |

### Példa boolean változó

| boolean szeretiACsokit; |
| --- |

### Példa boolean változó létrehozása kezdőérték megadásával

| boolean szeretiACsokit=true; |
| --- |

**Példa boolean változó használata**

| szeretiACsokit |
| --- |

### Példa változó használata

| int szuletesiEv=1990  System.out.print(“Született: ” + szuletesiEv) |
| --- |

## Operátorok

Természetes dolog, hogy egy programban számolni szeretnénk, esetleg logikai műveleteket végezni bizonyos változók értékeire vonatkozóan. A matematikai műveletekre vonatkozó operátorok már ismerősök lehetnek, viszont elég gyakran előfordulhat, hogy egy programban arra kell rákérdezni, hogy egy változó értéke igaz, vagy hamis, esetleg arra, hogy két változó értéke egyenlő, vagy nem egyenlő. Ebben a részben az előzőekben említett műveletek elvégzésére tanulhatod meg a java adta nyelvi elemeket.

**A főbb operátorok:**

* **=** egy változóhoz rendeli hozzá az értékét,
* **==** megvizsgálja, hogy két érték egyenlő-e, vagy hogy két változó ugyanarra az objektumra mutat-e a memóriában,
* **!=** ugyanaz, mint az előző, csak itt azt vizsgáljuk, hogy a két érték nem egyenlő,
* **&&** megvizsgálja, hogy két változó értéke igaz-e, ez a nyelvünkből is ismert "ÉS" feltétel (ha ez első érték hamis, akkor a másodikat már nem is vizsgálja meg a program, ezt nagyon fontos tudni),
* **||** megvizsgálja, hogy két változó értéke közül az egyik igaz-e, ez a nyelvünkből is ismert "VAGY" feltétel (ha ez első érték igaz, akkor a másodikat már nem is vizsgálja meg a program, ezt is nagyon fontos tudni),
* **+, -, \*, /**  összeadás, kivonás, szorzás, osztás (összeadás String-re is értelmezhető, ilyenkor a két karakterláncot összeköti),
* **++, --** egy változó értékét megnöveli/csökkenti eggyel,
* **+=, -=, \*=, /=** egy változó értékét megnöveli/csökkenti/szorozza/osztja a megadott értékkel.

| public class Operators {   public static void main(String[] args) {  int a = 2000;  int b = 31;   int c = a + b;  System.out.println(c);   int d = a - b;  System.out.println(d);   int e = a \* b;  System.out.println(e);   double f = 9.32;  double g = 1.93;   double h = f / g;  System.out.println(h);   boolean aIsEqualB = a == b;  System.out.println(aIsEqualB);   boolean aIsNotEqualB = a != b;  System.out.println(aIsNotEqualB);   boolean resultOfAnAndOperator = (a > 200) && (b < -200);  System.out.println(resultOfAnAndOperator);   boolean resultOfAnOrOperator = (a > 2000000) || (b > -200);  System.out.println(resultOfAnOrOperator);   System.out.println(7 % 2); // We call it modulo, this is the remaining part after division   int a2 = 5;  a2++;  System.out.println("a2 after ++ is " + a2);   int a3 = 7;  a3--;  System.out.println("a3 after -- is " + a3);   int a4 = 10;  a4 += 7;  System.out.println("a4 after += 7 is " + a4);   int a5 = 1;  a5 -= 7;  System.out.println("a5 after -= 7 is " + a5);   int a6 = 3;  a6 \*= 6;  System.out.println("a6 after \*= 6 is " + a6);   String hello = "Hello";  String world = " World!";  System.out.println(hello + world);  } } |
| --- |

## Elágazások (szelekció)

Néha van, hogy választási út előtt állunk és bizonyos feltételektől függően az egyik, vagy a másik irányba megyünk. Ugyanez a helyzet programoknál is előfordulhat, amikor egy megadott feltételt kell megvizsgálnunk, hogy a programunk melyik irányba folytatódjon, máshogy mondva hogyan viselkedjen.

Az ilyen típusú feladatok megoldására találták ki a HA, ÉS HA, EGYÉBKÉNT szerkezeteket. Ez az elem mindig egy igaz/hamis azaz boolean típusú értéket vár (például egy boolean változó értéke igaz, vagy hamis, egy szám kisebb, nagyobb, egyenlő, nem egyenlő, mint egy másik szám, esetleg bonyolultabb logikai kifejezések).

### Szintaxis - if

| if (feltétel){ //feltétel eredményének logikai igaz/hamis-t kell adjon  utasítás1;  ...  utasításn; } |
| --- |

**Például - HA** a kocsi ára kevesebb, mint 2 millió, akkor megveszem:

| int kocsiAr = 1900000; if (kocsiAr < 2000000){  System.out.print("Yeeee, vettem egy kocsit."); } |
| --- |

## 

### Szintaxis - if else

| if (feltétel){ //a feltétel logikai igaz/hamis   utasítás1;  ...  utasításn; }else{  utasítás1;  ...  utasításn; } |
| --- |

**Például - HA** a kocsi ára kevesebb, mint 2 millió, akkor megveszem **KÜLÖNBEN** 2 millióra alkudom:

| int kocsiAr = 2100000; if (kocsiAr < 2000000){  System.out.print("Megvettem"); }else{  System.out.print("2 millat adok erte"); } |
| --- |

### 

### 

### 

### 

### Szintaxis - if else if

| if (feltétel){  utasítás1;  ...  utasításn; }else if (feltétel){  utasítás1;  ...  utasításn; }else{  utasítás1;  ...  utasításn; } |
| --- |

**Például - HA** a kocsi ára kevesebb, mint 2 millió, akkor megveszem **ELLENBEN HA** 2-2,5 millió között akkor alkudom **KÜLÖNBEN** beszólok:

| int kocsiAr = 2100000; if (kocsiAr < 2000000){  System.out.print("Megvettem"); }else if (kocsiAr >= 2000000 && kocsiAr<=2500000){  System.out.print("2 millat adok"); }else{  System.out.print("Menj a sunyiba!"); } |
| --- |

Nézzük egy új példát, ahol egy random számot generálunk 0 és 2000 között és kiírjuk, hogy páros, vagy páratlan.

| import java.util.Random;  public class OddOrEven {   public static void main(String[] args) {  int maxNumber = 2000;  Random randomGenerator = new Random();  int randomNumber = randomGenerator.nextInt(maxNumber);   int remainder = randomNumber % 2;  if (remainder == 0) {  System.out.println(randomNumber + " is even!");  } else {  System.out.println(randomNumber + " is odd!");  }  } } |
| --- |

Készítsünk egy újabb programot, ami beolvas számot a felhasználótól és megmondja, hogy 2-vel 3-al vagy 4-el elosztva van-e maradék. Ha nem osztható egyikkel sem, akkor írja ki azt eredményként.

| import java.util.Scanner;  public class DivisibleByTwoOrThreeOrFour {   public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.println("Number:");  int number = Integer.parseInt(scanner.nextLine());  if (number % 2 == 0) {  System.out.println(number + " is divisible by 2");  } else if (number % 3 == 0) {  System.out.println(number + " is divisible by 3");  } else if (number % 5 == 0) {  System.out.println(number + " is divisible by 5");  } else {  System.out.println("Not divisible by 2 or 3 or 5");  }  } } |
| --- |

Figyeljük meg a harmadik esetet, amikor a 12-t vizsgáljuk meg. Ez a szám osztható 2-vel, 3-al és 4-el is, a programunk mégis csak annyit mond, hogy a 12 osztható 2-vel. Ez abból következik, hogy ha készítünk egy if/else if/else szerkezetet, akkor csak az első igaz ágban lévő kódrészlet fog lefutni, a többi nem.

### 

### Szintaxis - switch-case

| switch (kifejezes){ //kifejezes egy vált. ami egzakt értékeket vesz fel  case egakt\_ertek:   utasítás1;  ...  utasításn;  break; // break hatására kilép a switch-ből  case egakt\_ertek: //akármennyi case blokk lehet  utasítás1;  ...  utasításn;  break;  default: //opcionális  utasítás1;  ...  utasításn; } |
| --- |

**Például -** Különböző esetek megfogalmazására, mint **konkrét** lehetséges opciók. Nem feltételek, nem tartományok esetén használjuk elsősorban:

| int kocsiAr = 2100000; switch(kocsiAr){  case 2000000:  System.out.println("A kocsi ára pont 2 millió");  break;  case 2000001:  System.out.println("A kocsi ára pont 2 millió 1 Ft");  break;  default:  System.out.print("A kocsi ára valami más mint a fentiek"); }; |
| --- |

Megjegyzés: A break utasítás opcionális. Ha nem írjuk bele a case block-ba akkor ott nem lép ki a switch-ből és a következő case utasításait is végrehajtja. Ez első “talált” break utasításnál lép csak ki az adatszerkezetből, vagy ha végére ért az utasítások a switch-ben.

### Switch-case with lambda expression

| Lamdba expression //Csak a rövidítés funkció! switch (kifejezes) {  //Legfőbb ismertető jele a "->". Több utasítás esetén "{" zarójel kell!  case egzakt\_ertek -> {  utasítás1;  ...  utasításn; //nem kell break!  }  //használhatunk több konstans\_kifejezés-t is, ekkor "," kell közé és "()" köré.  case (egzakt\_ertek1, egzakt\_ertek2) -> {  utasítás1;  ...  utasításn;  }  // Ha kifejezésből és utasításból is csak egy van, akkor semmilyen zárójel nem szükséges  case egzakt\_ertek3 -> utasítás1;  default -> {  utasítás1;  ...  utasításn; } |
| --- |

## Ciklusok (iteráció)

### Szintaxis - while

Elöltesztelő ciklus, ami addig ismétlődik, amíg megadott a logikai kifejezés értéke igaz.

| while (feltétel) {  utastás1;  ...  utasításn; }; |
| --- |

**Példa** - elszámolok 1-20ig és kiírom a számokat a kimenetre:

| int szamom=1; while (szamom <= 20) {  System.out.print(szamom);  szamom++; //szamom=szamom+1 }; |
| --- |

### Szintaxis - do while

Hátultesztelő ciklus, ami addig ismétlődik, amíg megadott a logikai kifejezés értéke igaz.

| do {  utastás1;  ...  utasításn; } while (feltétel); |
| --- |

**Példa** - elszámolok 1-20ig és kiírom a számokat a kimenetre:

| int szamom=1; do{  System.out.print(szamom);  szamom++; //szamom=szamom+1 } while (szamom <= 20); |
| --- |

### 

### Szintaxis - for

| for (inicializálás; feltétel; növekmény) {  utastás1;  ...  utasításn; }; |
| --- |

**Példa** - elszámolok 1-20ig és kiírom a számokat a kimenetre:

| for (int szamom=1; szamom<=20;szamom++) {  System.out.println(szamom); }; |
| --- |

### Szintaxis - foreach

| for (datatype változó : objektum) {  utastás1;  ...  utasításn; }; |
| --- |

**Példa** - kiíratom a arrayNum tömböm értékeit a 0. indexű elemtől

| int[] numbers = {1,2,3,4,5,6,7,8,9,10};  for (int i = 0; i<= numbers; i++)  println("Count is: " + numbers[i]); }  // ugyanaz mint az alábbi for (int item : numbers)  println("Count is: " + item); } |
| --- |

# 

# 

# Értékek átadása a programnak

## Parancssori argumentumként

### egy darabot

| public static void main(String[] args){  System.out.println("Hello " + args[0]); } |
| --- |

### több darabot

| public static void main(String[] args){  System.out.println("Hello " + args[0] + " and " + args[1]); } |
| --- |

A main egy metódus, aminek egy argumentuma van, ez az args paraméter. Későbbiekben tanulunk a metódusokról, egyenlőre amit fontos megjegyezni, hogy minden paraméter, amit a java programnak átadunk az args paraméterben utazik és a programban használhatjuk.

## Konzolról futási időben

Előfordul, hogy az alkalmazásunkban a felhasználótól adatokat szeretnénk beolvasni, hogy aztán azokra reagálhassunk. Parancssori alkalmazásoknál a konzolról való beolvasáshoz a Scanner osztályt szoktuk használni.

| import java.util.Scanner;  public class ReadFromUserExample {   public static void main(String[] args) {  System.out.println("Hey!");  System.out.println("I am your bot.");  System.out.println("Who are you?");  Scanner scanner = new Scanner(System.in);  String name = scanner.nextLine();   System.out.println("Hello " + name + "! Good to see you!");  } } |
| --- |

Eredményként azt látjuk, hogy a program megkérdezi, hogyan hívnak minket, majd vár, amíg nem válaszolunk. Miután válaszoltunk a **scanner.nextLine()** metódus visszatér a beírt szöveggel. Az így kapott névvel a programunk köszön a felhasználónak.

Sok olyan rész van még persze a programban, amit nem érthetünk teljesen, például a **new** kulcsszó, a módszer, ahogy létrehoztunk egy Scanner-t. Vagy az 1. sorban az **import**…Ezeket a részeket egyenlőre, úgy ahogy vannak jegyezzük meg és használjuk, a későbbiekben megtanuljuk mit jelentenek ezek a még ködös foltok is.

# String műveletek

A java egyik alap típusa a String, ami karakterláncok tárolására szolgál. A String nem egy primitív típus, mint a boolean, int, double és a többi. A String egy összetett típus, egy osztály.

A belső működése szempontjából egy karakter tömböt tárol (char[]), amiben az idézőjelek között megadott karaktereket helyezi el, például String name = "Bob";. Gyakori feladat, hogy az ilyen összetett karakterláncokon műveleteket kell végeznünk. Ilyen lehet például, hogy:

* az összes betűt nagybetűre/kisbetűre akarjuk állítani,
* ki akarunk cserélni betűket, kifejezéseket,
* le akarjuk vágni a szóközöket a szöveg elejéről és végéről,
* szét akarjuk darabolni a szöveget bizonyos karakterek mentén,
* ...

A String osztály számos metódust biztosít, amik a fentebbi műveleteket lehetővé teszik.

**Néhány fontosabb metódus:**

* charAt(index): visszaadja a megadott helyen lévő karaktert (figyeljünk rá, hogy a 0 index jelenti az első karaktert),
* length(): a karakterlánc hosszát adja vissza,
* trim(): levágja a szóközöket a karakterlánc elejéről és végéről,
* toCharArray(): visszaadja a karakterláncot char[] formában,
* toLowerCase(): visszaad egy új String-et, amiben minden betű kisbetűre van konvertálva, pl.: "ABCD" ➡ "abcd"
* toUpperCase(): visszaad egy új String-et, amiben minden betű nagybetűre van konvertálva, pl.: "abcd" ➡ "ABCD",
* equals(other): megmondja, hogy két String egyenlő-e egymással, például.: "ABCD".equals("ABCD") ➡ true(nagyon fontos, hogy amikor leellenőrizzük, hogy két String megegyezik-e, akkor ezt használjuk és ne a ==kifejezést),
* substring(startIndex, endIndex): visszaad egy másik String-et, ami a startIndex és endIndex közötti részt tartalmaz (a startIndex helyen állót tartalmazni fogja, az endIndex helyen lévőt nem), például.: "cat".substring(0, 1) ➡ "c"
* startsWith(expression): megmondja, hogy a String a megadott kifejezéssel kezdődik-e,
* endWith(expression): megmondja, hogy a String a megadott kifejezéssel végződik-e,
* contains(expression): megmondja, hogy a String a megadott kifejezést tartalmazza-e,
* replace(target, replacement): kicseréli az összes előfordulását a target argumentumnak a replacement-re,
* split(regex): darabokra vágja a karakterláncot a megadott kifejezés szerint, például.: "a,b,c".split(",") ➡ ["a", "b", "c"]

A String osztályra vonatkozóan nagyon fontos megjegyezni, hogy bár azt hihetjük, hogy néhány metódusa megváltoztathatja a tartalmát, de valójában nem.

**Onnantól kezdve, hogy létrehoztunk egy String változót a tartalma soha nem változik meg.** Vannak olyan metódusok, mint például a replace(), toLowerCase(), amik visszaadnak egy másik String-et, de az eredetileg létrehozott String soha nem változik meg.

Nézzünk erre egy példát.

| public static void main(String[] args) {  String bob = "Bob";  System.out.println(bob);  bob.toLowerCase();  System.out.println(bob);   String adam = "Adam";  System.out.println(adam);  adam = adam.toLowerCase();  System.out.println(adam); } |
| --- |

Bob nem lett kisbetűs, mert az eredeti változó értékét nem tették egyenlővé a bob.toLowerCase() metódushívás által visszaadott értékkel.

## String konvertálása más típusokká

Előfordul, hogy egy String típusú adatot át kell konvertálni egész, esetleg tizedes számmá, esetleg boolean értékké. Ilyen előfordulhat, amikor fájlokból olvasunk fel adatokat esetleg a felhasználó által megadott adatokat akarjuk átalakítani egy adott típusra. Ilyen esetekre minden adattípusra biztosít számunkra a java egy parse() metódust, amivel elvégezhetjük a konvertálást.

| public class PrimitiveTypes {   public static void main(String[] args) {  // Previous examples   boolean myBoolean = Boolean.parseBoolean("false");  int myInteger = Integer.parseInt("12345899");  double myPi = Double.parseDouble("3.14");   System.out.println(myBoolean);  System.out.println(myInteger);  System.out.println(myPi);  } } |
| --- |

# Véletlenszerű (random) adatok generálása

## Számok

Nézzük egy példát, ahol egy random számot generálunk 0 és 5 között és kiírjuk.

| Random randomGenerator = new Random(); int randomNumber = randomGenerator.nextInt(maxNumber); |
| --- |

## Boolean

| Random random = new Random(); String result = random.nextBoolean(); |
| --- |

# String-ek összehasonlítása

A String-ek bár sokszor primitív adattípusnak emlegetik, valójában nem az és mint nem primitív adattípus beli adattípus, így az ott használt operátorok sem használhatók két String összehasonlítására. **==** helyett, ha két String típusú változó értékét szeretnénk összehasonlítani, mindig használjunk **.equals()** metódust az alábbi formában:

| String mithasonlitok; String mivelhasonlítom; boolean egyenloe = mithasonlitok.equals(mivelhasonlitom); |
| --- |

**Példa** - **HA** a felhasználó neve “admin”, akkor írjuk ki a képernyőre, hogy administrátor **EGYÉBKÉNT** írjuk ki a képernyőre, hogy felhasználó.

| Scanner sc = new Scanner(System.in); String mithasonlitok = sc.nextLine(); String mivelhasonlitok = "admin";  if (mithasonlitok.equals(mivelhasonlitok)) {  System.out.println("administrator"); } else {  System.out.println("felhasznalo"); } |
| --- |

# Tömbök (Arrays)

Lehetőség van elemeket csoportosítani és úgy tárolni, rendezni illetve keresni. Ezek az együtt tárolt elemek egyező adattípussal rendelkeznek!

Az array sorszám alapú tárolással rendelkezik. Az első elem tárolódik a 0. indexen, a második elem az 1. indexen és így tovább.

**Előnye**

* adatok véletlenszerű elkérése a tömbtől, bármelyik elemet elkérhetjük
* költséghatékonyabb mint megannyi változót létrehozni és használni

**Hátránya**

* méretkorlát, a létrehozáskor definiált méretet utána nem tudjuk módosítani futásidőben

## Létrehozás

Létrehoz egy **int** tömböt, aminek a mérete **10**, azaz maximum 10 elemet tehetünk bele

| int[] numbers = new int[10] |
| --- |

Létrehoz egy **String** tömböt, aminek a mérete **20**, azaz maximum 20 elemet tehetünk bele

| String[] names = new String[20] |
| --- |

Létrehoz egy **boolean**  tömböt, aminek a mérete **7**, azaz maximum 7 elemet tehetünk bele

| boolean[] flags = new boolean[7] |
| --- |

Létrehozhatjuk őket már eleve az elemek megadásával is és akkor nem kell a méretet külön beállítani. **Például**:

| int[] numbersWithValues = {1, 34, 2, 21}; |
| --- |

| String[] namesWithValues = {"Zsolt", "Csaba"}; |
| --- |

## Értékadás

Létrehozást követően a következő módon tudjuk beállítani az értékeket, mégpedig az adott index elem értékadásával

| arrayVariable[arrayIndex] = value; |
| --- |

**Például**

| **numbers[2] = 56;** |
| --- |

## Érték kiolvasása

| változónév[tömbindex] |
| --- |

**Például** az alábbi utasítás a*numbersWithValues*tömb változó első elemét adja vissza *(hisz az indexelés 0-tól indul)*

| numbersWithValues[0] |
| --- |

A tömb méretét mindig a **length** attribútumával érhetjük el. Például:

| int arraySize = numbersWithValues.length |
| --- |

## Array vs ciklusok

Több érték beállítása ciklus segítségével

| int[] numbers = new int[20]; for(int i = 0; i < numbers.length; i++) {  numbers[i] = i; } |
| --- |

Több érték kiolvasása ciklus segítségével

| for(int i = 0; i < numbers.length; i++) {  System.out.println(numbers[i]); } |
| --- |

# Collections

A JAVA biztosít szükséges architektúrát az objektumok csoportját tárolására és manipulálására. Az adatokon elvégezhető összes művelet elérhető, úgy mint a keresés, rendezés, beszúrás, manipuláció vagy törlés.

## Listák (Lists)

Olyan elemek csoportosítása, amely egy rendezett gyűjtemény. Ahogy a tömb, ez is engedi ugyanazon elem többszöri szerepeltetését.

**Előnye**

* lehet ugyanazon elem többször
* nincs mérletkorlát, létrehozását követően hozzá adhatunk és eltávolíthatunk elemeket, akár futás időben.

**Hátránya**

* több memóriát használ mint a tömb
* lassabb mint a tömb

**Fontosabb metódusai:**

* **add(element)** - hozzáad a listához egy új elemet
* **get(index)** - visszaadja az adott indexű elemet
* **size():** visszaadja, hogy hány elem van aktuálisan a listában
* **isEmpty():** megmondja, hogy a lista üres-e, vagy sem
* **remove(element):** kitörli az adott indexű elemet
* **contains(element):** megnézi, hogy az adott elem benne van-e a listában

### Létrehozás

**Szintaxis**

| List<adattípus> változónév = new ArrayList<>(); |
| --- |

**Példa**:

| List<String> randomNames = new ArrayList<>(); List<Integer> randomNums = new ArrayList<>(); List<Double> randomDoubleNums = new ArrayList<>(); List<Boolean> randomBooleans = new ArrayList<>(); |
| --- |

A boolean, int, double ... primitív típusok helyett a burkoló osztályaik neveit kell ilyenkor használnunk, mint Boolean, Integer, Double. Lista implementációként az ArrayList-et használtuk, ami egy speciális típusú lista. Vannak más opciók is az ArrayList helyett, de a tananyagban csak erről lesz szó.

### Értékadás

A fent látott **add()** metódus használatával lehetséges:

| randomNames.add("Viktor"); randomNums.add(5); randomDoubleNums.add(2.1); randomBooleans.add(false); |
| --- |

Lehetőség van úgy is létrehozni Listát, hogy beállítjuk a létrehozás pillanatában a kezdőértékét, vagyis az elemeit ami van benne:

| List<String> names = Arrays.asList("Viktor", "József", "Benedek"); List<Integer> names = Arrays.asList(1990,1987,2002,1992); |
| --- |

### Kiolvasás

A fent látott **get()** metódus használatával lehetséges:

| randomNames.get(1); randomNums.get(1); randomDoubleNums.get(1); randomBooleans.get(1); |
| --- |

Vagy ha az egész listára vagyunk kíváncsiak akkor egyszerűen elég csak a változó nevére hivatkozni:

| System.out.println(randomNums); |
| --- |

### Egyéb

| randomNums.size(); // eredménye egy egész szám lesz randomNums.isEmpty(); // eredménye true vagy false lesz  randomNums.remove(1); // végrehajtja randomNums.contains(15); // eredménye true vagy false lesz |
| --- |

### Példa

Tegyük fel, hogy számokat akarunk tárolni 100-111 között egy *numsList* nevű Listában

| List<Integer> numsList = new ArrayList<>(); for(int i = 100; i <= 111; i++) {  numsList.add(i); } System.out.println(numsList); |
| --- |

## 

## Halmazok (Set)

Set-eket akkor szoktuk használni, ha csak olyan elemeket akarunk tárolni, amik egyediek. Tegyük fel például, hogy random számokat sorsolunk ki 1-100 között és egy listába rakjuk őket. Előfordulhat, hogy egy számot kétszer is kisorsolunk, így az gond nélkül belekerül a listába mind a két alkalommal. Amennyiben set-et használunk, ez nem fordulhat elő.

Egy set-be ha kétszer belerakjuk ugyanazt az elemet, akkor is csak egyszer fogja tartalmazni. A set ezt úgy csinálja, hogy belső működése szempontjából mindig megnézi, hogy az adott elemet tartalmazza-e már és ha igen, akkor egyszerűen nem foglalkozik vele.

**Előnye**

* egy elem csak egyszer szerepelhet benne
* dinamikusan tud nőni a mérete, nem tud túlcsordulni
* gyorsan lehet keresni benne

**Hátránya**

* nem sorrendben rendezett, nem nézi melyik elem mikor került bele. Így egy konkrét indexű elemet nem is tudunk elkérni tőle. (nincs get() metódus)
* lassú az új elem hozzáadásában és törlésében

**Fontosabb metódusai:**

* **add(element)** - hozzáad a halmazhoz egy új elemet
* **size():** visszaadja, hogy hány elem van aktuálisan a listában
* **isEmpty():** megmondja, hogy a lista üres-e, vagy sem
* **remove(element):** kitörli az adott elemet
* **contains(element):** megnézi, hogy az adott elem benne van-e a halmaz

### Létrehozás

**Szintaxis**

| Set<adattípus> változónév = new HashSet<>(); |
| --- |

**Példa**:

| Set<String> randomNames = new HashSet<>(); Set<Integer> randomNums = new HashSet<>(); Set<Double> randomDoubleNums = new HashSet<>(); Set<Boolean> randomBooleans = new HashSet<>(); |
| --- |

A boolean, int, double ... primitív típusok helyett a burkoló osztályaik neveit kell ilyenkor használnunk, mint Boolean, Integer, Double. Set implementációként az HashSet-et használtuk, ami egy speciális típusú lista. Vannak más opciók is az HashSet helyett, de a tananyagban csak erről lesz szó.

### Értékadás

A fent látott **add()** metódus használatával lehetséges:

| randomNames.add("Viktor"); randomNums.add(5); randomDoubleNums.add(2.1); randomBooleans.add(false); |
| --- |

### Kiolvasás

Mivel itt nincs **get()** metódus, így csak egész set-et tudjuk a változó nevének használatával:

| System.out.println(randomNames); |
| --- |

### Egyéb

| randomNums.size(); // eredménye egy egész szám lesz randomNums.isEmpty(); // eredménye true vagy false lesz  randomNums.remove(15); // végrehajtja randomNums.contains(15); // eredménye true vagy false lesz |
| --- |

### Példa

Nézzünk egy példát, ahol random számokat sorsolunk ki és belerakjuk egy set-be. Ahhoz, hogy lássuk egy listában előfordulhat, hogy egy elemet kétszer is belerakunk, így párhuzamosan egy listát is feltöltünk a random elemekkel.

| Set<Integer> numbers = new HashSet<>(); List<Integer> numbersList = new ArrayList<>(); Random random = new Random(); for(int i = 0; i < 20; i++) {  int number = random.nextInt(15);  numbers.add(number);  numbersList.add(number); } System.out.println("Set:"); System.out.println(numbers); System.out.println("List:"); System.out.println(numbersList); |
| --- |

Eredménye:

| Set: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12] List: [12, 0, 6, 10, 4, 5, 0, 10, 5, 4, 2, 1, 2, 9, 9, 7, 2, 8, 3, 6] |
| --- |

Ha megnézzük az eredményt, azt látjuk, hogy a set-be a random számok rendezésre kerültek növekvő sorrendbe, míg a lista olyan sorrendben tartalmazza őket, ahogy beleraktuk őket. Általában igaz, ha egy HashSet-be elemeket pakolunk, akkor a sorrend, ahogyan azokat visszakapjuk nem meghatározott, ez csakis a belső implementációtól függ.

## Map

A map-ek a java-ban egy olyan adatszerkezetet jelölnek, ami kulcs-érték (key-value) párokat tárol. Ez annyit jelent, ha például Bobhoz szeretném hozzárendelni a telefonszámát, akkor a neve által mindig gyorsan vissza tudom majd kapni azt.

Kulcs csak egyszer szerepelhet a Map-ben viszont több ugyanolyan érték is feltűnhet.

**Előnye**

* különböző adattípusú elemeket is rakhatunk bele
* gyorsan tudunk kiolvasni belőle

**Hátránya**

* a kulcsot nekünk kell biztosítani
* több memóriát használ mint a többi Collection elem

**Fontosabb metódusai:**

* **put(key, value)** - hozzárendeli az adott kulcshoz az adott elemet, ha már benne van az adott kulcs, akkor felülírja
* **remove(key) -** kiszedi az adott kulcsot és a hozzátartozó értéket a map-ből
* **containsKey(key) -** megnézi, hogy a map tartalmazza-e az adott kulcsot
* **containsValue(value) -** megnézi, hogy a map tartalmazza-e az adott értéket
* **isEmpty() -** megmondja, hogy a map üres-e, vagy sem
* **size() -** visszaadja, hogy hány elem van a map-ben
* **keySet() -** visszaadja a map-ben található kulcsokat, mint egy set-et
* **values() -** visszaadja a map-ben található értékeket
* **entrySet() -** visszaadja a map-ben található összetartozó kulcs-érték párokat

### Létrehozás

**Szintaxis**

| Map<adattípus, adattípus> változónév = new HashMap<>(); |
| --- |

**Példa**:

| Map<String, String> randomNames = new HashMap<>(); Map<Integer, String> randomNames2 = new HashMap<>(); Map<String, Integer> randomNames3 = new HashMap<>(); Map<Integer, Boolean> randomBooleans = new HashMap<>(); |
| --- |

A boolean, int, double ... primitív típusok helyett a burkoló osztályaik neveit kell ilyenkor használnunk, mint Boolean, Integer, Double. Map implementációként az HashMap-et használtuk, ami egy speciális típusú lista. Vannak más opciók is az HashMap helyett, de a tananyagban csak erről lesz szó.

### Értékadás

A fent látott **put()** metódus használatával lehetséges:

| randomNames.put("BDA3435","Viktor"); randomNames2.put(223553, "Mekk Elek"); randomNames3.put("Viktor", 06807777777); randomBooleans.put(210, false); |
| --- |

### Kiolvasás

Mivel itt nincs **get()** metódus, így csak egész set-et tudjuk a változó nevének használatával:

| System.out.println(randomNames); |
| --- |

Foreach segítségével is be tudjuk járni a map-et, viszont a HashMap sajátossága miatt kissé eltér az eddigiektől:

| for (Map.Entry mapElement : mapVariable.entrySet()) {  System.out.println(mapElement.getKey() + " " + mapElement.getValue()); } |
| --- |

Az elemek kiíratása nem a hozzáadás sorrendjében történik, mivel nem indexelt adatszerkezet mint a List vagy Array.

### Egyéb

| randomNums.size(); // eredménye egy egész szám lesz randomNums.isEmpty(); // eredménye true vagy false lesz  randomNums.remove(223553); // végrehajtja, key mentén töröl randomNums.containsKey(223553); // eredménye true vagy false lesz randomNums.containsValue("Mekk Elek"); // eredménye true vagy false lesz |
| --- |

### Példa

Tegyük fel, hogy országokat és azok fővárosait szeretnénk tárolni. Erre kiválóan alkalmas a Map.

| HashMap<String, String> capitalCities = new HashMap<>(); // Add keys and values (Country, City) capitalCities.put("England", "London"); capitalCities.put("Germany", "Berlin"); capitalCities.put("Norway", "Oslo"); capitalCities.put("USA", "Washington DC"); System.out.println(capitalCities); |
| --- |

# Metódusok

A Java készítői rengeteg dolgot megírtak helyettünk, amelyeket kezdő programozóként sokszor úgy használunk, hogy nem is tudatosul. Ilyenek voltak az elsők között megtanult *System.out.println()* vagy például a listák esetén az *add()*.

A metódusokat tekinthetjük egyfajta üzeneteknek, melyek valamilyen feladat megoldását kérik valakitől. Az üzenet 3 részből állhat.

1. Címzett objektum vagy osztály
2. A címzett metódusának neve
3. Esetleges paraméterek

Az üzenet címzettjét abban az esetben kötelező megadni, ha a címzett nem önmaga. A programozásunk elején, amíg nem készítünk saját objektumokat, javarészt így fogjuk használni a metódusokat, hogy megadjuk a címzettet.

A metódus nevét azt kötelező megadni természetesen.

Mivel a metódusok valamilyen feladat végrehajtását kérik, ezért a feladattal kapcsolatos adatokat is oda lehet adni annak a metódusnak, amelyik azt elvégzi. Természetesen lehet olyan metódus is, ami nem használ bemenő adatokat.

| Integer.parseInt("123"); System.out.println("Ez egy rovid mondat."); luckyNumberList.add(21); userList.isEmpty(); nameSet.size(); |
| --- |

A metódusok két csoportját különböztetjük meg

* valamilyen értéket állítanak elő és adják oda az őt hívónak
* valamilyen tevékenységeket hajtanak végre

Az első csoport nagyjából olyan, mint egy táblázatkezelő program függvényei. Adunk neki cellákat és ad nekünk vissza egy eredményt. Ez az eredmény a **visszatérési érték**. A bemenő adatokat, pedig **paraméter**-eknek nevezzük.

A második csoport pedig gyakorlatilag végrehajtja a kívánt utasításokat és kész. Például egy metódus, amiben egy sout szerepel csupán.

Miért kell osztály vagy változónév? Mert létezhet több ugyanolyan nevű metódus is az alkalmazásunkban, viszont így könnyen beazonosítható.

## Jójó de mire jó ez nekem?

Nos elsősorban az ismétlődések kiiktatásához nagyszerű eszköz a metódus.

Például ha van egy több soros kód, ami a Map-et egy foreach-el bejárja és formázottan kiírja az elemeit, akkor ezt miért ne használhatnám fel másra is, egy másik Map-hez? Készíthetünk egy metódus aminek ha odaadjuk a létrehozott és feltöltött map változót, akkor ő bármelyiket fel tudja nekünk dolgozni.

## Példák létrehozásra

Szeretnénk egy metódust készíteni, ami kirajzol egy — jelekből álló vonalat az egyes feladatok elválasztására

| public static void vonal(){  System.out.println("-----"); } |
| --- |

Szeretnénk egy metódust készíteni, mely egy kapott tömb elemeit kiírja a képernyőre

| public static void kiir( int[] tomb ){  for( int i = 0; i < tomb.length; i++ )  {  System.out.println(tomb[i]+" ");  }  System.out.println(); } |
| --- |

Szeretnék egy metódust, amely ad egy random számot 0-100 között

| public static int getRandomNumber(){  Random randomGenerator = new Random();  int number1 = randomGenerator.nextInt(100);  return number1; } |
| --- |

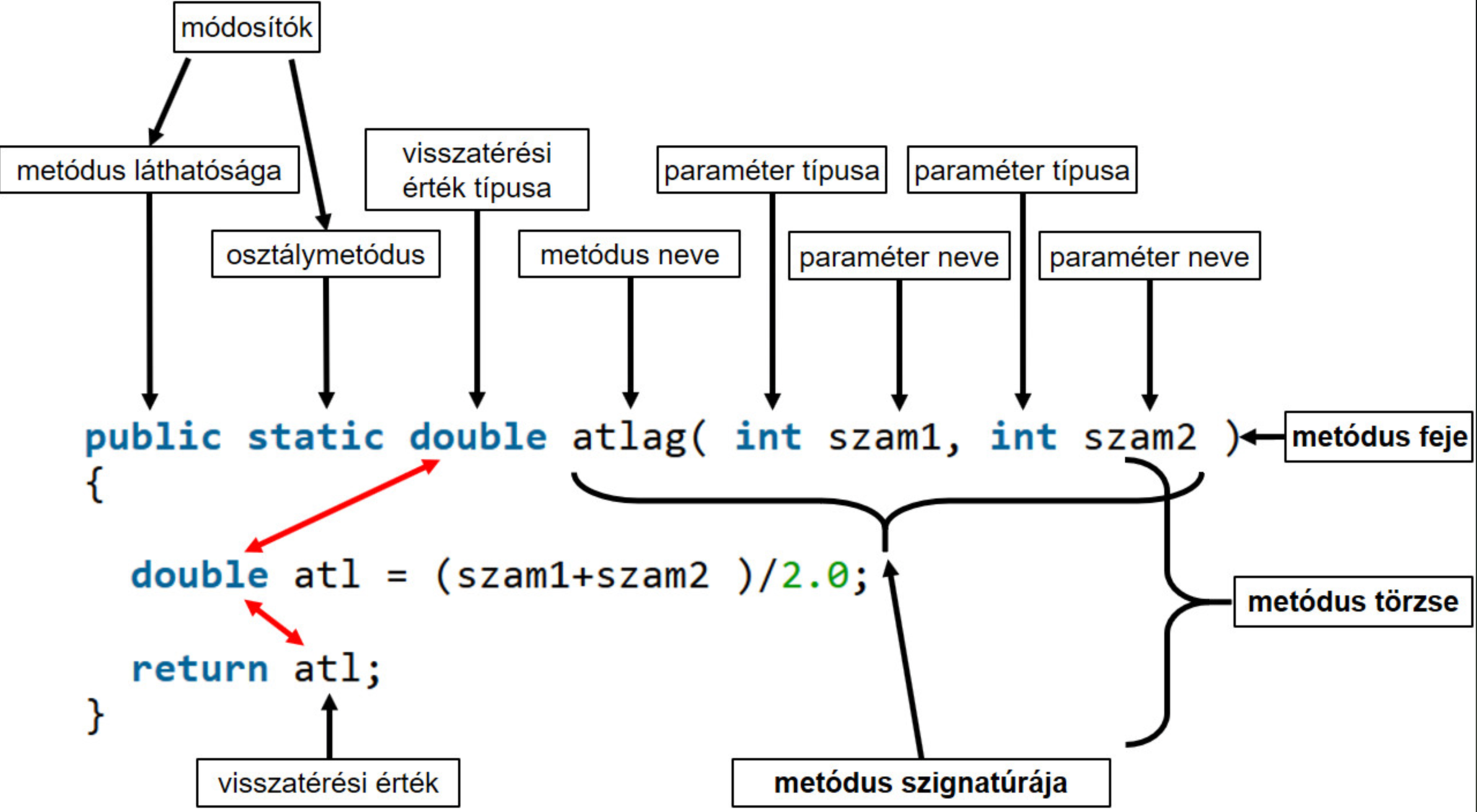
Szeretnénk egy metódust készíteni, amely két kapott szám átlagát adja vissza eredményként.

| public static double atlag( int szam1, int szam2 ){  double atl = (szam1+szam2 )/2.0;  return atl; } |
| --- |

## Egyéb példák:

| public static void hello(String name) {  System.out.println("Hello " + name); }  public static int sum(int a, int b) {  return a + b; }  public static double duplicate(double n) {  return 2 \* n; }  public void printSquares(List<Double> numbers) {  for (double number : numbers) {  System.out.println(number \* number);  } } |
| --- |

## Metódusok részei



A láthatóság, static és a paraméterek opcionálisak. A metódus neve csakúgy mint a változók, kis kezdőbetűs.

## Példák metódusok használatára, hívására

Ki szeretnék íratni egy 5 egység hosszú vonalat – jelekből a megírt metódus használatával:

| vonal(); |
| --- |

A metódusokat többször, akár egymás után is meghívhatjuk:

| vonal();  vonal();  vonal(); |
| --- |

Ki szeretném írni egy tömb elemeit a képernyőre egymás mellé szóközökkel elválasztva:

| kiir(tomb); |
| --- |

Szeretnék egy véletlen számot:

| int szam = getRandomNumber(); // sorsolom és eltárolom System.out.println( getRandomNumber() ); // sorsolom és kiírom |
| --- |

Ki szeretném számolni két szám átlagát:

| int szam = 3; double atlag = atlag( 4, szam ); |
| --- |

A példákban gyakorlatilag csak a szignatúra szerepel. Meg kell adni a metódus nevét, és oda kell neki adni a működéséhez szükséges adatokat direkt értékek, változók vagy kifejezések formájában.

Amikor a metódusnak nincs visszatérési értéke (**void** típusú), akkor a metódus hívás egyetlen utasításként **önmagában** áll.

Amikor azonban van **visszatérési érték**, azzal kezdeni kell valamit, különben nem lenne értelme az eredménynek. **Eltároljuk egy változóban**, közvetlenül kiírjuk a metódus eredményét a képernyőre, stb.

## Metódus túlterhelés (method overloading)

Előfordulhat, hogy egy adott metódus nevet többször is fel szeretnénk használni, például az *sum()* metódust szeretnénk ha működne tizedes szám bemenetre, de akár egy listányi *String*-re is.

Ilyen esetekben használatjuk a metódus túlterhelést, ami azt jelenti, hogy ugyanolyan nevű metódust készítünk, csak eltérő a bemeneti paraméter(ek) típusa és/vagy darabszáma, valamint a visszatérési értéknek a típusa.

| private static int sum(int a, int b) {  return a + b; }  private static double sum(double a, double b) {  return a + b; }  private static double sum(List<String> numbers) {  double sum = 0;  for (String number : numbers) {  sum += Double.parseDouble(number);  } return sum;  } |
| --- |

# Kivételek (Exceptions)

## Mi is az a hibakezelés?

Bármennyire is vagyunk jó programozók, sajnos előfordul, amikor valamilyen külső hatás eredményeképpen a programunk nem tudja ellátni a feladatát. Ilyen külső hatás lehet például, ha az alkalmazásunk interneten keresztül szeretne küldeni egy levelet, viszont nincsen kapcsolat, vagy a felhasználó szeretné, ha egy fájl tartalmát megjelenítenénk, de az általa megadott fájl nem létezik. Ilyen és ehhez hasonló esetekben hasznos, ha ismerjük a java adta hibakezelési, vagy másnéven kivételkezelési lehetőségeket.

Vannak olyan hibák, amikre képesek vagyunk felkészíteni a programot, így az magától is le tudja azt kezelni, viszont vannak olyan esetek, amikor a program külső segítség nélkül nem képes megoldást találni (az előbb említett internet hiánya).

## **try - catch** - a bekövetkezés helyén elkapjuk

Tegyük fel, hogy elkészítettünk egy tömböt, de olyan indexű elemre hivatkozunk, ami nem létezik a tömbben.

| int[] exampleArray = {21, 44, 33, 77}; System.out.println(exampleArray[15]); System.out.println("Thanks for using our service!"); |
| --- |

Mi történik, ha lefuttatjuk?

| Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 15 out of bounds for length 4  at Playground.main(Playground.java:6) |
| --- |

Mi lenne, ha gyanakvóbbak lennénk, mintsem majdan felhasználóra bíznánk hogy jó indexű elemre hivatkozik és felkészülnénk előre erre a lehetőségre hogy a tömböt túlindexeljük…

**Szintaxis**

| try {  utasitas1;  utasitas2;  ...  utasitasn; } catch (ExceptionType exceptionvariable) {  utasitas1;  utasitas2;  ...  utasitasn; } |
| --- |

**A példa**

| try {  int[] exampleArray = {21, 44, 33, 77};  System.out.println(exampleArray[15]); } catch (**ArrayIndexOutOfBoundsException** exception) {  System.out.println("Hey, there is no such an index in the array!"); }  System.out.println("Thanks for using our service!"); |
| --- |

## **throws** - a bekövetkezés helyéről tovább dobjuk

Amennyiben az adott metódusban vagy osztályban nem szeretnénk lekezelni a hibát, úgy lehetőségünk van azt tovább dobni. Ilyen esetben annak a másik metódusnak, osztálynak kell lekezelnie, aki a mi metódusunkat, osztályunkat hívja. Ahogy látható ilyenkor a metódus szignatúrájába írt throws IOException-el jelezhetjük, hogy ez a metódus esetleg hibát okozhat a programban és erre egy másik osztályban gondolni kell.

**Szintaxis**

| method() throws ExceptionType {  utasitas1;  utasitas2 ;  ...  utasitasn; } |
| --- |

**A példa**

| public static void arrayMethod() throws ArrayIndexOutOfBoundsException {  int[] exampleArray = {21, 44, 33, 77};  System.out.println(exampleArray[15]); }  public static void main(String[] args) {  try {  arrayMethod();  } catch (ArrayIndexOutOfBoundsException ex) {  System.out.println("Hey, there is no such an index in the array!");  }  System.out.println("Thanks for using our service!"); } |
| --- |

# Java IO

Fejlesztés során gyakran előforduló feladat, hogy adatokat kell beolvasnunk fájlokból, esetleg az alkalmazás által kezelt adatokat kell kiírni egy fájlba. Elképzelhető, hogy néha a fájlrendszerben kell megkeresni fájlokat, esetleg tömörített fájlokat készíteni. A java nyelv széles választékot biztosít a fájlokon végzett műveletekre, ezzel fogsz megismerkedni ebben a blokkban

## Fájlba írás

File-ba írásnál használhatjuk a Files.write(path, bytes) metódust. Ebben az esetben meg kell adnunk egy Path objektumot, valamit a tartalmat, amit a fájlba szeretnénk írni. A fájl tartalmát különböző karakterkódolás szerint tárolhatjuk. Az egyik leggyakrabban használt karakterkódolás az UTF-8, amit a lenti példában is használunk.

| import java.io.BufferedReader; import java.io.IOException; import java.nio.file.Files; import java.nio.file.Paths; import java.util.List;  public class FileWriting {   public static void main(String[] args) {  try {  String content = "Hello." + System.lineSeparator() + "This is great. " + System.lineSeparator() + "I could create a file from java.";  Files.write(Paths.get("src/resources/my-new-file.txt"), content.getBytes("UTF-8"));  } catch (IOException e) {  throw new RuntimeException(e);  }  } } |
| --- |

## Fájlból olvasás

File-okkal való műveletek végzésére leggyakrabban a [Files](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html) és [Paths](https://docs.oracle.com/javase/8/docs/api/java/nio/file/Paths.html) osztályokat használjuk. A Files osztály tartalmaz sok olyan metódust, amik mind az írásnál, mind az olvasásánál hasznosak. A Paths osztály segítségével fájlrendszer elérési útvonalakat írhatunk le könnyen.

A lenti programunkban training adatokat szeretnénk beolvasni. Képzeljük el, hogy ez a fájl tartalmazza az edzéseink adatait, esetleg mások edzéseinek adatait. Miután beolvastuk akár készíthetnénk értékes statisztikát és adhatnánk valamilyen visszajelzést arra vonatkozóan, hogy hogyan teljesített a felhasználó az edzéseken.

**Fontos** Ahhoz, hogy a lenti példa működjön készítsünk egy fájlt a resources mappában trainings.csv névvel, az alábbi tartalommal:

| Training Type, Duration [min] Slow long running, 120 Strength, 45 High intensity training, 50 Slow long running, 210 |
| --- |

| import java.io.BufferedReader; import java.io.IOException; import java.nio.file.Files; import java.nio.file.Paths; import java.util.List;  public class FileReading {   public static void main(String[] args) {  try {  List<String> lines = Files.readAllLines(Paths.get("src/resources/trainings.csv"));  for (String line : lines) {  System.out.println(line);  }  } catch (IOException e) {  throw new RuntimeException(e);  }  } } |
| --- |

Ahogy látható a speciális osztályok amiket használtunk a Files, aminek át kellett adni egy a Paths osztály segítségével készített Path objektumot. Ez gyakorlatilag az elérési útvonalat tartalmazó java objektum, tartalmazhatja például a "C:\\Users\\MyUser\\my-file.txt", "OtherUser\\my-file.txt" elérési útvonalakat.

A beolvasás után (Files.readAllLines() metódus hívás) az összes sort megkapjuk egy listában. Ha bármilyen hibával szembesülünk, például nem létezik a fájl, vagy valaki más használja, akkor egy IOException-t kapunk. Ez egy olyan hiba amit le kell kezelnünk, viszont mivel ez általában elég komoly hibát szokott jelenteni, így a programot külső beavatkozás nélkül nem nagyon lehet segíteni, így tovább forgatjuk egy RuntimeException-be, amit majd a felhasználó lát.

A program eredményeként azt látjuk, hogy a konzolban megjelenik a fájl tartalma.

# Date/Time

Az idő mérése, a dátumok kezelése minden programnak nagyon fontos eleme. A java régebbi verzióiban az idő kezelése nehézkes volt, a rendelkezésre álló osztályok kezdetlegesek voltak, körülményes volt a használatuk. A java 8-as verziótól jelentősen javítottak ezen és megjelent a [Time API](https://docs.oracle.com/javase/tutorial/datetime/iso/datetime.html). A megjelent új osztályok segítségével könnyebbé vált a műveletek végzése dátumokkal, időzónák közötti váltás.

## 

## LocalDate

A [LocalDate](https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html) osztály segítségével dátumok leírására szolgáló objektumokat hozhatunk létre. A LocalDate osztály nagy előnye, hogy a belőle létrehozott objektumokat nem lehet megváltoztatni (immutability). További nagy előnye közé tartozik, hogy egy rendkívül jól tervezett osztályról beszélünk, ami lehetővé teszi, hogy könnyen számoljunk dátumokkal.

A LocalDate fontosabb metódusai (a teljesség igénye nélkül):

* now(): statikus metódus, ami létrehoz egy objektumot, ami az aktuális napra mutat,
* getYear(): visszaadja az évet,
* getMonth(): visszaadja a hónapot,
* getDayOfYear(): visszaadja, hogy az adott dátum hanyadik nap az évben,
* getDayOfMonth(): visszaadja, hogy az adott dátum hanyadik nap a hónapban,
* getDayOfWeek(): visszaadja, hogy az adott dátum hanyadik nap a héten,
* isLeapYear(): visszaadja, hogy az adott év szőkőév-e,
* plusDay/Month/Year(amount): visszaad egy új dátumot hozzáadva az eredetihez a megadott napot/hónapot/évet,
* minusDay/Month/Year(amount): visszaad egy új dátumot kivonva az eredetihez a megadott napot/hónapot/évet,
* withDay/Month/Year(amount): visszaad egy új dátumot beállítva a megadott napot/hónapot/évet,
* isBefore(date): megmondja, hogy ez a dátum az argumentumban megadott dátum előtt van-e,
* isAfter(date): megmondja, hogy ez a dátum az argumentumban megadott dátum után van-e,
* of(year, month, day): segéd metódus dátumok készítéséhez,
* parse(text): String-ből dátumot készít,
* format(formatter): egy dátumból egy meghatározott formátumú szöveget készít például.: 2017-08-09, 20170809.

**Példa 1:** Készítsünk egy programot, ami megmondja, hogy milyen napra esik az aktuális év és a következő év első napja.

| public static void main(String[] args) {  LocalDate now = LocalDate.now();  LocalDate firstDayOfThisYear = now.withDayOfYear(1);  DayOfWeek dayOfWeek = firstDayOfThisYear.getDayOfWeek();  System.out.println("First day of this year: " + dayOfWeek);  System.out.println("First day of next year: " + now.plusYears(1).withDayOfYear(1).getDayOfWeek()); } |
| --- |

**Eredmény**

| First day of this year: MONDAY First day of next year: TUESDAY |
| --- |

**Példa 2:** Készítsünk egy programot, ami megmondja, hogy milyen napra esett 1992-05-22 és 1996-05-15.

| public static void main(String[] args) {  LocalDate date = LocalDate.parse("1992-05-22");  System.out.println("1992-05-22 was " + date.getDayOfWeek());  LocalDate date2 = LocalDate.of(1996, 5, 15);  System.out.println("1996-05-15 was " + date2.getDayOfWeek()); } |
| --- |

**Eredmény**

| 1992-05-22 was FRIDAY 1996-05-15 was WEDNESDAY |
| --- |

## LocalTime

A [LocalTime](https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html) osztály segítségével idő leírására szolgáló objektumokat hozhatunk létre. Hasonlóan a LocalDate-hez a LocalTime-ból létrehozott objektumok sem változtathatóak meg inicializálás után.

A LocalTime fontosabb metódusai (a teljesség igénye nélkül):

* now(): statikus metódus, ami létrehoz egy objektumot, ami az aktuális időre mutat,
* getHour(): visszaadja az órát,
* getMinute(): visszaadja a percet,
* getSecond(): visszaadja a másodpercet,
* plusHours/Minutes/Seconds(amount): visszaad egy új időt hozzáadva az eredetihez a megadott órát/percet/másodpercet,
* minusHours/Minutes/Seconds(amount): visszaad egy új időt kivonva az eredetihez a megadott órát/percet/másodpercet,
* withHour/Minute/Second(amount): visszaad egy új időt beállítva a megadott órát/percet/másodpercet,
* isBefore(time): megmondja, hogy ez az időpont az argumentumban megadott időpont előtt van-e,
* isAfter(time): megmondja, hogy ez az időpont az argumentumban megadott időpont után van-e,
* of(hour, minute, second): segéd metódus időpont készítéséhez,
* parse(text): String-ből időpontot készít,
* format(formatter): egy időpontból egy meghatározott formátumú szöveget készít például.: 12:44:37, 124437.

**Példa**: Készítsünk egy programot, ami megmondja, hogy ebédidő van-e. Az ebédidő legyen 12 és 13 óra között.

| public static void main(String[] args) {  LocalTime lunchStart = LocalTime.parse("12:00");  LocalTime lunchEnd = LocalTime.of(13, 0);  LocalTime now = LocalTime.now();  if (!lunchStart.isAfter(now) && now.isBefore(lunchEnd)) {  System.out.println("Lunch time");  } else {  System.out.println("It's not lunch time");  } } |
| --- |

**Eredmény:**

12:00 és 12:59:59 között: “*Lunch time”*, Egyébként: “*It’s not lunch time!*”

## 

## 

## LocalDateTime

A [LocalDateTime](https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html) osztály segítségével dátum és idő leírására szolgáló objektumokat hozhatunk létre. Hasonlóan a LocalDate-hez és a LocalTime-hoz hasonlóan a LocalDateTime-ból létrehozott objektumok sem változtathatóak meg inicializálás után.

| public static void main(String[] args) {  LocalDateTime now = LocalDateTime.now();   System.out.println("Before Formatting: " + now);   DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");   String formatDateTime = now.format(format);   System.out.println("After Formatting: " + formatDateTime); } |
| --- |

## System.currentTimeMillis()

A pontos idő meghatározásában fontos szerepet játszik a System.currentTimeMillis() metódus. Ez a java.lang packge alatt lévő System osztály egy metódusa, amely visszaadja a jelenlegi pontos időt ezredmásodperc pontossággal. Az így megkapott szám 1970. január 1. éjféltől az eddig eltelt ezredmásodperceket jelenti UTC időzónában. Ennek a számnak a pontossága változó lehet különböző operációs rendszereken.

| public static void main(String[] args) {  long milliseconds = System.currentTimeMillis();  System.out.println(milliseconds);   int seconds = (int) (milliseconds / 1000) % 60 ;  int minutes = (int) ((milliseconds / (1000\*60)) % 60);  int hours = (int) ((milliseconds / (1000\*60\*60)) % 24);   System.out.println(hours + ":" + minutes + ":" + seconds); } |
| --- |

**Eredmény**

| 1519847506817 19:51:46 |
| --- |

# Enumok

Speciális adattípus, mely engedélyezi a változóknak hogy előre beállított érteket vegyenek fel. Enum létrehozáshoz *enum* kulcsszót használjuk (a class vagy interface helyén) és az értékeket vesszővel választjuk el. Fontos, hogy az értékeknek nagybetűseknek kell lenniük. Létrehozása történhet külön fájlként vagy egy meglévő osztályban is.

**Példa a létrehozásra:**

| enum Level {  LOW,  MEDIUM,  HIGH } |
| --- |

**Példa a használatra a . segítségével:**

| Level myVar = Level.MEDIUM; System.out.println(myvar); |
| --- |

**Példa 2:**

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

| Day myVar = Day.FRIDAY; System.out.println(myvar); |
| --- |

## Metódusa

values() metódusa visszaadja tömb formában az enum előre definiált értékeit.

**Példa:**

| for (Level myVar : Level.values()) {  System.out.println(myVar); } |
| --- |

Enum nem lehet leszármazottja egy osztálynak sem.

Mikor használjuk? Elsősorban olyan értékek létrehozására, tárolására amik nem változnak például hónapok, napok, színek, kártyák a pakliban, nemek…stb

# 

# Feladatok

1. (Órai) Írjunk programot, ami tartalmazza a legfontosabb változó létrehozásokat és használatukat, valamint írassuk ki az értéküket a képernyőre.
2. (Órai) Készítsünk programot, ami a megadott **két szám**ot, “a”-t és “b” letárolja, majd megmondja mennyi lesz a **maradék**, ha “a”-t elosztjuk “b”-vel.
   1. parancssori argumentumként olvassa be a számokat
   2. felhasználótól kérjük be a számokat
3. Készítsetek programot, ami a megadott **két szám**ot, “a”-t és “b” letárolja, majd megmondja mennyi lesz “a” és “b” **szorzata**, miután megszoroztuk az “a”-t 3-al.
   1. parancssori argumentumként olvassa be a számokat
   2. felhasználótól kérjük be a számokat
4. (Órai) Írjunk **Fej vagy írás** programot, amely kérdezze meg, hogy mire tippelünk, mit választunk. Ezután sorsolja ki, hogy az érme melyik oldala kerül felülre. Attól függően, hogy eltaláltuk, vagy hibáztunk írja ki az eredményt.
5. (Órai) Írjunk programot, ami bekér 6 nevet a felhasználótól, majd visszaadja véletlenszerűen az egyikőjük nevét, mint a **tombola** nyertesét.
6. Készítsetek **szerencsesüti** programot, amely minden futtatása során kiír egy kis bölcs üzenetet. A program véletlenszerűen írjon ki üzeneteket, amit előre felvettünk benne. Mindig csak egyet.
7. (Órai) Írjunk programot, amely beolvassa hogy hányadik nap van a héten (1-7-ig), majd egy változóban tárolja azt. A **hét napjai** közül kitüntetett nap a csütörtök, péntek, szombat. Ha a változó értéke ezen napok valamelyikre, akkor írja ki a program hogy *“Italos nap van, ide a citromos sört de gyorsan”*. Egyébként írja ki, hogy *“Ez nem a Te napod, majd talán holnap”*
   1. a sorszám beolvasás történjen console-ról
   2. parancssori argumentumként olvassa be a sorszámot
8. Készítsetek programot, amely a felhasználótól vár 3 számot, amely számokról el tudja dönteni, hogy lehet-e egy **háromszög** 3 oldala, vagyis szerkeszthető-e a háromszög.
   1. Ha lehet, akkor írja ki: “You are a lucky person.” (Pl.: 2,3,4-re)
   2. Egyébként, írja ki: “Please, try again.” (Pl.: 1,2,4-re)

## 

## ------Ciklus feladatok------

1. (Órai) Írjunk programot, amely a **számok négyzetei**t írja ki 0-tól 20-ig.
   1. készítsük el while-al
   2. készítsük el for-al
2. Készítsetek egy programot, amely a **számok szorzatát** írja ki 0-tól 10-ig. (magukat a számokat nem, csak az összes szám szorzatát)
   1. készítsük el while-al
   2. készítsük el for-al
3. Készítsetek egy **összegző** programot, ami egész számokat olvas be a felhasználótól mindaddig, amíg az az x karaktert nem adja meg szám helyett. Amikor ez megtörténik írjuk ki a megadott számok összegét.
   1. készítsük el while-al
   2. készítsük el for-al
4. (Otthoni haladó) Írjunk egy **sörösüvegek** visszaváltó programot. A zöld üvegeket 10 forintért, a feketéket pedig 30-ért. A program minden üvegnél kérdezze meg, hogy milyen színű, illetve jegyezze meg, hogy mennyivel nőtt a beváltott üvegek értékének összege. Ha a felhasználó az X karaktert adja meg, úgy a program írja ki, hogy mennyi pénzt ad a beváltott üvegekért.
5. (Órai) Írjunk programot, mely **5-ös lottó lehetséges számai**t generálja le nekünk a program indítását követően.
6. Készítsetek programot, amely **más lottók**nak megfelelő potenciális nyerőszámokat is kiírja a képernyőre.
   1. hatoslottónak és skandináv lottó
   2. eurojackpot és luxor
7. (Órai) Írjunk programot, amely egy **dobókocka** szerencse**játék** program. A játékot 2 játékos játsza, neveiket kérje be a program. A játékosok felváltva dobnak. A játék 3+1 fordulóból áll. A 4. Forduló dupla pontért meg. Az a játékos nyer, aki a legtöbb pontot összegyűjti. *(Amennyit dob a játékos az egyes fordulókban, annyival több van neki)*
8. (Órai) Írjunk **kő-papír-olló** játékot. A program minden indítás után válasszon egy random eszközt, majd vesse össze a felhasználó által megadott eszközzel. Miután kiértékelte, hogy ki nyert írja ki, hogy mi volt nála és hogy ki nyerte a játékot. Figyeljünk rá, hogy a játékos csak a kő, papír, olló eszközöket választhassa. Ellenkező esetben írjunk ki egy figyelmeztető üzenetet.
9. Fejlesszétek tovább a **kő-papír-olló** játékot. A játék ne érjen véget egy kör után, hanem addig folytatódjon, amíg a felhasználó a három megengedett eszköz helyett az X karaktert adja meg. A játék számolja a pontokat.
   1. az “x” karakter leütése után írja ki, hogy vége a játéknak, valamit hogy kinek hány pontja lett
   2. amennyiben a felhasználó csal és valamilyen másik eszközt próbál használni, úgy vonjon le egy pontot tőle
10. Készítsetek programot, amely a program indításakor bekér két számot, valamint egy operátort. Majd írja ki konzolra a végeredményt. A **számológép** tudjon összeadni, kivonni, szorozni, osztani. A megoldást töltsétek fel a *progmaticjrta* group alatti repository-tokba.
    1. Opcionális: “*pumpedelni”*, ami a számok köbének szorzata.
11. (Órai) Egy 21 fős csoportban ki kell választanunk két tanulót, aki **felelni** fog. Random generátorok segítségével válasszuk ki a két tanuló sorszámát a naplóban. (1-től indul a sorszámuk)
    1. használjunk while ciklust az implementációhoz
    2. használjunk do while ciklust az implementációhoz
12. Készítsetek egy **életkor ellenőrző** programot, amely bekéri a felhasználótól a használni kívánt életkort, majd ellenőrzi, hogy negatív számot (vagy 0-t) adott-e meg. Ismételjük mindaddig a bekérés utasítását, amíg helyes életkort nem adott meg a felhasználó.
    1. használjunk while ciklust az implementációhoz
    2. használjunk do while ciklust az implementációhoz
13. Fejlesszétek tovább a **Háromszög** alkalmazást (8as), hogy mindaddig ismételje a bekérést a felhasználótól, amíg nem ad neki helyes háromszög oldal adatokat.
14. (Órai) Írjuk át a **számológép** feladatot, switch-case adatszerkezet segítségével, kiváltva az if-else szelekciókat. Miután kiírta az eredményt kérdezze meg, hogy a felhasználó szeretne-e további számításokat végezni. Y/N-val, (azaz Yes/No-val) lehet válaszolni. Y esetén természetesen újraindul a fenti folyamat. N esetén pedig az illedelmes számológép megköszöni, hogy szolgálhatta felhasználóját.
15. Készítsétek el a **szerencsesüti** program switch case szerinti megvalósítását. Cseréljétek le az *if/else if/else* blokkokat *switch case*-ekkel.
16. Készítsetek **szlogen automatát,** amely a felhasználó által beolvasott márkára visszaadja a felhasználónak az adott márka szlogenjét.
    1. switch-case adatszerkezettel oldjátok meg
    2. induljon újra automatikusan a program mindaddig, amíg a márka helyett “exit”-et nem ír a felhasználó
    3. a szlogen visszaadása után kérdezzük meg a felhasználót, hogy kíváncsi-e új szlogenre. Yes/No. Yes esetén mindig újraindul a program, No esetén szép napot kívánunk neki és kilépünk.
17. Készítsétek el az **hét napjai** program switch case szerinti megvalósítását. Cseréljétek le az *if/else if/else* blokkokat *switch case*-ekkel.
18. Készítsetek egy **prímszám** eldöntő programot. A program bekér indulást követően egy számot, majd kiírja hogy az adott szám prím szám-e vagy sem.
19. Készítsetek egy **tökéletes szám** eldöntő programot. A program bekér indulást követően egy számot, majd kiírja hogy az adott szám tökéletes szám-e vagy sem. A tökéletes szám, egy olyan speciális szám amely, pozitív osztóinak összege mindig a szám kétszerese lesz.
20. (Órai) Írjunk programot, ami a parancssori paramétereket beolvassa induláskor, majd azokat ki is írja a képernyőre. Ha ez megvan, ezután írjuk ki ezeket a bemenő paramétereket visszafelé is.
21. (Órai) Írjunk programot, ami véletlenszerűen kiválaszt egy szülőt és a szülőkhöz tartozó előre definiált programokból egy lehetséges programot. Pl. Anyával: Sütés,Játszózás, Apával: Barkácsolás, Fűnyírás, lehetséges output: Anya - Sütés

## ------ARRAY feladatok------

1. (Órai) Készítsünk egy tömböt, amiben tároljuk a hét napjait és írassuk ki az elemeit egy ciklussal majd külön egy napot is, amelyik a kedvenc napunk.
2. (Órai) Készítsünk egy tömböt, amiben 1-90-ig random számokat tárolunk, szám szerint 5-öt (lehetnek egyezőek). Majd írassuk is ki őket egysorban.
3. Készítsetek egy programot, amely elszámol 1-től 100-ig és a páros számokat lementi egy evenNumbers nevű array-be. Írjuk is ki a képernyőre egy sorban az értékeit.
4. Készítsetek egy programot, amelyben egy tömbnek már a létrehozásakor definiáljuk az értékeket. A String típusú array neve legyen *carsArray* és állítsunk be neki legalább 7db autómárkát. Hozzunk létre egy *myFavoriteCar* nevű String változót és állítsuk be az egyik, tömbhöz is hozzáadott autómárkát, mint kezdőérték. Egy ciklussal menjünk végig az elemeken, azaz járjuk be a tömböt. Ha az adott indexű tömb elem megegyezik a *myFavoriteCar* változó értékével akkor írjuk ki annak értékét a konzolra és fűzzük hozzá, hogy “ a kedvenc autómárkám.”
5. (Órai) Írjunk egy programot, amely egy array-ben elmenti a számokat 1-21-ig majd kiírja őket visszafele egymás után szóközzel elválasztva.
6. Készítsetek egy *ageInput* nevű arrayst, amiben a következő számok találhatóak: 5, -3, 0, 21, 55, 18, 32, 255, 42. Írjatok egy for ciklust, amely bejárja ezt a tömböt és valid age esetén kiírja azt a képernyőre, de csak az életszerű, valós age-eket.
7. (Órai) Írjunk programot, amely előre meghatározott tojásárak között, kiszámolja mennyi a tojás átlagos ára.
   1. az árak egy *eggPrices* array-ben kerülnek előzetesen megadásra
   2. az árak a felhasználótól a console-ról kerülnek beolvasásra (5db).
   3. kérdezzük meg a felhasználótól hány árat szeretne rögzíteni, majd pontosan annyi értéket kérjünk is el tőle és tároljuk le őket az *eggPrices* arraybe.

## ------LIST feladatok------

1. (Órai) Készítsünk egy listát kedvenc gyümölcsökkel. Adjunk hozzá a listához legalább 7 gyümölcsöt és foreach ciklus segítségével írassuk ki az elemeit a képernyőre egymás alá.
2. Írjatok egy programot, amely egy Listában eltárolja majd “visszaolvassa” a felhasználónak azt az 5 kedvenc filmjét, amit bekértünk tőle a konzol segítségével. (Külön ciklus készüljön a beolvasásra és mentésre és külön egy az eredmény kiírására)
3. (Órai)Fejlesszük tovább az előző feladatot úgy, hogy csak akkor mentse el a beolvasott filmet, ha még előzetesen nem adta meg azt. Vagyis kerüljük el azt, hogy ha a lista eleme már létezik, akkor ne adhassuk újra hozzá.
4. Írjatok vendéglista programot, amit a felhasználó tud feltölteni. Házibulit szervezünk és tudjuk, hogy 11-en férnek el a lakásban. Azonban fontos, hogy egy vendéget csak egyszer adhasson hozzá a felhasználó.
5. (Órai) Készítsünk egy számla végösszeg kalkulátor programot, amelybe be tudom rögzíteni az adott hónap különböző számláinak az összegét. A program pedig összeadja őket és kiírja mennyit kell a családi kasszából fizetni összesen. Magát, hogy milyen számla az amit be kell fizetni, nem fontos tárolni most, elég ha az összegek mentésre, majd összeadásra kerülnek.
6. Írjatok programot, amely a megadott testhőmérsékleti adatokból, amit a felhasználó rögzített, a listában tárolást követően
   1. kiszámolja mennyi volt az átlagos testhő
   2. hányszor volt lázas a bevitt adatok alapján a delikvens

Az adatokat az utolsó 3 nap, napi 3 (reggeli, napközbeni és esti) adataival kell mindig rögzíteni, tehát mindig fix 9 testhőmérsékleti adatot.

1. Írjatok egy bevásárló lista programot. Amely nem csinál mást, mint a felhasználótól beolvasott elemeket elmenti egy List-be és a végén ki is írja a felhasználó számára az elmentett elemeket.
   1. fix 7 elemű legyen a lista, pontosan 7 elemet lehessen hozzáadni
   2. ne legyen limitálva, hogy mennyi elemet adhat a felhasználó. Ha a felhasználó az “x” karaktert ír lista elemként, akkor szakítsa meg a beolvasást és folytatódjon a program futása.
   3. oldjuk meg, hogy csak akkor adja hozzá a listához a felhasználó által beírt elemet ha még nincs benne. Ha már benne van, akkor írjuk ki, hogy “Ez az elem már hozzáadásra került!”.
2. Írjatok egy nyaralós checklist programot, amelybe tudjátok rögzíteni, hogy mik azok a tevékenységek, dolgok amiket a nyaralás elindulása előtt el kell intézni vagy el kell tenni, hogy minden meglegyen, rendben legyen. Ha ‘X’-et ír tevékenység helyett, akkor fejezze be az elemek hozzáadását.
   1. ügyeljetek rá, hogy kétszer ne lehessen ugyanazt az elemet a listához adni. Ha mégis megpróbálja a felhasználó akkor kapjon hibaüzenetet.
   2. amennyiben a felhasználó azt az elemet adja meg, ami már létezik a listában, úgy akkor azt törölje onnan. (jelezve, hogy az a feladat/tevékenység teljesült)
   3. amennyiben elfogyott minden elem a listáról, akkor írja ki a program, hogy “Minden megvan, indulhat a nyaralás!”
3. (Órai) Készítsünk egy elemzést a legutóbbi futásainkról. Az adatokat már kinyertük az aktivitás tracker-ünkbők, az alábbi formában: *Arrays.asList( 12.30, 12.45, 12.35, 12.54, 12.53, 12.45, 12.61, 12.62, 12.73, 12.91, 12.90, 12.72, 12.85, 12.90, 12.70, 12.58, 12.61, 12.67, 12.63, 12.61, 12.74, 12.84, 12.99, 13.10, 13.21, 13.39, 13.23, 13.09, 13.15, 13.00, 12.91, 12.85, 12.89, 13.00, 13.03, 13.16, 13.25, 13.35, 13.36, 13.28,13.42, 13.49, 13.36, 13.32, 13.20, 13.23, 13.20, 13.02, 12.91, 12.74, 12.81, 12.63, 12.79, 12.65, 12.59, 12.77, 12.77, 12.89, 12.76, 12.76,12.91, 12.90, 12.86, 12.89, 12.83, 12.67, 12.69, 12.65, 12.73, 12.87);*

Ami a feladat, hogy írassuk ki a konzolra a listában található legnagyobb, legkisebb és átlagos sebességet.

1. Készítsetek programot, amely kiírja azokat a számokat, amiket a legutolsó 300 lottó ötöslottó nyerőszámok (1-90) között nem szerepel. A számokat nyugodtan tegyük egy listába a programunkban az List<Integer> winnerNumbers = Arrays.asList(31, 87, 6, 47, ...); módon. A számok pedig a következők:  
   *31, 87, 6, 47, 38, 44, 79, 16, 87, 76, 17, 13, 73, 13, 90, 77, 1, 25, 31, 61, 55, 13, 67, 88, 36, 59, 56, 67, 46, 34, 68, 90, 18, 28, 3, 55, 31, 45, 85, 43, 11, 9, 59, 78, 66, 71, 50, 18, 37, 50, 76, 22, 26, 33, 32, 52, 28, 45, 80, 64, 17, 57, 81, 67, 51, 21, 9, 5, 57, 15, 37, 69, 60, 59, 85, 18, 4, 62, 14, 46, 85, 42, 4, 87, 37, 90, 37, 5, 53, 74, 60, 70, 25, 77, 30, 21, 53, 6, 10, 25, 89, 10, 18, 78, 57, 18, 87, 66, 35, 38, 82, 54, 4, 46, 41, 4, 77, 74, 11, 50, 4, 85, 57, 71, 47, 68, 76, 23, 26, 10, 70, 15, 14, 34, 35, 59, 37, 63, 49, 26, 47, 22, 90, 19, 3, 90, 2, 63, 69, 81, 13, 29, 61, 82, 54, 69, 5, 7, 6, 36, 5, 50, 88, 42, 65, 22, 33, 74, 11, 9, 71, 14, 67, 56, 52, 80, 81, 66, 30, 48, 7, 49, 4, 13, 54, 42, 46, 61, 71, 35, 17, 56, 83, 76, 15, 48, 84, 74, 89, 33, 17, 28, 34, 36, 58, 81, 28, 23, 1, 87, 84, 5, 65, 61, 66, 5, 76, 54, 77, 52, 60, 56, 65, 90, 21, 10, 82, 56, 85, 85, 77, 88, 67, 83, 52, 8, 3, 2, 38, 78, 19, 24, 34, 6, 83, 37, 57, 50, 85, 74, 82, 24, 31, 4, 43, 70, 78, 3, 18, 30, 23, 18, 46, 10, 85, 56, 8, 68, 79, 46, 59, 4, 34, 30, 40, 5, 48, 74, 35, 69, 19, 7, 89, 35, 66, 79, 77*

Az eredményt is mentsünk le egy *notWinnerNumbers* listába és írjuk ki a tartalmát a képernyőre.

1. (Órai) Írjunk programot, ami összeszámolja a dolgozói szavazatokat amely a karácsonyi party helyszínével kapcsolatos. Szerencsére egy gyakornok összegyűjtötte a szavazatokat egy Listába, íme:

*"Rumour", "Babel", "Babel","Babel", "Babel","Babel", "Babel","Babel", "Babel","Babel", "Babel","Babel", "Babel","Rumour","Babel", "Babel", "Rumour","Rumour", "Rumour","Babel", "Babel", "Rumour","Rumour", "Rumour", "Babel", "Rumour", "Rumour", "Rumour", "Rumour", "Rumour", "Rumour", "Babel", "Rumour","Rumour", "Rumour","Babel", "Babel", "Rumour", "Rumour", "Rumour", "Rumour"*

Készítsünk egy List-et ezzel a kezdőértékkel és számoljuk meg a szavazatokat, majd írjuk ki a győztes helyet a képernyőre.

## ------SET feladatok------

1. (Órai) Készítsünk egy halmazt a kedvenc gyümölcsökkel. Adjunk hozzá legalább 7 gyümölcsöt és foreach ciklus segítségével írassuk ki az elemeit a képernyőre egymás alá.
2. Írjatok egy programot, amely egy Set-ben eltárolja majd “visszaolvassa” a felhasználónak azt az 5 kedvenc filmjét, amit bekértünk tőle a konzol segítségével. (Külön ciklus készüljön a beolvasásra és mentésre és külön egy az eredmény kiírására)
3. Írjatok vendéglista programot, amit a felhasználó tud feltölteni. Házibulit szervezünk és tudjuk, hogy 11-en férnek el a lakásban. Tároljuk a vendégeket Set-ben, hogy ne kelljen arra figyelni, hogy valaki benne van-e a listában már vagy sem.
   1. Viszont oldjuk meg, hogy addig kérje inputként a program a vendégeket, amíg fel nem töltjük a 11 helyet
   2. A felhasználó a program indításakor dönthesse el, hány vendéget szeretne meghívni
4. Írjatok egy nyaralós checklist programot, amelybe tudjátok rögzíteni, hogy mik azok a tevékenységek, dolgok amiket a nyaralás elindulása előtt el kell intézni vagy el kell tenni, hogy minden meglegyen, rendben legyen. Ha ‘X’-et ír tevékenység helyett, akkor fejezze be az elemek hozzáadását.
5. Írjatok egy tippelő játék programot. Töltsünk fel egy Set-et 3 darab random számmal. A számok 0 és 10 között lehetnek (0-val, de 10-el nem). A felhasználótól kérjünk be számokat hasonlóan 0 és 10 között és vizsgáljuk meg benne van-e a Set-ben. Amennyiben igen, vegyük ki belőle. Ha kiürült a Set, mert kivettünk belőle minden elemet, akkor vége a játéknak.
   1. Számoljuk, hogy hányszor kellett tippelnie a felhasználónak, majd írjuk ki: “Gratulálunk, X kör kellett a sikerhez”.
   2. Ha eltalált egy számot a felhasználó a játék során, akkor írja ki a program, hogy “Talált, még X szám, amit meg kell keresned”.

## ------MAP feladatok------

1. (Órai) Készítsünk egy Map-et kedvenc gyümölcsökkel. Adjunk hozzá a listához legalább 7 gyümölcsöt és egy értéket, hogy mennyire szeretjük. Ez az érték 1-3 között lehet, mely esetén a 3 amit jobban szeretünk. Foreach ciklus segítségével írassuk ki az elemeit a képernyőre egymás alá.
2. Írjatok egy telefonkönyv programot, amely a konzolról a felhasználó által megadott és beolvasott 5 darab név-hez a hozzá tartozó telefonszámot lementi egy HashMap-ben.
3. Írjatok egy bevásárló lista programot, amely nem csinál mást, mint a felhasználótól beolvasott elemeket, majd az elemhez tartozó darabszámot elmenti egy Map-be és a végén ki is írja a felhasználó számára az elmentett elemeket.
   1. fix 7 elemű legyen a Map, pontosan 7 elemet lehessen hozzáadni
   2. oldjuk meg, hogy ha az elem már szerepel a Map-ben, akkor írja ki a program, hogy “Az elem darabszáma frissítésre került”
   3. ne legyen limitálva, hogy mennyi elemet adhat a felhasználó. Miután hozzáadott 3 elemet a felhasználó, kérdezze meg minden elem hozzáadása után, hogy szeretne-e új elemet hozzáadni. “Y” esetén újbóli beolvasás, “N” esetén a Map kiíratása következik.
4. Készítsetek egy számla végösszeg kalkulátor programot, amelybe be lehet rögzíteni az adott hónap különböző számláinak az összegét. A program pedig összeadja őket és kiírja mennyit kell a családi kasszából fizetni összesen. Magát, hogy milyen számla az amit be kell fizetni, az is kerüljön beolvasásra az összegekkel együtt (de külön beolvasással - 3 darab -).
   1. Az elemek kerüljenek Mapben mentésre, majd összeadásra és kiíratása.
   2. Amennyiben a felhasználó 0 ft-os számlát ír, úgy azt törölje ki a Map-ből a program
5. (Órai) Írjunk könyv nyilvántartás programot. A feladat, hogy meg kell számolni pontosan hány különböző könyv van (cím szerint) és hogy hány példány van belőle a könyvesboltban. Szerencsére a kutató és adminisztrációs munka nagyját megcsinálták a diákmunkások, így egy ömlesztett lista a könyvekről már rendelkezésünkre áll:

*"Son", "Harry Potter VI.", "Down and Out in Paris and London", "All about Sam", "1984", "Keep the Aspidistra Flying", "The Fellowship of the Ring", "Keep the Aspidistra Flying", "A Summer to Die", "Harry Potter V", "The Return of the King", "Harry Potter VI.", "The Return of the King", "Down and Out in Paris and London", "Harry Potter V", "A Summer to Die", "Harry Potter I.", "Harry Potter III.", "All about Sam", "Animal Farm", "Gathering Blue", "Homage to Catalonia", "Son", "The Two Towers", "Harry Potter III.", "Messenger", "The Return of the King", "Homage to Catalonia", "Harry Potter III.", "Harry Potter I.", "The Two Towers", "Gathering Blue", "Messenger", "The Fellowship of the Ring", "1984", "Harry Potter VI.", "Keep the Aspidistra Flying", "Gathering Blue", "Harry Potter V", "Harry Potter II.", "Homage to Catalonia", "Harry Potter V", "Animal Farm", "All about Sam", "Son"*

Array.asList(“Son”,... -ként nyugodtan állítsuk be a programunknak a List kezdeti értékét és a feladat végeredményét mentsük egy Map-be.

* 1. Írjuk ki a map tartalmát a képernyőre úgy, hogy az alábbi példának megfeleljen: “The Fellowship of the Ring - 4 darab”
  2. Hány darab könyv van összesen és hány különböző könyv van?
  3. Tételezzük fel, hogy azokat a könyveket amikből egy darab van, azokat kivesszük ideiglenesen a nyilvántartásból, mert a cégvezető értékesnek találja és meg szeretné tartani őket. Keressük meg ezeket a könyveket és töröljük a nyilvántartásból.
  4. (next level) Tételezzük fel, hogy a cégvezető nem is annyira rossz arc és azokat a könyveket, amelyekből 4 darab van, 1-et elajándékoz rászorulóknak. Frissítsük ezen könyvek darabszámát 3-ra.

1. Készítsetek programot, ami összeszámolja a dolgozói szavazatokat amely a karácsonyi party helyszínével kapcsolatos. Szerencsére egy gyakornok összegyűjtötte a szavazatokat egy Listába, íme:

*"Salt", "Platán", "42", "Essencia", "Babel", "Borkonyha", "Costes", "42", "Borkonyha", "Rumour", "Platán", "Babel", "Stand", "Rumour", "Salt", "Platán", "42", "Rumour", "Rumour", "Borkonyha", "Rumour", "Borkonyha", "Babel", "Essencia", "Salt", "Platán", "Borkonyha", "42", "Babel", "Essencia", "42", "Stand", "Salt", "Costes", "42", "42", "Costes", "Stand", "Borkonyha", "Stand", "Costes", "Babel", "Essencia", "Stand", "Salt", "Platán", "Costes", "Salt", "Platán", "Salt", "Platán", "Essencia", "Rumour", "Salt", "Platán", "Rumour", "Stand", "Salt", "Essencia", "Babel", "Babel", "Essencia", "Rumour", "Essencia", "Salt", "Platán", "Borkonyha", "Stand", "Essencia", "Rumour", "Borkonyha", "Costes", "Babel", "Costes", "Salt", "Platán", "Essencia", "Stand", "Costes", "Borkonyha", "Essencia", "42", "Essencia", "42", "Stand", "Rumour", "Salt", "Platán", "Rumour", "Borkonyha", "Stand", "Babel", "Rumour", "Stand", "Salt", "Essencia", "Costes", "Costes", "42", "42", "Salt", "Platán", "Essencia", "Babel", "Costes", "Rumour", "Costes", "Stand", "Rumour", "Essencia", "Borkonyha", "Salt", "Platán", "Rumour", "Borkonyha", "42", "Babel", "Stand", "42",* "Costes", "42", "Salt", "42", "Salt", "Essencia", "Borkonyha", "Essencia", "Borkonyha", "Rumour"

Array.asList(“Essencia”,... -ként nyugodtan állítsuk be a programunknak a List kezdeti értékét és a feladat végeredményét mentsük egy Map-be.

* 1. Írjuk ki a képernyőre a szavazás eredményét az alábbi példának megfelelően: “Borkonyha - 11 szavazat”
  2. Hány Michelin csillagos étterem van? Hány szavazat érkezett?
  3. Tételezzük fel, hogy az *Essencia* jelezte, hogy nem tudják fogadni a céget. Távolítsuk el őket az eredmény Map-ből
  4. Tételezzük fel, hogy a főnök kicsit megcinkeli a szavazást a *Stand* javára és befolyásolja a szavazás végeredményét. Fiktív szavazatokat állít ki és utólag könyvelik le őket. Adjunk hozzá a *Stand* szavazataihoz 11-et.
  5. Extra (google-ből): Ki nyerte a szavazást?

1. Gondoltam egy számra …

## ------Metódus feladatok------

1. (Órai) Készítsünk metódust, amely amikor meghívjuk, akkor kiírja a képernyőre, hogy “Have a nice day, goodbye!”
2. (Órai) Készítsünk egy metódust, amely kapott szám esetén eldönti, hogy az 18-nál nagyobb-e. Ha igen, kiírja hogy “Elmúltál 18, jogosult vagy a belépésre!”, egyébként pedig írja ki, hogy “Nem vagy még 18, sicc innen!”
3. Készítsetek egy *getValidAgeFromConsole()* metódust, amelyet a main metódusban meg tudunk hívni és visszaad egy helyes age-et integer-t amit a felhasználó a console-be írt. Akkor helyes egy *age* ha 0 és 140 között van *(0 lehet, 140 nem)*. A metódus visszatérési értékét mentsük le egy *age* nevű változóba, majd írassuk ki a képernyőre annak tartalmát.
4. (Órai) A háromszög eldöntős programunk dolgozzuk át, hogy külön metódus legyen az oldal beolvasás ( *getTriangleDataFromConsole()* ), külön metódus a háromszög szerkeszthetősége ( *isTriangeValid(aoldal,boldal,coldal)* )és külön az eredmény String kiíratása *( showResultToTheUser(result) ).*
5. Dolgozzátok át a számológép programot úgy, hogy tartalmazzon külön metódust az adatokbekérésre a felhasználótól ( getNumberFromUser() ), valamint egyet az operátor beolvasásra (getOperatorFromUser() ). Ezt követően egy külön metódust amely elvégzi a műveletet és visszaadja az eredményt ( *getCalculationResult(operator, number1,number2)* ), Végül egy metódust ami kiírja az eredményt a konzolra (*showResultToTheUser(result)* )
6. Task 16 metódusítása
7. Task 33 metódusítása
8. Task 14 metódusítása
9. Task 36 metódusítása
10. Task 37 metódusítása

## ------String feladatok------

1. (Órai) Adott egy neveket tartalmazó lista. Készítsünk egy programot, ami készít egy újabb listát a meglévő nevekből nagybetűre módosítva azok minden betűjét. Miután elkészítettük az új listát írjuk ki annak tartalmát. Nem szükséges for ciklust használnunk a végeredmény kiíratásához, elég ha a listát odaadjuk a System.out.println() metódusnak.
2. Készítsetek egy alkalmazást ami mindaddig beolvas egy szöveget a konzolról, amíg x-et nem kap. Minden értéket amit beolvasott mentsünk egy egy List-be, de már csak a bevitt szöveg kisbetűs változatát!
3. (Órai) Adott egy List amiben az adott hónapban felvásárolt cégeket rögzítjük. Pl ez:

| List<String> loggedHours = Arrays.asList(  "Zuckerberg,4", "Steve,4", "Bill,1", "Bill,2", "Zuckerberg,4", "Bill,2",  "Zuckerberg,1", "Zuckerberg,3", "Zuckerberg,1", "Zuckerberg,1", "Steve,4",  "Bill,2", "Zuckerberg,3", "Bill,2", "Zuckerberg,4", "Steve,1",  "Bill,3", "Steve,2", "Steve,2", "Bill,1", "Zuckerberg,4",  "Bill,4", "Steve,4", "Bill,2", "Bill,3", "Zuckerberg,3",  "Steve,4", "Zuckerberg,4", "Bill,2", "Zuckerberg,3" ); |
| --- |

Összegezzük, hogy összesen hány céget vásároltak fel a fenti úriemberek.

1. Készítsetek egy alkalmazást, amely egy bevásárló lista. Az elemeket be tudja olvasni az alábbi formában: tétel-db vagyis, kenyér-2 vagy sör-6. Az így beolvasott elemeket feldarabolja a - jel mentén és elmenti egy shoppingList nevű Map-be. Az első fele, a kötőjel előtti rész legyen a kulcs, a második rész pedig a darabszám. System.out.println()-el írjátok ki a képernyőre a map tartalmát.
2. (Órai) Írjunk programot, amely a beolvasott cikkszám (fixen 6 karakter hosszú), majd közvetlen utána lévő darabszám beviteli értéket fel tudja dolgozni, és össze tudja adni őket, hogy összesen hány termék van. Figyelünk oda, hogy 0-s darabszám nem lehet, illetve az egyszerűség kedvéért maximum 9 lehet a darabszám. Egy példa tétel: 1376367
3. Készítsetek teszteset futási összesítő alkalmazást. A futási eredmények az alábbi formában tároljuk le egy ArrayList-ben vagy konzolról beolvasva kapjuk. Minden teszteset futás kezdődik egy 4 számjegyű számmal, majd egy 1-es ha PASS, X ha SKIP, valamint 0 ha FAILED. Egy példa adat tehát: 00171 A feladat, hogy számoljuk össze
   1. Összesen hány teszteset került futtatásra?
   2. Mennyi volt PASS? SKIP? FAILED?
   3. Jelenítsük meg az adatokat százalékos formában is
4. (Órai) Készítsünk egy konzolról beolvasó metódust, amely a beírt szöveget Stringként beolvassa és eltávolítja a felesleges szóközöket az elejéről és végéről, majd visszaadja azt. Építsünk bele olyan logikák, hogy ha 0 hosszú a bevitt kifejezés, akkor írjon ki a konzolra hibaüzenetet.
5. (Órai) Készítsünk egy metódust, amely eldönti a konzolról beolvas egy szöveget, majd a szintén beolvasott keresendő szövegről eldönti, hogy szerepel-e a szövegben. Amennyiben igen, térjen vissza true-val, egyébként false-al.
6. Írjatok egy metódust a paraméterként kapott szövegről, illetve a másik paraméterként kapott kifejezésről eldönti, hogy azzal a kifejezéssel kezdődik-e az adott szöveg vagy sem. Amennyiben igen, térjen vissza true-val, egyébként false-al.
   1. hogy ki tudjuk próbálni a metódust, konzolról olvassunk be két értéket a main metódusban és hívjuk meg a készített metódust velük.
   2. parancssori argumentumként olvassunk be két értéket a main metódusban és hívjuk meg a készített metódust velük.
   3. a metódus távolítsa el a felesleges szóközöket a String-ek elejéről és végéről
   4. amennyiben van a program indításakor parancssori argumentum beállítva akkor használja azokat. Ha viszont nincs, akkor olvassa be az adatokat a konzolról.
7. (Órai) Tételezzük fel, hogy valamit választania kell a felhasználónak a,b,c. lehetőség közül. Készítsünk olyan metódus, amely ezt a konzolról beolvasott értéket feldolgozza és csak az első karaktert adja vissza, mint felhasználói választás. Elkerülve, hogy a felhasználó pontot, zárójelet, szóközt vagy bármi mást ír véletlenül az opciója után.
8. Írjatok egy metódust, amely képes megkeresni a paraméterként kapott szöveg elejéről, a szintén paraméterként kapott darabszámig kivágni a karaktereket. Az eredményStringet adja vissza a metódus. Parancssori argumentumként feldolgozott értékekkel, futassátok meg a metódust a main-ből.
9. (Órai) Írjunk programot, amely a beolvasott szövegben lecseréli a “legrosszabb” szót, a “legjobb”-ra.
10. Írjatok családbarát metódust, amely egy, a metódusban előre létrehozott listában tárolt szavazat megkeres egyesével a paraméterként kapott szövegben.
    1. Amennyiben megtalálja, cserélje ezeket “\*” karakterekre.
    2. A csillagozott rész legyen pont annyi karakter, amennyi az eredeti szó volt
11. Egy jövőbeli társadalomban az emberek vitaminfogyasztását pontosan jegyzik. A vitaminok közül a C, D és E vitaminok naponta bevitt mennyiségét gyűjtik egy ArrayList-be. Ebben a társadalomban a napi ajánlott mennyiség C vitaminból 500 mg, D vitaminól 800 NE (nemzetközi egység), E vitaminból pedig 100 mg.

A feladat, hogy dolgozzuk fel a lenti listát, figyelmeztetést adva azokról a személyekről, akik nem fogyasztottak elegendő vitamint. Írjuk ki a személyek nevét és a vitamin nevét, amiből nem fogyasztottak elegendő mennyiséget, a fogyasztott mennyiséggel együtt.

| List<String> vitaminConsumptions = Arrays.asList(  "Name:Scarlett-6fccbdca,C:50,D:600,E:3000",  "Name:Scarlett-c52ec32f,C:350,D:2000,E:4000",  "Name:Bob-600b03a6,C:150,D:2000,E:2200",  "Name:Scarlett-38d7e3a2,C:150,D:1000,E:100",  "Name:Scarlett-3abde882,C:950,D:1200,E:300",  "Name:Scarlett-76b92994,C:900,D:1600,E:4000",  "Name:Bob-2b4a1544,C:600,D:800,E:1800",  "Name:Bob-2421990a,C:50,D:200,E:1200",  "Name:Bob-dd9be590,C:800,D:1200,E:600",  "Name:Bob-4e0104b2,C:850,D:1000,E:2400",  "Name:Bob-d63cb4ae,C:300,D:1800,E:200",  "Name:Scarlett-294ece9c,C:100,D:1600,E:3100",  "Name:Bob-a2a8b3d4,C:300,D:1200,E:1000",  "Name:Bob-b87ae9dc,C:800,D:2000,E:2500",  "Name:Scarlett-8e174152,C:350,D:200,E:3800",  "Name:Scarlett-ec5a81be,C:450,D:1600,E:3700",  "Name:Scarlett-ef7b71b9,C:50,D:400,E:1500",  "Name:Scarlett-97856f0d,C:1000,D:1200,E:2000",  "Name:Bob-2883e779,C:950,D:800,E:4000",  "Name:Scarlett-abcc50bc,C:550,D:600,E:2800",  "Name:Scarlett-28af295b,C:650,D:800,E:500",  "Name:Bob-3d58e4c1,C:700,D:800,E:1700",  "Name:Bob-0b8301b7,C:150,D:1400,E:1000",  "Name:Bob-4041df3e,C:900,D:1800,E:1700",  "Name:Scarlett-3e15502a,C:450,D:600,E:2200",  "Name:Scarlett-1fae31bc,C:950,D:800,E:1800",  "Name:Bob-47126df2,C:1000,D:1000,E:1900",  "Name:Scarlett-63167044,C:550,D:1400,E:1600",  "Name:Scarlett-ea01d4c2,C:800,D:800,E:3000",  "Name:Scarlett-1c0c8bd5,C:1000,D:800,E:4000",  "Name:Scarlett-af993b7d,C:800,D:1400,E:500",  "Name:Scarlett-dc49a3f7,C:500,D:800,E:1200",  "Name:Bob-2c2d260b,C:600,D:1200,E:2600",  "Name:Bob-a4e7e3f7,C:150,D:1800,E:3600",  "Name:Scarlett-991f6f64,C:750,D:2000,E:4000" ); |
| --- |

Tehát egy példa kiíratás: Hey Bob-a4e7e3f7! Your C-vitamin consumption is too low!

## ------Exception feladatok------

1. (Órai) Készítsünk programot, amely kiírja a “There is no program arguments.” ha nem talál hozzáadott program argumenteket. Készítsünk egy metódust (*printArgConsole()*), amelyik kiírja az első paraméter tartalmát a konzolra.
   1. Kapjuk el a lehetséges keletkező exceptiont a metódusban
   2. Dobjuk tovább a lehetséges exceptiont és kapjuk el azt a main metódusban
2. Készítsetek egy metódust (getNumberFromConsole()), amely beolvas egy számot a konzolról, a nextInt() metódus segítségével. Próbáljátok ki milyen exception érkezik, ha futtatáskor szám helyett valami szöveget írtuk. Ha megvan az exception neve,
   1. akkor írjatok rá egy try-catch blokkot amely-ben elkapjátok ezt az exceptiont és kiírjátok a konzolra az alábbi üzenetet: “You added a string, only number please!”
   2. Dobjátok tovább a metódusban keletkező lehetséges exceptiont, és a hívás helyén a main-ben kapjátok el
   3. (extra) mindaddig kérjen be a felhasználótól adatot, amíg számot nem ad
3. Kérjünk el egy stringet a felhasználótól és írjuk ki a 10. karakterét. Kezeljük le a keletkező exceptiont.

## ------IO feladatok------

1. (Órai) Adott egy neveket tartalmazó fájl. Készítsünk egy programot, ami beolvassa annak tartalmát és megvizsgálja, hogy a console-ról beolvasott név benne van-e.
2. Készítsetek programot, amely beolvas egy híres sportolóat tartalmazó fájl, majd megvizsgálja, hogy a parancssori argumentumként megadott sportolót tartalmazza-e. Ha igen, írja ki hogy “It’s in the file”, egyébként pedig írja ki, hogy “No luck” a konzolra.
3. (Órai) Írjunk egy alkalmazást, amely biztonsági mentést készít a használt *Program arguments*-ekről. Ehhez készítsünk egy metódust, amely elvégzi a fájlba írást nekünk. A fájl neve legye *used\_arguments.txt*.
4. Készítsétek el a classic bevásárlólista alkalmazást egyszerűsített változatát, amely beolvas a konzolról tételek és darabszámokat (pl.: “tej 3db”). Ezen tételeket mentsük el egy String elemű listában. Készítsünk egy metódust, amely a paraméterül kapott String-et kiírja egy shoppinglist.txt-be. (Tehát a listát át kell alakítani, összefűzni egy hosszú Stringgé egy elválasztó karakter, pl vessző használatával)
5. Készítsetek egy élelmiszerkészlet alkalmazást. Tegyünk fel, hogy van egy fájl, ami tartalmazza, hogy éppen milyen típusú élelmiszerből mennyi van otthon és hogy mennyi az ajánlott mennyiség. A fájl hasonlóan néz ki a ehhez *(actual-food-set.csv)*:

| Name,Amount, Required Amount Coffee, 17, 25 Milk, 0, 3 Cola, 1, 2 Orange, 4, 5 Oat flakes, 2, 4 |
| --- |

A feladat, hogy készítsünk egy programot, ami értesít minket arról, hogyha valamelyik komponensből 50%, vagy annál kevesebb van, és szól, hogy vegyünk belőle annyit amennyi hiányzik az ajánlott mennyiség eléréséhez.

1. Módosítsátok az 57-es feladat megoldását úgy, hogy a könyvek egy books.txt-ből jöjjenek az előre definiált arraylist helyett, valamint a megoldás is egy booksCount.txt-be kerüljön.
2. Módosítsátok az 58-as feladat megoldását úgy, hogy az éttermek egy restaurants.txt-ből jöjjenek az előre definiált arraylist helyett, valamint a megoldás is egy restaurantsVoteCount.txt-be kerüljön.
3. A task64 megoldása alapján bővítsétek ki a számológép működését, hogy amit a felhasználó számol és annak eredménye kerüljön elmentésre egy naplófájlként. *(calculator.log)* néven
4. Készítsetek egy olyan alkalmazást, amely egy fájlból felolvas különböző háromszögek lehetséges értékeit. Minden egyes sor egy lehetséges háromszög oldalait tartalmazza. (triangleExamples.txt). A program végig iterál a listán ami a fájlban van és minden egyes sorról eldönti hogy a háromszög szerkeszthető-e és ki is írja a képernyőre. Minimum minden olyan esetet vegyetek fel mint példa háromszög, ami szerintetek szükség van, hogy megfelelően le legyen tesztelve a logika. Az eredmény kerüljön lementésre egy fájlba (t*riangleResults.log*).
5. Készítsünk egy reportoló alkalmazást, ami a 75-s feladat megoldásán alapul. Tegyük fel, hogy a teszt futások eredményei egy testexecutionresults.txt-ben vannak tárolva. Olvassuk be a fájl (minimum 20 sor szerepeljen benne), dolgozzuk fel és az eredményt egy testexecutionreport.txt-be mentsük el.

## ------Date/Time feladatok------

1. (Órai) Szombaton focimeccs. Ebben a feladatban mi vagyunk a középiskolások focimeccseit összeállító vezető testület egyik tagja. Az a döntés született, hogy a bajnokság megnyitója 2023. január 07.-ére került, a bajnokság meccseire pedig ezt követően minden szombaton kerül sor az elkövetkezendő 12 hétben. A feladat, hogy írjunk egy programot, amit meghatározza milyen dátumokra esnek ezek a szombati napok. A dátumokat írjuk ki a konzolra.
2. Készítsetek egy programot amely kiírja az aktuális napot és még 10 darab napot, melyek az ezt követő hónapban vannak. (Tehát minden egyes kiírt nap pontosan egy hónappal későbbi az előző dátumnál)
3. (Órai) Érkezés munkába. Az egyik nagyvállalat, aki feljegyzi, hogy a dolgozói mikorra járnak be dolgozni és hogy mikor távoznak szeretne egy elemzést készíteni. Az elemzéshez összegyűjtötték a fejlesztők reggeli érkezésére vonatkozó adatokat egy listába. A feladatunk, hogy írjunk egy programot, ami kiszámítja, hogy a fejlesztők hány százaléka ment be az adott napon a 7-8, 8-9, 9-10, 10-11 időszakban.

A beérkezési időpontok:

| List<String> logs = Arrays.asList(  "07:38", "07:53", "07:17", "07:04", "07:48", "07:36", "07:52",  "07:12", "07:43", "07:45", "08:11", "07:53", "08:32", "07:44",  "07:19", "07:54", "07:36", "08:39", "07:22", "08:58", "07:23",  "07:40", "07:22", "07:06", "07:56", "07:26", "08:09", "07:30",  "07:51", "08:49", "08:41", "07:56", "07:12", "07:15", "07:47",  "07:14", "08:40", "07:50", "07:06", "07:39", "08:16", "07:26",  "07:56", "08:52", "07:24", "07:22", "07:05", "08:32", "07:32",  "07:54", "07:04", "07:13", "07:07", "08:55", "07:14", "07:48",  "08:19", "08:53", "08:50", "08:32", "07:47", "07:48", "08:41",  "08:15", "07:20", "07:29", "08:39", "07:58", "07:17", "07:18" ); |
| --- |

1. Készítsetek összesítő alkalmazást, ami megszámolja hogy az adott nap hányszor volt valamilyen rendszerleállás az egyik microservice kapcsán. Az adott napi leállásokat sikerült a monitoring alkalmazásból kinyerni és egy listába elmenteni a napra vonatkozóan:

| List<String> logs = Arrays.asList(  "07:17", "07:04", "07:48", "07:36", "07:52",  "07:12", "07:43", "07:45", "08:11", "07:53", "08:32", "07:44",  "07:51", "08:49", "08:41", "07:56", "07:12", "07:15", "07:47",  "07:56", "08:52", "07:24", "07:22", "07:05", "08:32", "07:32",  "07:54", "07:04", "17:13", "17:07", "18:55", "07:14", "07:48",  "08:19", "08:53", "08:50", "08:32", "07:47", "07:48", "08:41" ); |
| --- |

Az összleállás mellett, fontos hogy mennyi leállás volt délelőtt (12:00) és mennyi délután.

1. Készítsetek egy alkalmazást, amely a konzolról bekért dátumot leellenőrzi és ha érvényes, valid dátum akkor írj ki hogy “Valid datetime.”, egyébként pedig hogy “Invalid datetime.”
2. (Órai) Hozzunk létre egy LocalDateTime típusú birthday változót, melyben elmentjünk a születésnapunkat dátum + idő formátumban. Írjunk programot, mely kiírja hogy a következő 5 évben milyen napra fog esni a születésnapunk
3. Készítsetek programot, mely képes arra, hogy a console-ról Stringként beolvassa az adataitokat és letárolja egy Person nevű DOJO osztályban azokat. Minimum *keresztnév, vezetéknév, születési hely*, (String-ként) és *születési idő* (LocalDate-ként), valamint *kedvenc filmek listája* (List<String>-ként), *boldog-e* (boolean), *testmagasság* (int-ként) kerüljön letárolásra private osztályváltozók segítségével. Ezen, példányosításkor kötelezően megadandó attribútumok pedig konstruktor segítségével kapjanak értéket. E mellett adjatok hozzá legalább **2 tetszőleges opcionális attribútumot** az osztályhoz, melyet példányosítást követően a **setter** metódusokkal (ha akarunk), be lehet állítani. Készítsetek egy **toString** metódust is, mely visszaadja az összes beállított tulajdonságot, és írassátok ki vele a konzolra a lementett adatokat.
   1. Készüljön el a fenti követelmény.
   2. Refactoráljátok úgy a kódot, hogy az adatok beolvasása egy metódusban történik, mely a String-ként beolvasott elemeket egy List-be pakolja és a metódus ezzel a listával tér vissza. A példányosításkor pedig a konstruktor ezen lista elemeit kapja meg és kerüljön neki átadásra. Pl.: *getUserInputList()*
   3. Refaktoráljátok úgy a kódot, hogy a példányosítás is külön metódusban történik, amely medódus megkapja paraméterként a beolvasott értékeket tartalmazó listát, elvégzi a példányosítást és visszatérési értéke ez a létrehozott *Person* típusú példány. Pl.: *getPerson()* A main metódusban így a két metódushívás szükségeltetik, illetve egy utasítás, mely kiírja ezt a létrehozott példányt a konzolra.
4. Készíts egy alkalmazást, amely folyamatosan egymás alá kiírja az aktuális timestamp-et, az aktuális időt, miliszekundumban. (szakítsuk meg a program futását a 100. alkalom után)

## ------Enum feladatok------

1. (Órai) Készítsünk egy alkalmazást, mely az előre definiált hét napjait tartalmazó enumot felolvassa és visszaadja a lehetséges értékeit
2. (Órai) Készítsünk egy alkalmazást, mely előre definiált vizsgához tartozó értékelési PASSED, FAILED értékeket hoz létre enumként. A program a konzolról bekéri a felhasználótól az elért pontszámot, és a parancssori argumentumként megadott köszöbérték vagy a feletti érték esetén kiírja a felhasználónak, hogy PASSED, egyéb esetben pedig hogy FAILED.
   1. Finomhangoljuk az alkalmazásunk tovább úgy, hogy az enum értékei esetén egy konszolidáltabb, nem csak az enum értékét tartalmazó szöveget írunk ki a képernyőre.
3. Készítsetek egy OrderStatus nevű enumot, mely ACCEPTED, PAYED, TRANSIT, DELIVERED értékeket tartalmaz. A program indítása után kiírja, hogy “Your order is currently … “ A … helyén pedig legyen a fenti enumok értéke, például “Your order is currently ACCEPTED and waiting for the payment.” Az pedig, hogy a fenti értékek közül melyiket írja ki a program véletlenszerűen dőljön el. Egy random generált 1-4 közötti szám döntse el a státuszt. Pl.: 1 esetén ACCEPTED
4. (Órai) Készítsünk egy alkalmazást, mely az aktuális napról eldönti, hogy az italos nap-e vagy sem. A napok vizsgálatához használjunk switch-case-t.
5. Készítsetek programot, mely a beírt, magyar nyelvű hónapról eldönti, hogy az melyik évszakban van. Készítsétek el a Month enumot, melyben a hónapok magyar nyelvű megfelelői vannak. A program vizsgálja meg switch-case segítségével a konzolról beírt értéket és írja ki, hogy az tavasz, nyár, ősz vagy téli évszakba tartozik.

# Linkek

IDE: IntelliJ IDEA Community Edition: <https://www.jetbrains.com/idea/download/#section=windows>

Java gyakorló oldal: <https://www.codecademy.com/learn/learn-java>

Array mit ír ki a program feladatok és megoldások: <https://www.cs.uwlax.edu/notable/credit-by-exam/array-solutions.pdf>