

# Programozási alapok Java nyelven

Java - BGN



Training360 Kft.  
2019.10.01.

# TEMATIKA

---

- Szoftverfejlesztés menete
- Java program készítése
- A Java nyelv alapjai
- Változók, típus, literál
- Osztályok készítése
- Utasítások, kifejezések, vezérlési szerkezetek
- Tömb, lista
- Gyakori algoritmusok

# A szoftverfejlesztés menete

---




# PROGRAMOZÁS CÉLJA

---

- Problémák automatizált megoldása számítógép segítségével
  - Matematikai
  - Műszaki
  - Nyelvi
  - Képi

# A PROGRAMOZÁS LÉPÉSEI

---

- Feladat megfogalmazása, követelmények feltárása
  - Specifikáció
  - Algoritmus
  - Kódolás
  - Tesztelés
  - Dokumentálás
- 
- Hibajavítás, hatékonyságjavítás

# KÖVETELMÉNYEK FELTÁRÁSA

---

- Feladat szabatos megfogalmazása
- Követelmények:
  - Funkcionális
  - Nem funkcionális
  - Szakterületi követelmények (funkcionális és nem funkcionális is)

# FELADAT SPECIFIKÁCIÓ

---

- A probléma precíz megfogalmazása
  - Bemenet + kimenet + előfeltétel + utófeltétel
- Készülhet
  - Szabatos nyelven
  - Matematika megfogalmazás
  - Egyezményes grafikus nyelven (UML)

# ALGORITMUS

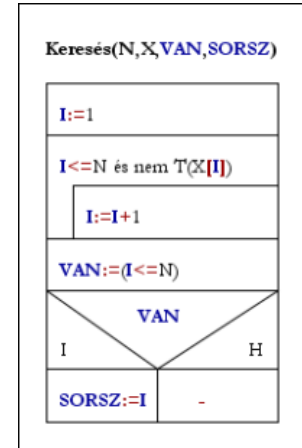
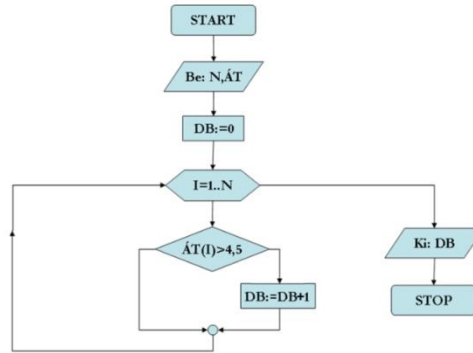
---

- A feladat megoldásához vezető egyértelmű lépések véges sorozata.
- Lépés → adat + utasítás
- Egyértelmű → mindenki számára ugyanazt az információt hordozza
- Véges → ha nincs vége, nem vezet megoldáshoz

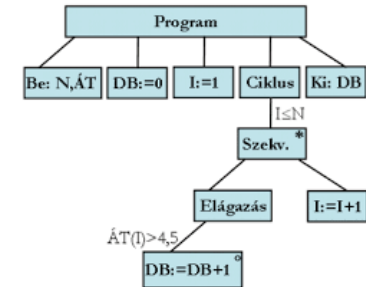


# ALGORITMUS MEGFOGALMAZÁSA

- Folyamatábra
- Stuktogram
- Pszeudokód
- Jackson-diagram



```
eljárás LNKO
be: A, B
ciklus, amíg A ≠ B
  ha A > B, akkor
    A = A - B
  különben
    B = B - A    // harmadik eset nincs
  elágazás vége
ciklus vége
ki: A
eljárás vége
```



# KÓDOLÁS

---

- Program: az algoritmus számítógép által értett megfogalmazása
- Számítógép által értett nyelv: processzor által ismert utasítások (gépi kód)
- Programozási nyelvek
  - Alacsony szintű: ASSEMBLY
  - Magas szintű: C#

# PROGRAMOZÁSI NYELVEK CSOPORTOSÍTÁSA

---

- Szintje szerint:
  - Alacsony szintű
  - Magas szintű
- Típusok kezelése szerint:
  - Gyengén típusos
  - Erősen típusos

# PROGRAMOZÁSI NYELVEK CSOPORTOSÍTÁSA 2

---

- Nyelvi szerkezet szerint:
  - Deklaratív
  - Imperatív
  - Funkcionális
- Hangsúly szerint
  - Procedurális
  - Objektorientált

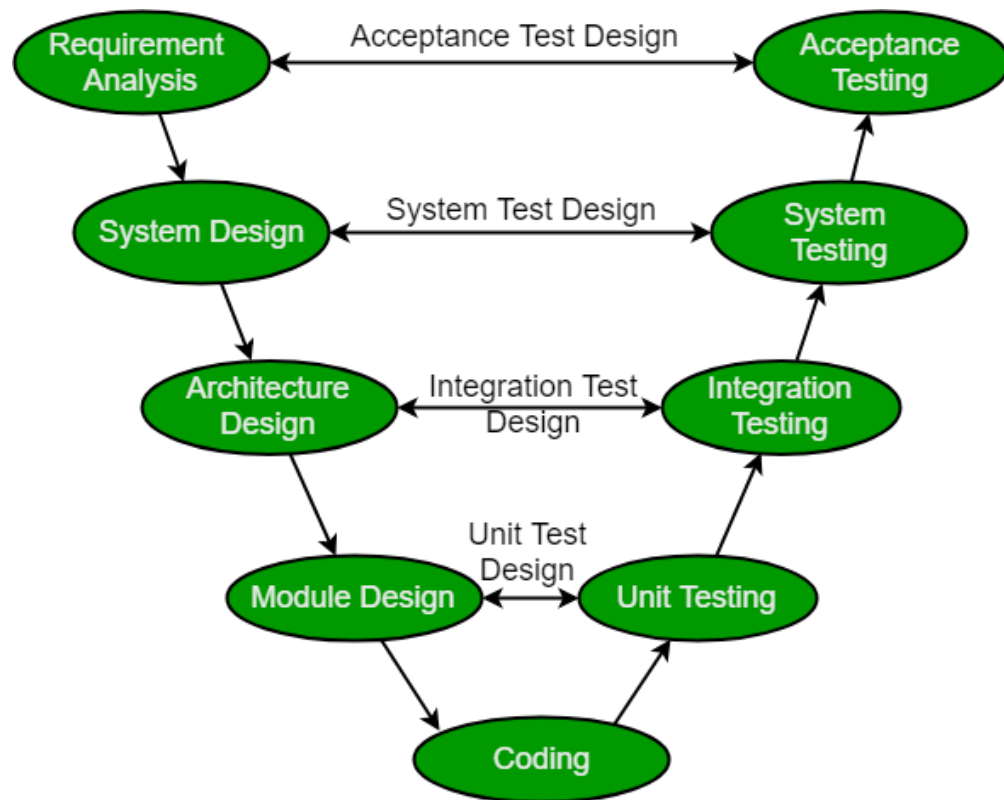
# PROGRAMOZÁSI NYELVEK CSOPORTOSÍTÁSA 3

---

- Környezethez való viszonya szerint:
  - Függő
  - Független
- Párhuzamosság szerint
  - Egyszálú
  - Többszálú

# TESZTELÉS

---

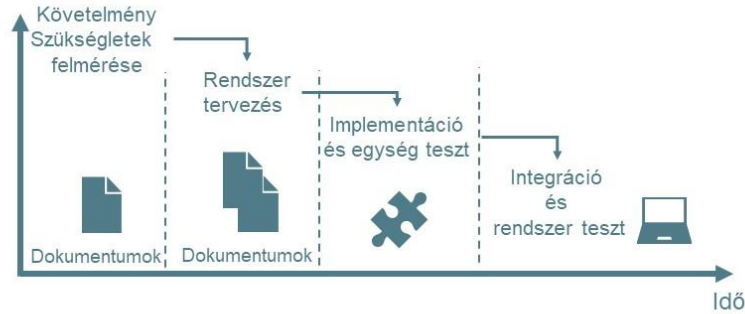


# DOKUMENTÁLÁS

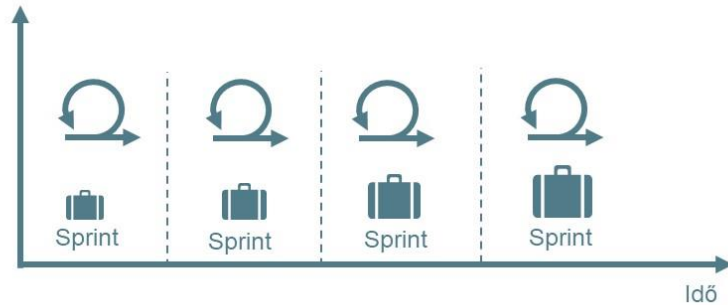
---

- Követelmény és tervdokumentáció
  - Magas szintű
  - Részletes
- Teszt dokumentáció
  - Terv
  - Eredmény
- Rendszer dokumentáció
- Telepítési útmutató
- Felhasználói dokumentáció

# SZOFTVERFEJLESZTÉSI MÓDSZEREK



Vizesés



Agilis



# A Java program

---

TRAINING360

# JAVA TOTÓ

---

- Alacsony szintű vagy magas szintű nyelv?
- Gyengén vagy erősen típusos?
- Deklaratív, imperatív vagy funkcionális?
- Procedurális vagy objektumorientált?
- Környezetfüggő vagy környezetfüggetlen?
- Egyszálú vagy többszálú?

# JAVA TÖRTÉNETE

---

- Szoftverkrízis
- Új programozási paradigmák: objektumorientált programozás
- 1991 Sun Microsystems titkos projektje, James Gosling
- Első Java: Mosaic böngészőben

# CÉLOK

---

- Egyszerű, objektumorientált
- Robusztus, biztonságos
- Architektúra-semleges, hordozható
- Nagy teljesítményű
- Interpretált, többszálú és dinamikus

# JELLEN

---

- Oracle
- Főleg nagyvállalati backend rendszerek
- Beágyazott rendszerek
- Mobil: Android (a Kotlin alapja)
- Kliens oldal: Swing, JavaFX  
(Java 11-ben már nincs benne)
- 10 millió Java fejlesztő, 15 milliárd eszköz

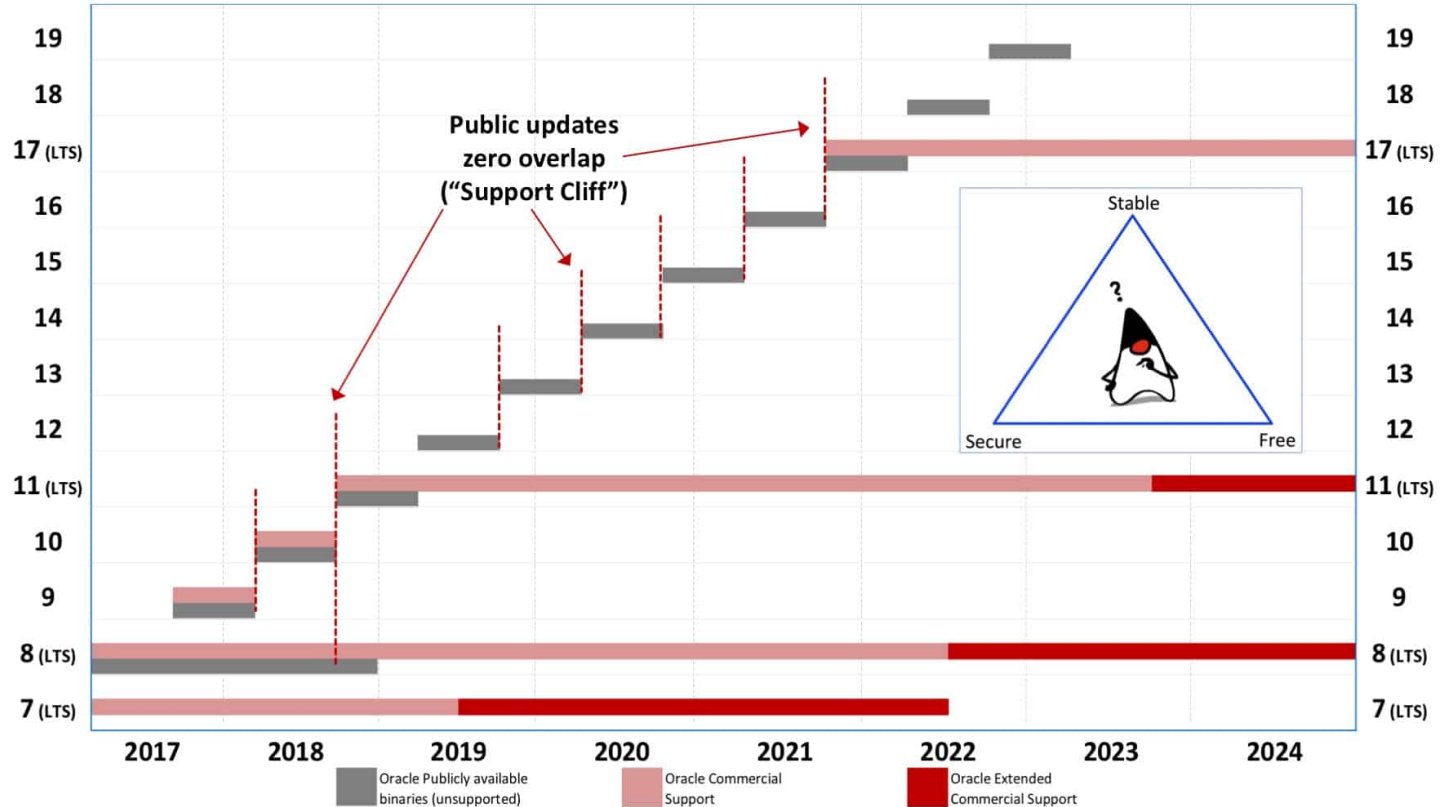
# VERZIÓSZÁMOK

---

- Java SE 8u172 (1.8.0\_172-b11)
- Hétköznapien: Java 8
- Már a Java 12-nél járunk
  - A folyamatos hivatalos frissítések csak fizetés ellenében elérhetőek az éles rendszereken.
  - Fejlesztésre, tesztelésre és végfelhasználóknak ingyenes.
  - Cserében fél évente új verzió jön.

# Java SE Lifecycle – 5+ Year Timeline

Java SE Version



# MI KELL A JAVÁHOZ

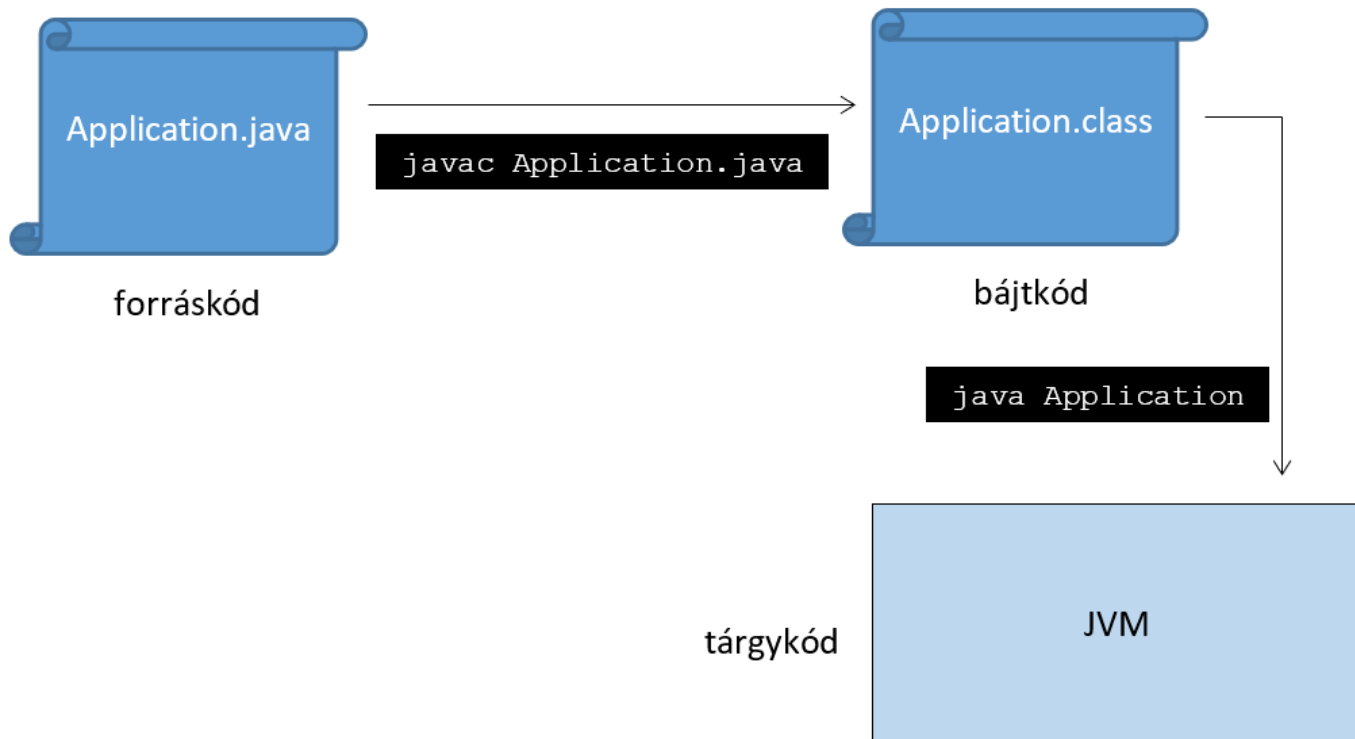
---

- Futtatáshoz:
  - JRE (Java Runtime Enviroment) (Java 11 óta nincs külön)
- Fejlesztéshez:
  - JDK (Java Development Kit)
  - JRE + Tools (pl. compiler)
  - Szövegszerkesztő
  - Jó ha van: IDE



# FEJLESZTÉS MENETE

---



# JAVA PROGRAM ÍRÁSA

---

- Forráskód:
  - Egyszerű szöveg
  - A fájl neve .java kiterjesztésű és megegyezik a benne található osztály nevével.

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

# JAVA PROGRAM FORDÍTÁSA

---

- Fordítás:
  - javac paranccsal
  - Csak akkor fordítható, ha a fájl neve és a benne található osztály neve megegyezik!

```
javac Application.java
```

# JAVA PROGRAM FUTTATÁSA

---

- Futtatás:
  - java paranccsal
  - Csak akkor futtatható, ha van benne **main** metódus!

```
java Application
```

# JAVA ALKALMAZÁS

---

- Sok-sok osztályból áll
- Egymással kommunikáló objektumok összessége
- Terjesztéshez 1 fájlba kell csomagolni: kiterjesztése .jar
- Elég 1 futtatható osztály bele

# BUILD FOLYAMAT

---

- Forrás állományok fordítása
- Többi, un. erőforrás állomány kezelése
- Teszt esetek futtatása
- Alkalmazás összecsomagolása

# MAVEN

---

- Nincs Java platformon standard projekt struktúra
- Segít a build folyamatban
- Függőségkezelés

# MAVEN PROJEKTSTRUKTÚRA

---

- **pom.xml:** projekt leíró állomány
- **src\main\java:** Java forráskódok
- **src\main\resources:** Erőforrás állományok
- **src\test\java:** Teszt esetek, nem része az alkalmazásnak
- **src\test\resources:** Teszt esetekhez szükséges egyéb erőforrás állományok, nem része az alkalmazásnak



# POM.XML FELÉPÍTÉSE

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.training360.yellowroad</groupId>
  <artifactId>formatnumberformat</artifactId>
  <version>1.0-SNAPSHOT</version>

  <packaging>jar</packaging>

  <name>${project.artifactId}</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>12</maven.compiler.source>
    <maven.compiler.target>12</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

} projekt azonosítói

karakterkódolás  
Java verzió

} külső függőségek

# A Java nyelv alapjai

---

TRAINING360

# PROGRAMOZÁSI ALAPFOGALMAK

---

- **Adat:** az információ hordozója
- **Utasítás:** az adat létrehozását, átalakítását, közlését vagy törlését szolgáló, vagy a programot vezérlő egyértelmű folyamat leírása
- **Változó:** az adat tárolására szolgáló névvel ellátott memóriaterület, tartalma módosítható
- **Szubrutin:** utasítások névvel ellátott sorozata
  - eljárás: nincs visszatérési értéke
  - függvény: van visszatérési értéke

# JAVA ALAPFOGALMAK

---

- Legkisebb építőkö: osztály
- **Osztály**: az adatok és a rajtuk dolgozó szubrutinok egységbe foglalt terve (modell: tulajdonságok + műveletek)
  - attribútum
  - konstruktor
  - metódus
- **Csomag**: az osztályok csoportosítása alkalmazáson belül

# JAVA NYELV JELLEMZŐI

---

- Kis- és nagybetű érzékeny
- Karakterkészlete: Unicode
- Foglalt szavak
- Objektumorientált, de vannak elemi adattípusok (primitív típusok)
- Kódolási konvenciók:  
<http://www.oracle.com/technetwork/java/codeconvt-136057.html>

# VÁLTOZÓK

---

- **Attribútumok:** az osztály tulajdonságai
- **Lokális változók:** metóduson belül deklarált változók
- Létrehozása: típus név
- Értékadás: változó = érték

```
int number;  
number = 5;
```

# ATRIBÚTUM

---

```
public class Person {  
    String name;  
    int age;  
}
```

# LOKÁLIS VÁLTOZÓ

---

```
public class Application {  
  
    public static void main(String[] args) {  
        int number = 6;  
        System.out.println(number);  
    }  
  
}
```



# SCOPE

---

- A deklarált változó hol érhető el?
  - Lokális változó:
    - csak a deklarációtól a deklarációt tartalmazó blokk végéig
  - Attribútum:
    - Osztályon belül bárhol
    - Máshonnan: alapértelmezetten azonos csomagban lévő osztályokból, de szabályozható módosító szóval

# ADATTÍPUSOK

---

- Primitív típusok
  - byte
  - short
  - int
  - long
  - char
  - float
  - double
- Referencia, ami egy objektumra hivatkozik

# LÉTEZŐ OSZTÁLYOK HASZNÁLATA

---

- Beépített osztálykönyvtár (Java API)
- A használni kívánt osztályokat importálni kell, kivéve a java.lang csomagban lévőket.
- Az osztály adata vagy metódusa:
  - közvetlen használható (`System.out.println()`)
  - objektum létrehozása után használható

# OBJEKTUMOK

---

- Osztály = tervrajz
- **Objektum** = az osztály egy legyártott példánya
- Példányosítás (gyártás): **new** operátor
- Az osztályban deklarált attribútumok és metódusok elérése: **.** operátor

# OBJEKTUM PÉLDA

---

```
public class Application {  
  
    public static void main(String[] args) {  
        Person john = new Person();  
        Person jane = new Person();  
        john.name = "John Doe";  
        john.age = 23;  
        jane.name = "Jane Doe";  
        jane.age = 21;  
    }  
}
```

# KOMMUNIKÁCIÓ A FELHASZNÁLÓVAL

---

- Kiírás:
  - `System.out.print()`
  - `System.out.println()`
- Beolvasás:
  - Scanner objektum metódusaival
    - `nextInt()`
    - `nextLine()`

# KOMMUNIKÁCIÓ PÉLDA

---

```
public class Application {  
  
    public static void main(String[] args) {  
        Person person = new Person();  
        Scanner scanner = new Scanner(System.in);  
  
        person.name = scanner.nextLine();  
        person.age = scanner.nextInt();  
  
        System.out.println(person.name + " " + person.age);  
    }  
}
```

# Osztályok készítése

---

TRAINING360



# OSZTÁLY

---

- Osztály  $\neq$  objektum
- **class** kulcsszó
- Tagok
  - attribútum
  - konstruktor
  - metódus

# TAGOK LÁTHATÓSÁGA

---

- **private**: csak az adott osztályon belül látható
- **(package private)**: azonos csomagban lévő más osztályokból is látható
- **public**: mindenhol látható

# ATTRIBÚTUM

---

- Az objektum tulajdonságait tárolja
- **Information hiding** alapelv: ne lehessen közvetlenül elérni kívülről!
- Láthatósági módosító előtte: `private`
- **Állapot:** az objektum attribútumainak pillanatnyi értéke

```
public class Person {  
  
    private String name;  
  
    private int age;  
  
}
```

# KONSTRUKTOR


---

- Az osztály példányosítása során fut le
- Feladata az attribútumok inicializálása (kezdőérték adása)
- A **new** operátor után a konstruktort hívjuk meg
- A konstruktor neve megegyezik az osztály nevével, láthatósága általában `public`
- Kívülről kaphat paramétereket

# KONSTRUKTOR PÉLDA

---

```
public class Person {  
  
    private String name;  
  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
}
```



# DEFAULT KONSTRUKTOR

---

- Minden osztálynak van konstruktora
- Ha mi nem írunk, akkor a JVM ad egy paraméter nélküli konstruktort → default konstruktor

```
public class Person {  
    private String name;  
  
    private int age;  
  
    public Person(){}  
}
```

# METÓDUS

---

- A metódus utasítások névvel ellátott véges sorozata.
- **Paraméterek:** bemenő adatok
- **Visszatérési érték:** kijövő adat  
Javában mindig meg kell adni, ha nincs, akkor void
- **Törzs:** a metódusban lévő utasítások sora
- Feladata: az objektum állapota és a bemenő adatok alapján a kimenő adat előállítás vagy az objektum állapotának módosítása

# METÓDUS 2

---

- Speciális metódusok:
  - **Getter**: attribútum értékét lekérdező metódus
  - **Setter**: attribútum értékét módosító metódus
- Deklarációja:





# METÓDUS PÉLDA

---

```
public class Person {  
    private String name;  
  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String newName) {  
        name = newName;  
    }  
}
```

# PARAMÉTERÁTADÁS

---

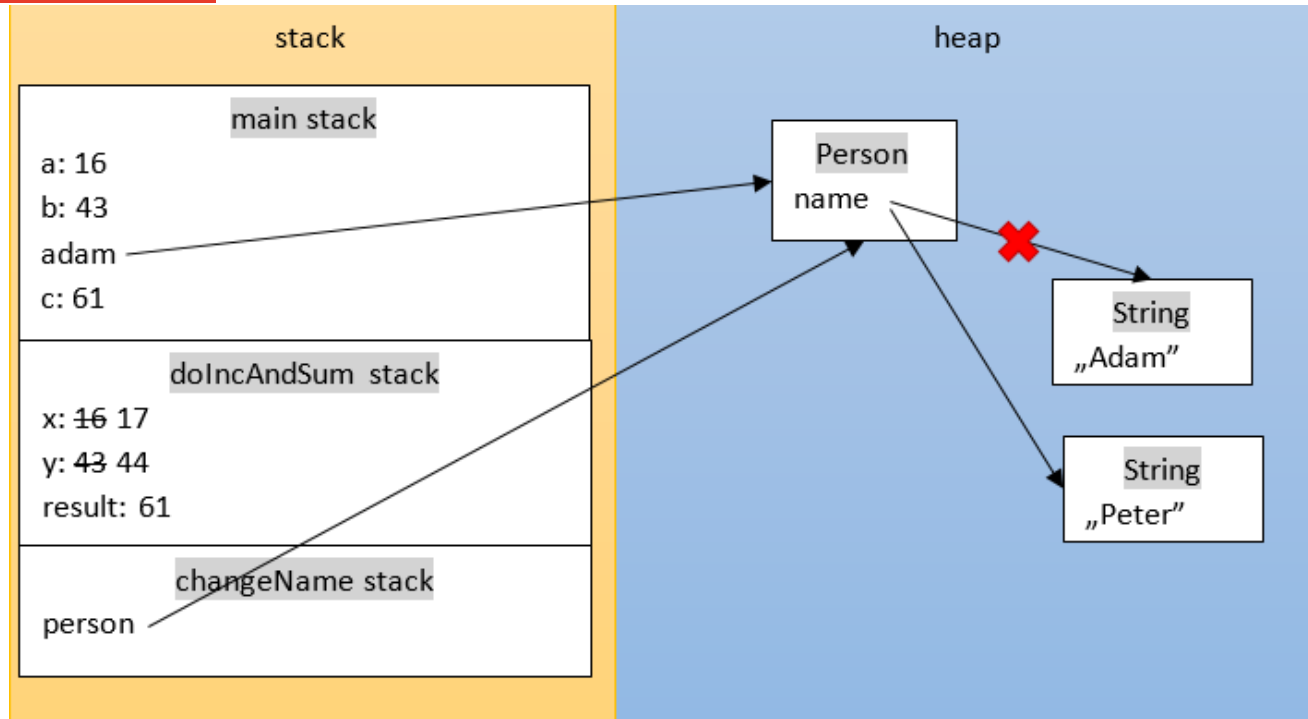
- **Formális paraméter:** a metódus deklarációjában lévő paraméter  
`public void setName(String name) {`
- **Aktuális paraméter:** a metódus hívásakor átadott érték  
`person.setName("Adam");`
- Értékmásolás szerint
- Típusegyeztetés
- Sorrendi kötés

# ADATOK A MEMÓRIÁBAN

---

```
public class Main {  
  
    public static void main(String[] args){  
        int a = 16;  
        int b = 43;  
        Person adam = new Person("Adam");  
        int c = doIncAndSum(a, b);  
        System.out.println(a); //16  
        System.out.println(b); //43  
        System.out.println(c); //61  
        System.out.println(adam.getName()); //"Adam"  
        changeName(adam);  
        System.out.println(adam.getName()); //"Peter"  
    }  
  
    public static int doIncAndSum(int x, int y){  
        x++;  
        y++;  
    }  
}
```

# ADATOK A MEMÓRIÁBAN 2



# METÓDUS TÚLTERHELÉS

---

- Ugyanolyan funkciójú metódus ugyanolyan névvel, de különböző paraméterekkel
- A visszatérési érték is eltérhet
- Gyakori konstruktoroknál
- Gyakori általános célú metódusoknál
- Példa: `System.out.println();`

# DESTRUKTOR

---

- Javában NINCS
- Garbage Collector: automatikus szemétgyűjtés

# Java utasítások

---

TRAINING360

# UTASÍTÁSOK TÍPUSAI

---

- deklaráció
- értékadás
- vezérlő utasítás
- metódushívás



# KIFEJEZÉSEK

---

- Operátorok (műveleti jelek) és operandusok (azok a literálok, változók vagy metódushívások, amelyekkel a műveletet elvégezzük) kombinációja, tipikusan egy érték kiszámítására.

```
int a = 3;  
int b = 4 * a;  
int x = (a + b) / 2;
```

# OPERÁTOROK

---

- Operátorok:

**+**, **-**, **\***, **/**, **%**

**new**

**<**, **>**, **<=**, **>=**, **==**, **!=**

**!**, **&&**, **||**, **&**, **|**

**++**, **--**

**?:**

**=**

# VEZÉRLŐ UTASÍTÁSOK

---

- Szekvencia
- Szelekció
- Iteráció

# SZELEKCIÓ (ELÁGAZÁS) 1

---

```
if(feltétel) {  
    utasítás;  
}
```

```
if(feltétel) {  
    utasítás1;  
} else {  
    utasítás2;  
}
```

# SZELEKCIÓ (ELÁGAZÁS) 2

---

```
if(feltétel1) {  
    utasítás1;  
} else if (feltétel2){  
    utasítás2;  
  
} ...  
} else {  
    utasításn;  
}
```

# SZELEKCIÓ (ELÁGAZÁS) 3

---

```
switch(kifejezés) {  
    case érték1:  
        utasítás1; break;  
    case érték2:  
        utasítás2; break;  
    ...  
    default:  
        utasításn;  
}
```

# ITERÁCIÓ (CIKLUS) 1

---

```
for(init; feltétel; léptetés) {  
    utasítás;  
}
```

```
while(feltétel) {  
    utasítás;  
}
```

# ITERÁCIÓ (CIKLUS) 2

---

```
do {  
    utasítás;  
} while(feltétel);
```

break – a ciklust követő utasításra ugrik

continue – a következő végrehajtási ciklusba ugrik



# Tömb, lista

---

TRAINING360

# TÖMB

---

- Több ugyanolyan típusú adat sorozata, amelyben minden elem elérhető közvetlenül a sorszámaival (index)
- Index 0-tól kezdődik
- Deklaráció: `int[] numberArray;`
- Létrehozás: `numberArray = new int[10];`

# TÖMB 2

---

- Literálja: {elem1, elem2, elem3, ...}
- Hossza: length

```
int[] numbers = new int[3];  
for (int i = 0; i < numbers.length; i++) {  
    numbers[i] = scanner.nextInt();  
}
```

# AZ ARRAYS OSZTÁLY

---

- Segédmetódusok tömbök kezeléséhez
  - `toString()`: tömbök kiírása egyben
  - `sort()`: rendezés
  - `copyOf()`, `copyOfRange()`: másolás

```
int[] numbers = new int[3];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = scanner.nextInt();
}
Arrays.sort(numbers);
int[] numbers2 = Arrays.copyOf(numbers, 2);
```

# AZ ARRAYLIST OSZTÁLY

---

- „Dinamikus tömb”
- Elemek sorozata, indexelhető
- Nem kell előre megmondani, hogy hány elem lesz
- Az elemek beszúrása, törlése, lekérdezése metódusokkal történik
- Az elemek típusát <> között kell megadni
- Csak referencia típusú eleme lehet

# CSOMAGOLÓ OSZTÁLYOK

---

- Minden primitív típushoz létezik neki megfelelő osztály
- Köztük a be- és kicsomagolás automatikus

# CSOMAGOLÓ OSZTÁLYOK 2

Primitív típus	Csomagoló osztály
byte	Byte
short	Short
int	Integer
long	Long
char	Character
boolean	Boolean
float	Float
double	Double

```
int a = 5;  
Integer b = a;  
Integer c = 6;  
int d = c;
```

# ARRAYLIST PÉLDA

---

```
List<Integer> numberList = new ArrayList<>();  
for (int i = 0; i < 10; i++) {  
    numberList.add(i);  
}  
  
for(int element: numberList) {  
    System.out.println(element);  
}  
  
System.out.println(numberList);
```



# ARRAYLIST METÓDUSOK

---

- `size()`: lista hossza
- `add()`: elem beszúrása
- `get()`: elem lekérdezése
- `set()`: elem módosítása
- `remove()`: elem törlése
- `clear()`: teljes lista ürítése

# Gyakori algoritmusok

---

TRAINING360

# SOROZATSZÁMÍTÁS

---

Feladat: Mennyi az összes elem függvényértéke?

eredmény := kezdőérték

ciklus amíg van elem

    elem := következő elem

    eredmény := f(eredmény, elem)

ciklus vége

# SOROZATSZÁMÍTÁS PÉLDA

---

```
sum = 0;
for(int element: array) {
    sum = sum + element;
}
```

# MEGSZÁMLÁLÁS

---

Feladat: Hány megfelelő elem van?

db := 0

ciklus amíg van elem

    ha megfelelő(elem), akkor db := db + 1

ciklus vége

# MEGSZÁMLÁLÁS PÉLDA

---

```
count = 0;
for(int element: array) {
    if(element > 0) {
        count++;
    }
}
```

# ELDÖNTÉS

---

Feladat: Van-e megfelelő elem?

van := hamis

ciklus amíg van elem és nem van

elem := következő elem

van := megfelelő(elem)

ciklus vége

# ELDÖNTÉS PÉLDA

---

```
found = false;  
int i = 0;  
while(i < array.length && !found) {  
    found = array[i] > 0;  
    i++;  
}
```



# ELDÖNTÉS HATÉKONYABBAN

---

```
found = false;
for(int element: array) {
    if(element > 0) {
        found = true;
        break;
    }
}
```

# ELDÖNTÉS 2

---

Feladat: Minden elem megfelel-e?

mind := igaz

ciklus amíg van elem és mind

    elem := következő elem

    mind := megfelelő(elem)

ciklus vége

# ELDÖNTÉS 2 PÉLDA

---

```
all = true;
int i = 0;
while(i < array.length && all) {
    all = array[i] > 0;
    i++;
}
```

# ELDÖNTÉS 2 HATÉKONYABBAN

---

```
all = true;
for(int element: array) {
    if(element > 0) {
        all = false;
        break;
    }
}
```

# KERESÉS

---

eredmény := üres

ciklus amíg van elem és nem megfelelő(elem)

elem := következő elem

ciklus vége

ha van elem, akkor eredmény := elem

Az első megfelelőt találja meg, ha van.

# KERESÉS PÉLDA

---

```
found = null; //???  
int i = 0;  
while(i < array.length && array[i] <= 0) {  
    i++;  
}  
if(i < array.length){  
    found = array[i];  
}
```

# KERESÉS HATÉKONYABBAN

---

```
found = null;
for(int element: array) {
    if(element > 0) {
        found = element;
        break;
    }
}
```

# SZÉLSŐÉRTÉK KIVÁLASZTÁS

---

max := első elem

min := első elem

ciklus amíg van elem

    elem := következő elem

    ha elem > max, akkor max := elem

    ha elem < min, akkor min := elem

ciklus vége



# SZÉLSŐÉRTÉK KIVÁLASZTÁS PÉLDA

---

```
max = array[0];
min = array[0];
for (int element: array) {
    if(element > max) {
        max = element;
    }
    if(element < min) {
        min = element;
    }
}
```

# KIVÁLOGATÁS

---

eredmény = üres sorozat

ciklus amíg van elem

elem := következő elem

ha(megfelelő(elem)), akkor  
eredménybe(elem)

ciklus vége

# KIVÁLOGATÁS PÉLDA

---

```
result = new ArrayList<>();  
for(int element: array) {  
    if(element > 0) {  
        result.add(element);  
    }  
}
```