

Java Programozás

1. óra



BITMAN

v: 2020.02.28

Kishaver!

Az informatikai cégeknek szükségük van Rád!

A legrövidebb út ahhoz, hogy programozóként munkát vállalhass a **224 órás** java programozó képzésünkön átvezet.

Teljesen kezdő szintről indulva megtanulod a JAVA SE, Java EE és Android programozás alapjait.

A 4,5 hónapos intenzív tanfolyamon olyan tudást tehetsz magadévá, amivel **rögtön a tanfolyam után elkezdhetsz junior JAVA programozóként dolgozni.**

Ha sikeresen megcsináltad a vizsgákat, mi is beajánlunk partnereinkhez, akik már várják az új programozók érkezését.



Rövid áttekintő az
előadások anyagjaiból

2

A Java nyelv jellemzői

■ Objektum orientált nyelv

- Megvalósítja az OOP alapelveit
- A bonyodalmakat okozó részeket kihagyja,
 - pl: többszörös öröklődés (nehezebb megvalósítás, kód újrafelhasználás)
 - operátor overloading (félreérthető használat, kevésbé olvasható kód)
- Szigorúan OOP (nincsen osztályon kívüli deklaráció, utasítás)

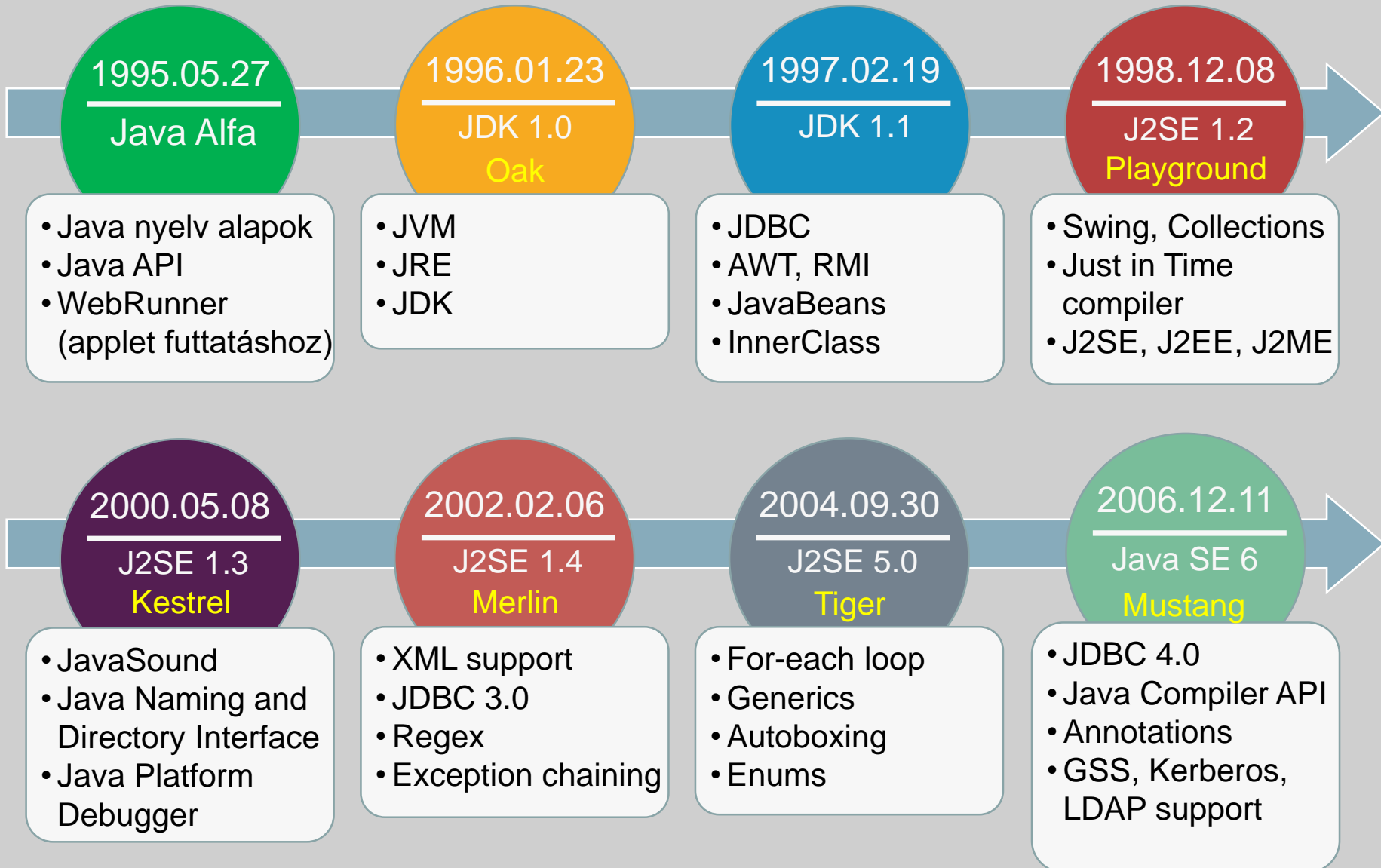
■ Hasonlóság a C++-hoz

- A szintaktika teljesen C++-szerű
- Nincsenek mutatók

■ Gazdag osztály könyvtár (217 csomag)

- Kb. 5000 osztály illetve interfész a JDK 9-ben

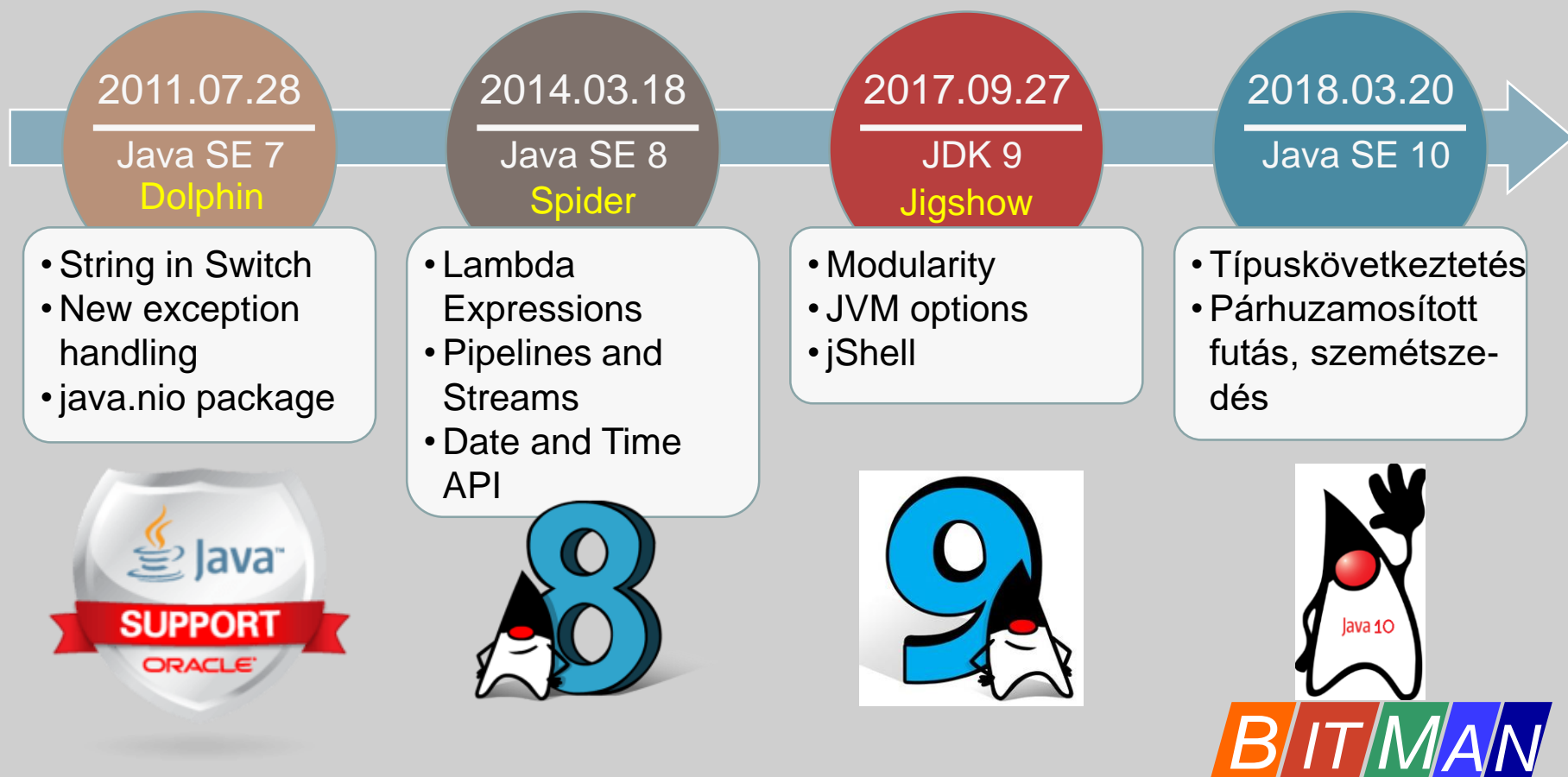
A Java nyelv verziói



A Java nyelv verziói



- 2009.04.20: az Oracle megveszi a Sun Microsystems-t.
- Az ez utáni verziókat az Oracle fejleszti



A Java nyelv verziói

2018.09.25

Java SE 11

- Nem ingyenes üzleti célra!
- Kimarad a vállalati platform
- Unicode 10



2019.03.19

Java SE 12

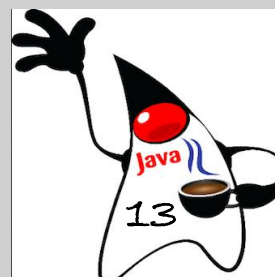
- Sok apró újítás



2019.09.17

Java SE 13

- Sok apró újítás



JDK 9-től:

- Féléves kiadási ciklus
- Csak a fél év alatt támogatott
- Életciklusa alatt két frissítés

Ka



Kávé bébi! Minden class fájl hexa kódja így kezdődik!

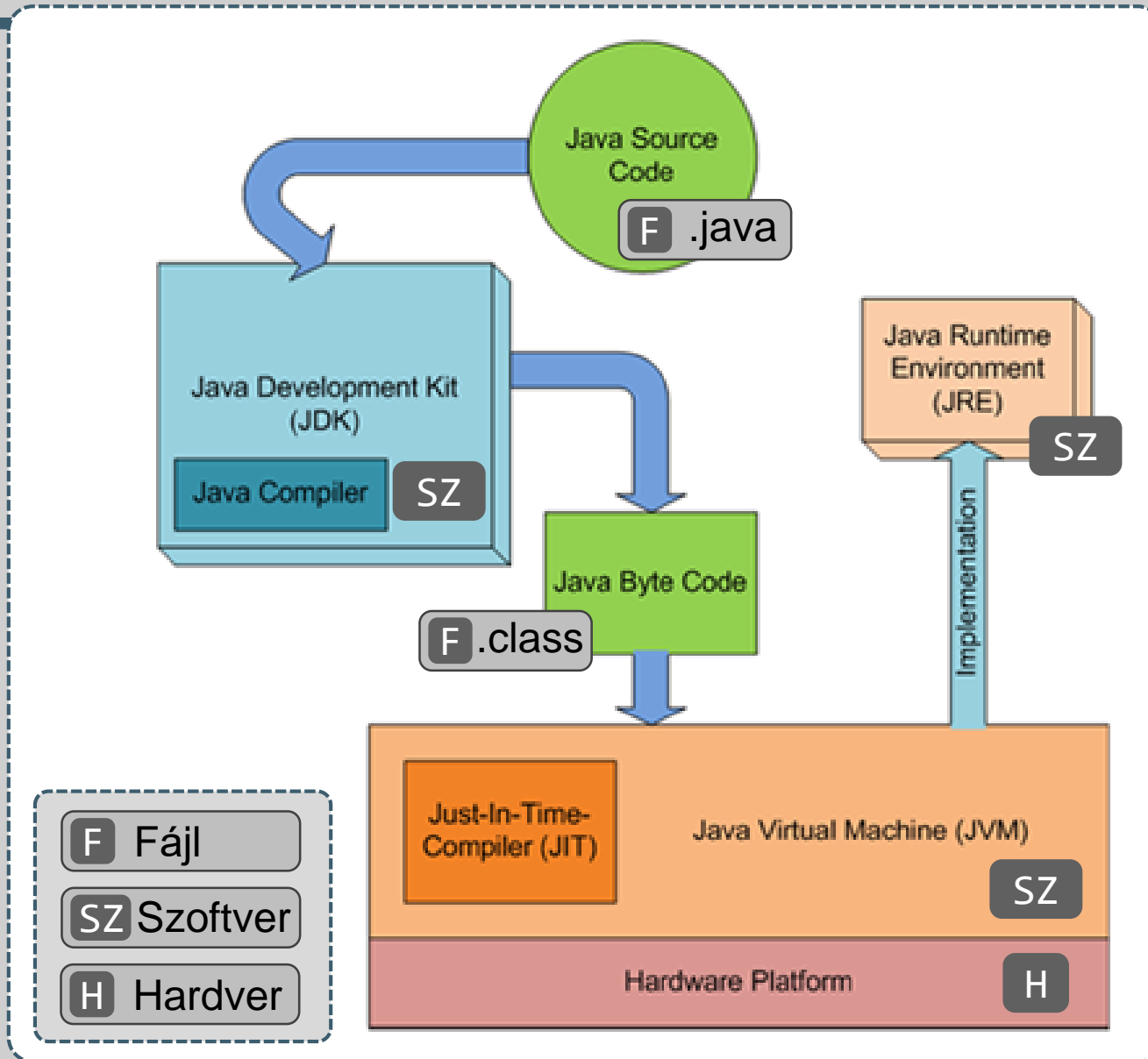
The image shows two windows of a hex editor. The top window is titled 'Lister - [c:\Java_AbKez\JFAlap.class]' and the bottom window is titled 'Lister - [c:\Java_AbKez\Fkezel\Fkezel_1\$1.class]'. Both windows show a hex dump of a file. In both cases, the first four bytes of the file are highlighted with a red box and are 'CA FE BA BE'. The right pane of the bottom window shows the corresponding ASCII representation of the hex data, which includes Java bytecode instructions and class names.

```
00000000: CA FE BA BE 00 00 00 32 | 00 26 0A 00 0A 00 13 0A | EtsI 2 & 0
00000010: 00 00 00 14 00 00 00 00 | 00 00 00 00 00 00 00 00 | 
00000020: 17 07 00 18 00 00 00 00 | 
00000030: 06 00 1A 07 00 00 00 00 | 
00000040: 00 03 28 29 00 00 00 00 | 
00000050: 69 6E 65 4E 00 00 00 00 | 
00000060: 04 6D 61 69 00 00 00 00 | 
00000070: 6C 61 6E 67 00 00 00 00 | 
00000080: 0A 53 6F 75 00 00 00 00 | 
00000090: 41 6C 61 70 00 00 00 00 | 
000000A0: 1C 00 1D 0C 00 00 00 00 | 
000000B0: 62 61 0C 00 00 00 00 00 | 
000000C0: 0C 00 22 00 00 00 00 00 | 
000000D0: 61 78 2F 73 00 00 00 00 | 
000000E0: 00 07 73 65 00 00 00 00 | 
000000F0: 56 01 00 15 00 00 00 00 | 
00000100: 65 6C 61 74 00 00 00 00 | 
00000110: 76 61 2F 61 00 00 00 00 | 
00000120: 3B 29 56 01 00 00 00 00 | 
00000130: 15 28 4C 6A 00 00 00 00 | 
00000140: 69 6E 67 3B 00 00 00 00 | 
00000150: 75 6C 74 43 00 00 00 00 | 
00000160: 6E 01 00 04 00 00 00 00 | 
00000170: 73 69 62 6C 00 00 00 00 | 
00000180: 00 0A 00 00 00 00 00 00 | 
00000190: 00 0D 00 00 00 00 00 00 | 
000001A0: 00 01 2A 11 00 00 00 00 | 
000001B0: 03 2A 12 04 00 00 00 00 | 
000001C0: 00 16 00 05 00 00 00 00 | 
000001D0: 00 13 00 0D 00 00 00 00 | 
000001E0: 00 0D 00 00 00 00 00 00 | 

00000000: CA FE BA BE 00 00 00 32 | 00 67 09 00 1B 00 33 0A | EtsI 2 g 3
00000010: 00 1C 00 22 07 00 34 0A | 00 03 00 32 07 00 35 0A | 2 4 2 5
00000020: 00 31 00 36 0A 00 05 00 | 37 07 00 38 07 00 39 0A | 1 6 7 8 9
00000030: 00 09 00 3A 0A 00 08 00 | 3B 0A 00 08 00 3C 0A 00 | : ; <
00000040: 03 00 3D 0A 00 08 00 3E | 07 00 3F 07 00 40 0A 00 | = > ? @
00000050: 10 00 32 08 00 41 0A 00 | 10 00 42 0A 00 0F 00 43 | 2 A B C
00000060: 0A 00 10 00 44 08 00 45 | 0A 00 46 00 47 07 00 48 | D E F G H
00000070: 0A 00 18 00 49 0A 00 18 | 00 4A 07 00 4B 07 00 4D | I J K M
00000080: 07 00 4E 01 00 06 74 68 | 69 73 24 30 01 00 0A 4C | N this$ L
00000090: 46 6B 65 7A 65 6C 5F 31 | 3B 01 00 06 3C 69 6E 69 | Fkezel_1; <ini
000000A0: 74 3E 01 00 0D 28 4C 46 | 6B 65 7A 65 6C 5F 31 3B | t> (Lfkezel_1;
000000B0: 29 56 01 00 04 43 6F 64 | 65 01 00 0F 4C 69 6E 65 | )U Code Line
000000C0: 4E 75 6D 62 65 72 54 61 | 62 6C 65 01 00 0F 61 63 | NumberTable ac
000000D0: 74 69 6F 6E 50 65 72 66 | 6F 72 6D 65 64 01 00 1F | tionPerformed
000000E0: 28 4C 6A 61 76 61 2F 61 | 77 74 2F 65 76 65 6E 74 | (Ljava/awt/event
000000F0: 2F 41 63 74 69 6F 6E 45 | 76 65 6E 74 3B 29 56 01 | /ActionEvent;)U
00000100: 00 0D 53 74 61 63 6B 4D | 61 70 54 61 62 6C 65 07 | StackMapTable
00000110: 00 4B 07 00 4F 07 00 34 | 07 00 35 07 00 38 07 00 | K 4 5 8
00000120: 50 07 00 3F 01 00 0A 53 | 6F 75 72 63 65 46 69 6C | P ? SourceFil
00000130: 65 01 00 0D 46 6B 65 7A | 65 6C 5F 31 2E 6A 61 76 | e Fkezel_1.jav
00000140: 61 01 00 0F 45 6E 63 6C | 6F 73 69 6E 67 4D 65 74 | a EnclosingMet
00000150: 68 6F 64 07 00 51 0C 00 | 20 00 52 0C 00 1E 00 1F | hod Q R
00000160: 01 00 13 6A 61 76 61 2F | 75 74 69 6C 2F 41 72 72 | java/util/Arr
00000170: 61 79 4C 69 73 74 01 00 | 17 6A 61 76 61 2F 69 6F | ayList java/io
00000180: 2F 46 69 6C 65 49 6E 70 | 75 74 53 74 72 65 61 6D | /FileInputStream
00000190: 0C 00 53 00 54 0C 00 20 | 00 55 01 00 18 6A 61 76 | S T U java
000001A0: 61 2F 69 6F 2F 4C 69 6E | 65 4E 75 6D 62 65 72 52 | a/io/LineNumberR
000001B0: 65 61 64 65 72 01 00 19 | 6A 61 76 61 2F 69 6F 2F | eader java/io/
000001C0: 49 6E 70 75 74 53 74 72 | 65 61 6D 52 65 61 64 65 | InputStreamReade
000001D0: 72 0C 00 20 00 56 0C 00 | 20 00 57 0C 00 58 00 59 | r U W X Y
000001E0: 0C 00 5A 00 5B 0C 00 5C | 00 52 01 00 13 6A 61 76 | Z [ \ R java
```

- Magas szintű programozási nyelv:
 - platform-független
 - általános célú
 - teljesen objektum-orientált
 - egyszerű
 - interpretált (de speciális módon)
 - elosztott (distributed)
 - robusztus
 - biztonságos
 - hordozható
 - többszálú (multithreaded)
 - A C++ ismeretében született, annak formalizmusát tekintette mintának
 - A C++ hátrányait igyekezett kiküszöbölni

Hárombetűsek: JDK, JVM, JRE



A Java program felépítése

■ Forrás file szerkezete:

csomag deklaráció

import deklarációk

osztály, interfész definíciók (ezek közül egy publikus!)

Sorrend fontos!

```
package saját;  
import java.lang.*;  
public class Hello {  
    public static void main(String[ ] args) {  
        System.out.println("Helló világ");  
    }  
}
```

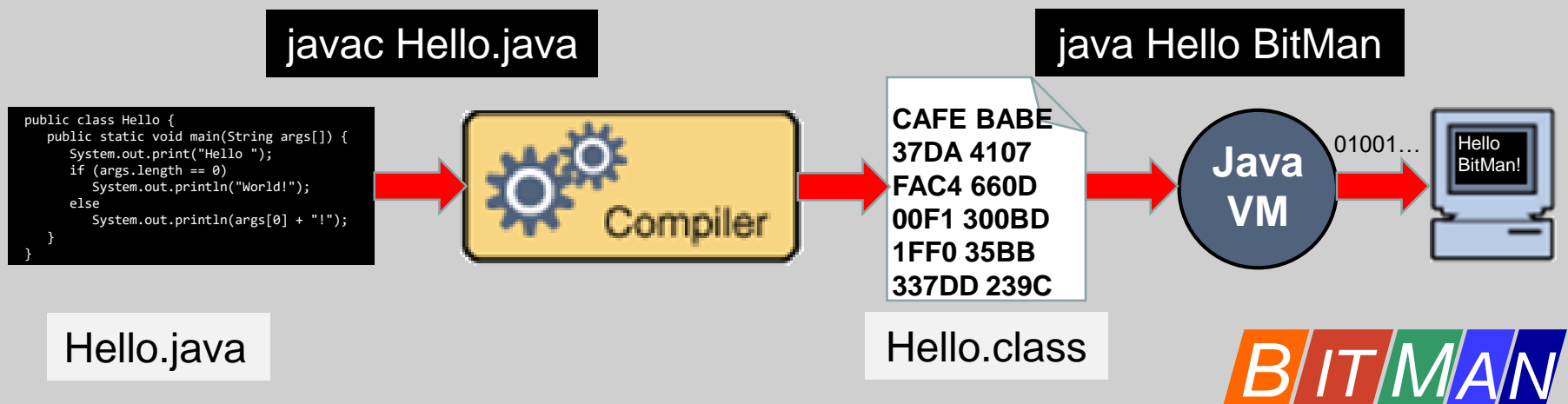
Hello Java program

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.print("Hello ");  
        if (args.length == 0)  
            System.out.println("World!");  
        else  
            System.out.println(args[0] + "!");  
    }  
}
```

Kötelezően egy [Hello.java](#) nevű fájlban kell lennie!

Fejlesztési lépések

1. Forrásfájl (Hello.java) megírása egy tetszőleges szövegszerkesztővel
2. Fordítás: `javac Hello.java`
 - Abban a könyvtárban célszerű kiadni, ahol a forrásfájl van
 - A fordítás eredménye a Hello.class bájtkód
3. Futtatás: `java Hello BitMan`
 - A Hello.class fájlt futtatja

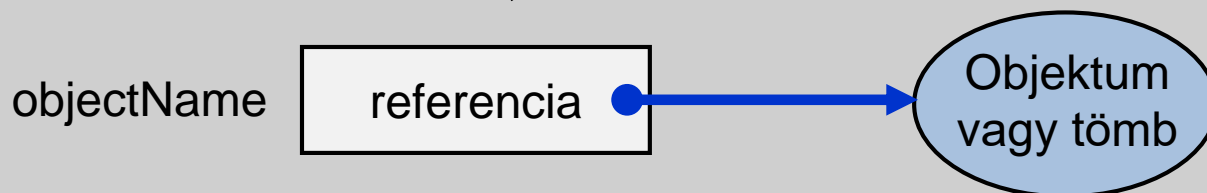


A Java nyelv típusai

■ Referencia típusok

- Osztály
- Interfész
- Speciális szintaktikával rendelkező osztály típusok
 - Tömb
 - Felsorolás (enum)

- A referencia-változó más nyelvek mutató vagy memóriacím fogalmára hasonlít.
- Az objektum neve (objectName) nem egy közvetlen értéket, hanem csak egy referenciát jelent.
- Az értéket közvetetten, a referencián keresztül érhetjük el.



Típuskonverzió

- A Java erősen típusos nyelv, a fordító a típus kompatibilitásának ellenőrzését minden lehetséges esetben elvégzi.
- Egy típus kompatibilis egy másik típussal, ha az adott környezetben helyettesíthető vele.
- Bizonyos műveletek végrehajtása előtt az operandus típusát konvertálni kell egy másik típussá.
- A típuskonverzió kiváltási módja lehet:
 - Implicit (automatikus) – a fordító automatikusan elvégzi
 - Explicit (kényszerített) – a programozónak kell azt kikényszeríteni.
- A konverzió iránya lehet:
 - Szűkítő
 - Bővítő

Típuskonverzió

■ A konverzió iránya

```
int i = 5;  
double d = i;
```

int konvertálása double típusra implicit módon. Ez **bővítés**.

```
int i;  
double d = 79.4;  
i = (int) d;
```

double konvertálása int típusra explicit módon. Ez **szűkítés**.

```
int i;  
double d = 79.4;  
i = d;
```

Szintaktikai hiba!!!

A Java nyelv utasításai

- Blokk
- Lokális változó deklaráció
- Üres utasítás
- Kifejezés utasítás
- Elágazások (if, switch)
- Ciklusok (while, do, for)
- Ugró, vezérlésátadó utasítások (break, continue, return)
- Kivételek kezelésével kapcsolatos utasítások (throw, try-catch)

Szintaktikailag helyes a kód?

IGEN

Ha igen, mit ír ki?

Ha nem, mi a hiba?

1
2
3
4

```
public class Main {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 10; ++i) {  
            if (i == 5) {  
                break;  
            }  
            System.out.println(i);  
        }  
    }  
}
```

```
X  
XXX  
XXXX
```

Mit ír ki a kód?

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            int a = 0;  
            System.out.println ("X");  
            int b = 20 / a;  
            System.out.println ("XX");  
        } catch(ArithmeticException e) {  
            System.out.println ("XXX");  
        } finally {  
            System.out.println ("XXXX");  
        }  
    }  
}
```

2

3

Osztály fogalma

■ A Kör osztály definíciója:

```
public class Kör {  
    double kp_x;  
    double kp_y;  
    double sugár;  
    void eltol(double dx, double dy) {  
        kp_x += dx;  
        kp_y += dy;  
    }  
    void átméretez(double ujsugár) {  
        sugár = ujsugár;  
    }  
}
```

Tulajdonságok: **adattagok**

Viselkedési jellemzők: **metódusok**



Egységbezárás:

Az osztály az adatait és a metódusait egy egységként kezeli, és elzárja őket a külvilágtól.

Példányosítás

- A **Kör** egy típus
- Deklarálhatunk ilyen típusú változókat: `Kör a, b;`
- Létrehozhatunk példányokat belőle, pl: `new Kör();`
- A példányosítás kifejezés eredménye egy megfelelő típusú (példánkban: `Kör`) referencia, amely a létrejött objektumra mutat.
- Ezt a referenciát eltárolhatjuk referencia változókba:

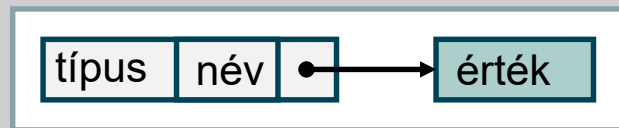
```
Kör a,b;  
a = new Kör();  
b = new Kör();
```


Referencia változó



- Az osztályok referencia típusú változók, ez azt jelenti, hogy értékadásnál nem maga az érték, hanem annak hivatkozása (memóriacíme) rendelődik a változóhoz.
- Az referencia értéke (a cím) a programozó számára hozzáférhetetlen (és szükségtelen is).
- Létezik **null** értékű referencia, ami még (vagy már) nem mutat sehova
- Ha deklarálunk (definiálunk) egy referencia típusú változót, a memóriában lefoglalódik számára hely, de nem jön létre objektum (pl. **Kör k**;)
- Objektum létrehozásához a new operátort kell használni!
 - `Kör a = new Kör();`

Referencia változó



Mit tehetünk a referenciákkal?

- Deklarálhatjuk: `Kör a, b;`

- Értéket adhatunk neki (= operátorral)

```
a = new Kör();
```

- Referencia átadás: `b = a;` **b** ezután ugyanarra az objektumra mutat, mint **a**

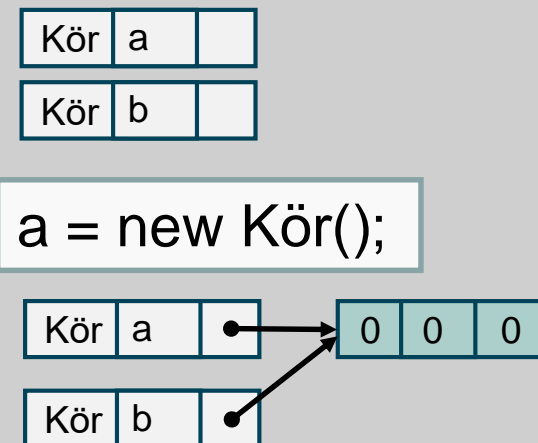
- Két referencia közötti egyenlőséget vizsgálhatjuk (`==`, `!=`)

```
boolean egyenlo = (a == b);
```

- Konvertálhatjuk (később)

- Hivatkozhatunk a tagjaira

```
a.kp_x = 2;  
a.eltol(3, 2);
```



Referencia változó



Deklarált, de nem inicializált referenciával nem lehet semmilyen műveletet végezni, a fordító hibát jelez!

```
public class AutoP {  
    public static void main(String[ ] args) {  
        Auto a;  
        System.out.println(a);  
    }  
}
```

```
C:\Java_5>javac -d . AutoP.java  
AutoP.java:5: variable a might not have been initialized  
    System.out.println(a);  
                      ^  
1 error
```

Referencia változó



```
public class AutoP {  
    public static void main(String[ ] args) {  
        Auto a = new Auto("PXM218");  
        Auto b = new Auto("PXM218");  
        System.out.println(a==b);  
    }  
}
```

```
C:\Java_5>java AutoP  
false
```

Két referencia akkor sem egyforma, ha minden adattagjuk megegyezik!

Osztály fogalma

■ Végrehajtható kódot tartalmazó osztály készítése:

```
public class KörProgram {  
    public static void main(String[ ] p){  
        Kör a, b;  
        a = new Kör();  
        a.kp_x = 0;  
        a.kp_y = 0;  
        a.sugar = 10;  
        b = new Kör();  
        b.kp_x = 5;  
        b.kp_y = 8;  
        b.sugar = 3;  
        b.eltol(5, 5);  
    }  
}
```

● Végrehajtható metódus

● Deklarálás

● Inicializálás

● Hivatkozás adattagra

Hivatkozás: az objektum nevével minősítve, a . (pont) operátorral

● Hivatkozás metódusra

Hivatkozás: névvel, zárójelben paraméterekkel. A hivatkozás a metódus **elindítását** jelenti.

Tagok hozzáférési kategóriái

- Az **információ rejtés** alapelv megvalósításának eszköze.
- Feladatuk az objektumok (osztály, metódus, adattag) láthatóságának (elérhetőségének) szabályozása

Módosító	Hozzáférési kategória
private	Privát: más osztályból nem látható, de az adott osztály összes példánya számára elérhető
nincs megadva	Alap (félnyilvános): csak az azonos csomagban levő osztályok érhetik el (angolul <i>default</i> vagy <i>package-private</i>)
protected	Védett: az adott csomagban lévő és a leszármazott osztályok érik el
public	Nyilvános: bármely csomagban levő bármely osztályból elérhető

Tagok hozzáférési kategóriái

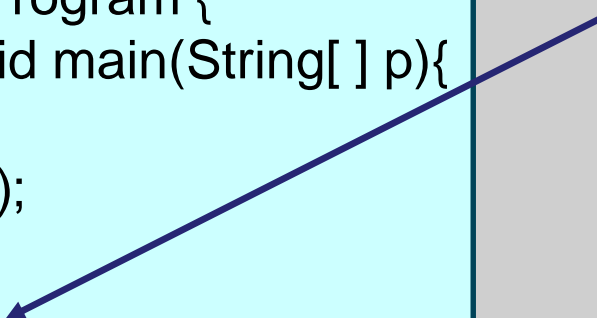
- A Kör osztály definíciója, az adatrejtés elvét betartva:

```
public class Kör {  
    private double kp_x;  
    private double kp_y;  
    private double sugár;  
    public void eltol(double dx, double dy) {  
        kp_x += dx;  
        kp_y += dy;  
    }  
    public void átméretez(double ujsugár) {  
        sugár = ujsugár;  
    }  
}
```

Tagok hozzáférési kategóriái

■ A KörProgram hibás lett! Miért?

```
public class KörProgram {  
    public static void main(String[ ] p){  
        Kör a, b;  
        a = new Kör();  
        a.kp_x = 0;  
        a.kp_y = 0;  
        a.sugar = 10;  
        b = new Kör();  
        b.kp_x = 5;  
        b.kp_y = 8;  
        b.sugar = 3;  
        b.eltol(5, 5);  
    }  
}
```



● Adattagok privátak, elérésük nem lehetséges!

Megoldás: az adattagokat elérő **setter** metódusok a Kör osztályban!

```
public class Kör {  
    ...  
    public void setKp_x(double p_kp_x) {  
        kp_x = p_kp_x;  
    }  
    public void setKp_y(double p_kp_y) {  
        kp_y = p_kp_y;  
    }  
    public void setSugar(double p_sug) {  
        sugar = p_sug;  
    }  
}
```

Tagok hozzáférési kategóriái

■ A javított KörProgram.

```
public class KörProgram {  
    public static void main(String[] p){  
        Kör a, b;  
        a = new Kör();  
        a.setKp_x(0);  
        a.setKp_y(0);  
        a.setSugar(10);  
        double r = a.getSugár();  
        double terület = r*r*Math.PI;  
        System.out.println(terület);  
    }  
}
```

A setter metódusokhoz hasonlóan **getter** metódusokat is használunk az adattagok értékének eléréséhez.

```
public class Kör {  
    ...  
    public double getKp_x() {  
        return kp_x;  
    }  
    public double getKp_y() {  
        return kp_y;  
    }  
    public double getSugár() {  
        return sugár;  
    }  
}
```

```
C:\Java_5>java KörProgram  
314.1592653589793
```

Objektumok inicializálása

- A Kör osztály adattagjait beállító metódus

```
public class Kör {  
    private double kp_x;  
    private double kp_y;  
    private double sugár;  
    public void setKör(double dx, double dy, double s) {  
        kp_x = dx;  
        kp_y = dy;  
        sugár = s;  
    }  
}
```

- Ha elfelejtjük meghívni a metódust, a program hibásan fog működni! – Hibalehetőség!
- Megoldás: **konstruktor** használata

Konstruktor

- **A konstruktor:** speciális metódus, amelyet nem lehet meghívni normál módon, hanem **automatikusan hívódik meg az osztály példányosításakor**. Szerepe az objektum induló állapotának beállítása (pl. adattagok inicializálása).
- A konstruktor szintaktikája:

```
módosítók azonosító (paraméterlista) {  
    utasítások  
}
```

- Metódushoz hasonló szintaktika, de
 - Az azonosító kötelezően az osztály neve kell legyen
 - Nincs visszatérési érték típus megadás
 - Lehetnek paramétereik, amelyeknek a példányosításkor adhatunk meg értékeket.
 - Csak a new operátorral hívható

Konstruktor

■ A Kör osztály konstruktora

```
public class Kör {  
    private double kp_x;  
    private double kp_y;  
    private double sugár;  
    public Kör(double dx, double dy, double s) {  
        kp_x = dx;  
        kp_y = dy;  
        sugár = s;  
    }  
}
```

A konstruktor:

- Túlterhelhető, több is lehet belőle (eltérő paraméterekkel)
- A konstruktor az osztály bármely metódusát használhatja

Metódusnév túlterhelés (overloading)

- Egy osztályhoz **több metódus** is tartozhat **azonos névvel, de különböző paraméterszignatúrával** (a formális paraméterek száma és típus-sorrendje). A visszatérési érték típusa ebből a szempontból közömbös.
- A metódus hívásakor a fordítóprogram az aktuális paraméterek szignatúrája alapján dönti el, hogy melyik metódust kell alkalmaznia.
- Ha egy megfelelőt sem talál, vagy ha többet is talál hibajelzést ad.
- Egy osztályhoz úgy tartozhat több azonos nevű metódus, hogy:
 - az osztályban definiálhatók azonos nevű metódusok
 - ugyanilyen nevű metódusokat örökölhet is az osztály
- A saját és örökölt metódusok együttesére érvényes a túlterhelés szabályrendszere

Osztályszintű tagok

- Az adattagokból minden példánynak saját készlete van. Ezeket ezért szokás *példányváltozónak* is nevezni.
- Ha az adattag deklarációja elé kitesszük a **static** minősítőt, **osztályszintű adattagot**, osztályváltozót kapunk
- A statikus adattagból csak egyetlen darab jön létre, minden példány elérí, közösen, osztottan használják
- Inicializálása az osztály első inicializálásakor történik
- Hivatkozhatunk rá az osztály bármely példányával
- Hivatkozhatunk rá az osztály nevével is

Osztályváltozó és osztálymetódus. Példakód

```
class Teszt {  
    int i;  
    static int k;  
  
    Teszt(int i){  
        this.i = i;  
    }  
    static void setK(int z){  
        k = z;  
    }  
}
```

Lefordítható a kód?

Igen

```
public class Program{  
    public static void main(String args[ ]) {  
        Teszt t = new Teszt(5);  
        Teszt.setK(7);  
        System.out.println(Teszt.k);  
    }  
}
```

Lefordítható a kód?

Igen

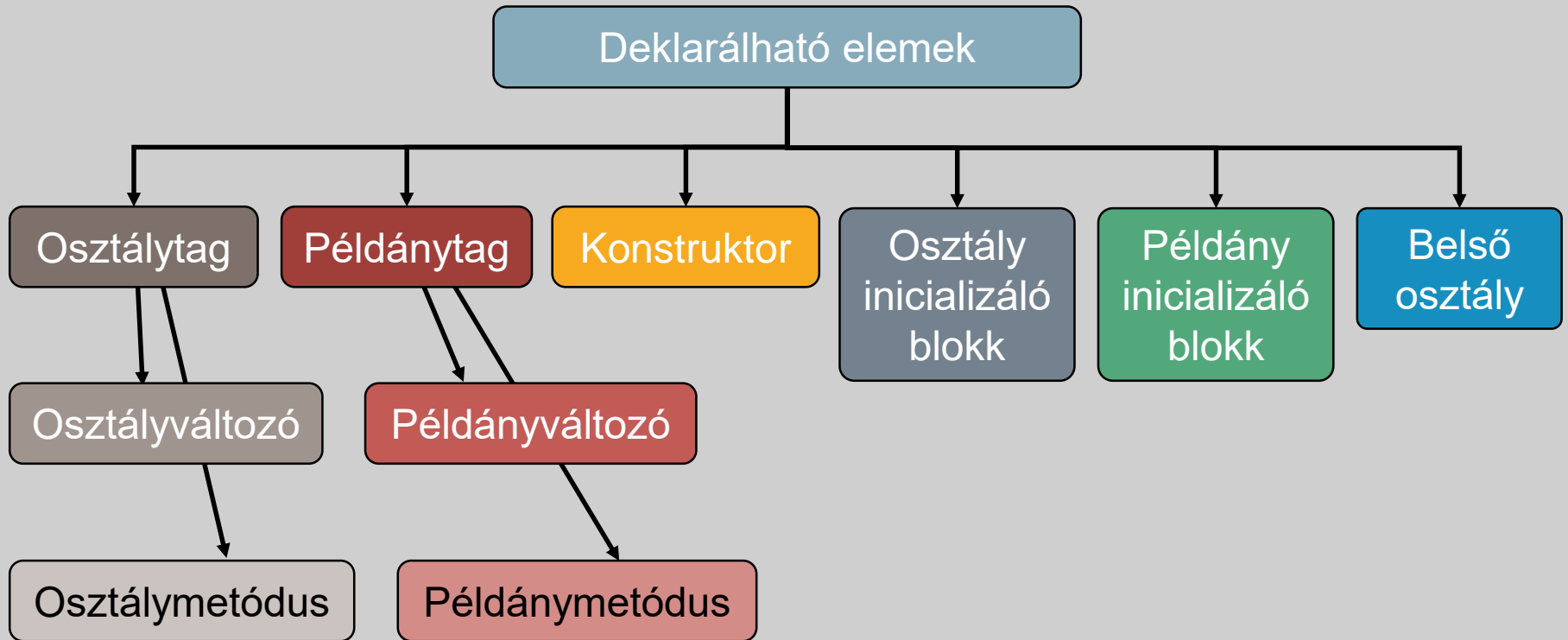
Mit ír ki a program?

7

Osztályszintű tagok – Osztálymetódusok

- Létrehozásuk oka: akkor is használhatók, ha az osztálynak nem léteznek példányai.
- Létrehozásuk: a metódust **static** minősítővel látjuk el
- Hivatkozás:
 - osztályon belül névvel,
 - osztályon kívül minősített hivatkozással: az osztály vagy bármely példányának neve, pont, metódusnév
 - Előző oldali példában: `Teszt.setK(7);`
- Osztálymetódusban a **this** és a **super** referenciák nem használhatók!
- A programosztályokban a main metódus mindig statikus, vagyis osztálymetódus!

Deklarációk egy osztályban



Lefordítható az alábbi kód? **Nem**

```
class A {  
    protected void function() {}  
}  
class B extends A {  
    void function() {}  
}  
public class Test {  
    public static void main(String args[]) {  
        B b = new B();  
        b.function();  
    }  
}
```

```
Error: function() in B cannot override function() in A  
    void function() {}  
        ^
```

Attempting to assign weaker access privileges; was protected

Lefordítható az alábbi kód? **Nem**

```
class A {  
    public void function() {}  
}  
class B extends A {  
    void function() {}  
}  
public class Test {  
    public static void main(String args[]) {  
        B b = new B();  
        b.function();  
    }  
}
```

```
Error: function() in B cannot override function() in A  
    void function() {}  
        ^
```

Attempting to assign weaker access privileges; was public

Lefordítható az alábbi kód?

Nem

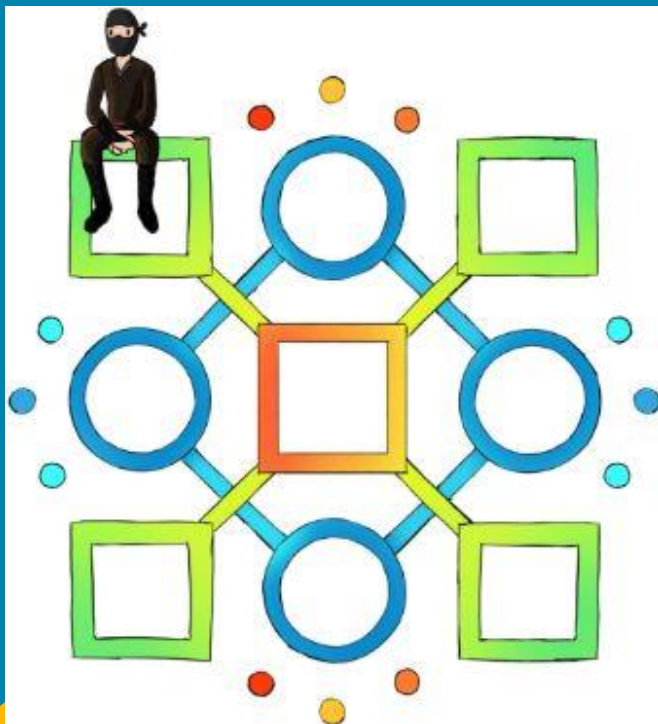
Ha igen, mit ír ki a kód?

```
class Test {  
    public static void main(String args[]) {  
        System.out.println(fun());  
    }  
    int fun() {  
        return 20;  
    }  
}
```

Error: non-static method fun() cannot be referenced
from a static context

```
        System.out.println(fun());  
                           ^
```

3



4

Tömb

- A tömb olyan adatszerkezet, melyet egyforma típusú elemek alkotnak
- A tömb elemeire sorszámukkal (indexükkel) lehet hivatkozni
- A Java statikus tömböket használ, ez azt jelenti, hogy az elemek száma fix, nem változtatható
- Javában a tömb egy speciális szintaktikával definiálható osztály:
 - referenciával hivatkozhatunk rá
 - példányosítani kell (speciális szintaktikával)

```
int[] t = new int[5];  
t[0] = 3;
```

```
Alma[] a = new Alma[4];  
a[0] = new Alma("Jonatán");
```

A tömb létrehozásával a hivatkozott objektumok nem jönnek létre, azokat külön létre kell hozni!

Enum típus

- Az **enum** egy speciális Java osztály, amelyet konstansok gyűjteményeinek tárolására, kezelésére használhatunk.
 - A Java 5.0 változatában jelent meg
 - A `java.lang.Enum` osztályból származnak le
 - Egyszerű deklarációval jönnek létre, nem lehet őket példányosítani
- ```
enum Egtaj {
 ESZAK, DEL, KELET, NYUGAT
}
```
- Az enum típusoknak nem adunk értéket, csak a nevükkel hivatkozunk rájuk
  - Az enum konstansok valójában statikus referenciák az osztály egy-egy példányára.

# ArrayList – Rugalmas tömb

---

- A tömbök korlátai
  - Fix méret, nem lehet menet közben megnövelni
    - Ha túl nagyra választjuk, fölösleges helyfoglalás
  - Rendezett tömb módosítása
    - Elem beszúrása
    - Elem törlése
  - Keresés a tömbben
    - Végig kell nézni az összes elemet
- Megoldás: ArrayList, a változtatható méretű tömb

# ArrayList

---

- A java.util.ArrayList osztályban található

- Példányosítani kell

```
ArrayList<típus> = new ArrayList<típus>();
```

- Csak burkoló osztály lehet az elemtípusa
  - Integer, Double, Character, Long, Boolean...
  - A lista csak és kizárólag referencia típusú adatokat képes tárolni, így primitív típusokat nem rakhatunk bele.
- Fontos metódusok:
  - add(), size(), get(index), indexOf(érték), toString()
  - remove(érték), set(index, érték), toArray()

## Enum, ArrayList példa

```
import java.util.ArrayList;
import java.util.List;

public class Card {
 public enum Ertek { KETTES, HÁRMAS, NÉGYES, ÖTÖS, HATOS,
 HETES, NYOLCAS, KILENCES, TÍZES, BUBI, DÁMA, KIRÁLY, ÁSZ }

 public enum Szin { PIKK, KÁRÓ, TREFF, KÖR }

 private final Ertek ertek;
 private final Szin szin;
 private Card(Ertek ertek, Szin szin) {
 this.ertek = ertek;
 this.szin = szin;
 }
 public String toString() { return "" +szin+ " " +ertek; }

 private static final List<Card> kartyaLapok = new ArrayList<Card>();

 static {
 for (Szin szin : Szin.values())
 for (Ertek ertek : Ertek.values())
 kartyaLapok.add(new Card(ertek, szin));
 }

 public static ArrayList<Card> ujPakli() {
 return new ArrayList<Card> (kartyaLapok);
 }
}
```

# Enum példa

```
import java.util.ArrayList;
import java.util.List;

public class CardList {

 public static void main(String[] args) {
 List<Card> card = new ArrayList<Card>();
 card = Card.ujPakli();
 for (Card c : card) System.out.println(c);
 }
}
```

PIKK KETTES  
PIKK HÁRMAS  
PIKK NÉGYES  
PIKK ÖTÖS  
PIKK HATOS  
PIKK HETES  
PIKK NYOLCAS  
PIKK KILENCES  
PIKK TÍZES  
PIKK BUBI  
PIKK DÁMA  
PIKK KIRÁLY  
PIKK ÁSZ

KÁRÓ KETTES  
KÁRÓ HÁRMAS  
KÁRÓ NÉGYES  
KÁRÓ ÖTÖS  
KÁRÓ HATOS  
KÁRÓ HETES  
KÁRÓ NYOLCAS  
KÁRÓ KILENCES  
KÁRÓ TÍZES  
KÁRÓ BUBI  
KÁRÓ DÁMA  
KÁRÓ KIRÁLY  
KÁRÓ ÁSZ

TREFF KETTES  
TREFF HÁRMAS  
TREFF NÉGYES  
TREFF ÖTÖS  
TREFF HATOS  
TREFF HETES  
TREFF NYOLCAS  
TREFF KILENCES  
TREFF TÍZES  
TREFF BUBI  
TREFF DÁMA  
TREFF KIRÁLY  
TREFF ÁSZ

KŐR KETTES  
KŐR HÁRMAS  
KŐR NÉGYES  
KŐR ÖTÖS  
KŐR HATOS  
KŐR HETES  
KŐR NYOLCAS  
KŐR KILENCES  
KŐR TÍZES  
KŐR BUBI  
KŐR DÁMA  
KŐR KIRÁLY  
KŐR ÁSZ

Lefordítható az alábbi kód? **Igen**

Ha igen, mit ír ki a kód? **5510**

```
public class Test {
 static void methodOne(int[] a) {
 int[] b = new int[5];
 a = b;
 System.out.print(a.length);
 System.out.print(b.length);
 }

 public static void main(String[] args) {
 int[] a = new int[10];
 methodOne(a);
 System.out.print(a.length);
 }
}
```

Mit ír ki a kód?

1  
1  
1

```
enum Enums {
 A, B, C;
 private Enums() {
 System.out.println(1);
 }
}

public class Test {
 public static void main(String[] args){
 Enum en = Enums.B;
 }
}
```

Lehet egy enum-nak public konstruktora?

Nem



Lefordítható az alábbi kód?

Igen

Ha igen, mit ír ki a kód?

```
public class Test {
 public static void main(String[] args) {
 ArrayList<Integer> list =
 new ArrayList<Integer>(List.of(3,5,1));
 System.out.println("Numbers: " + list);
 list.add(6);
 list.add(0, 4);
 list.remove(1);
 System.out.println("Numbers: " + list);
 }
}
```

Numbers: [3, 5, 1]

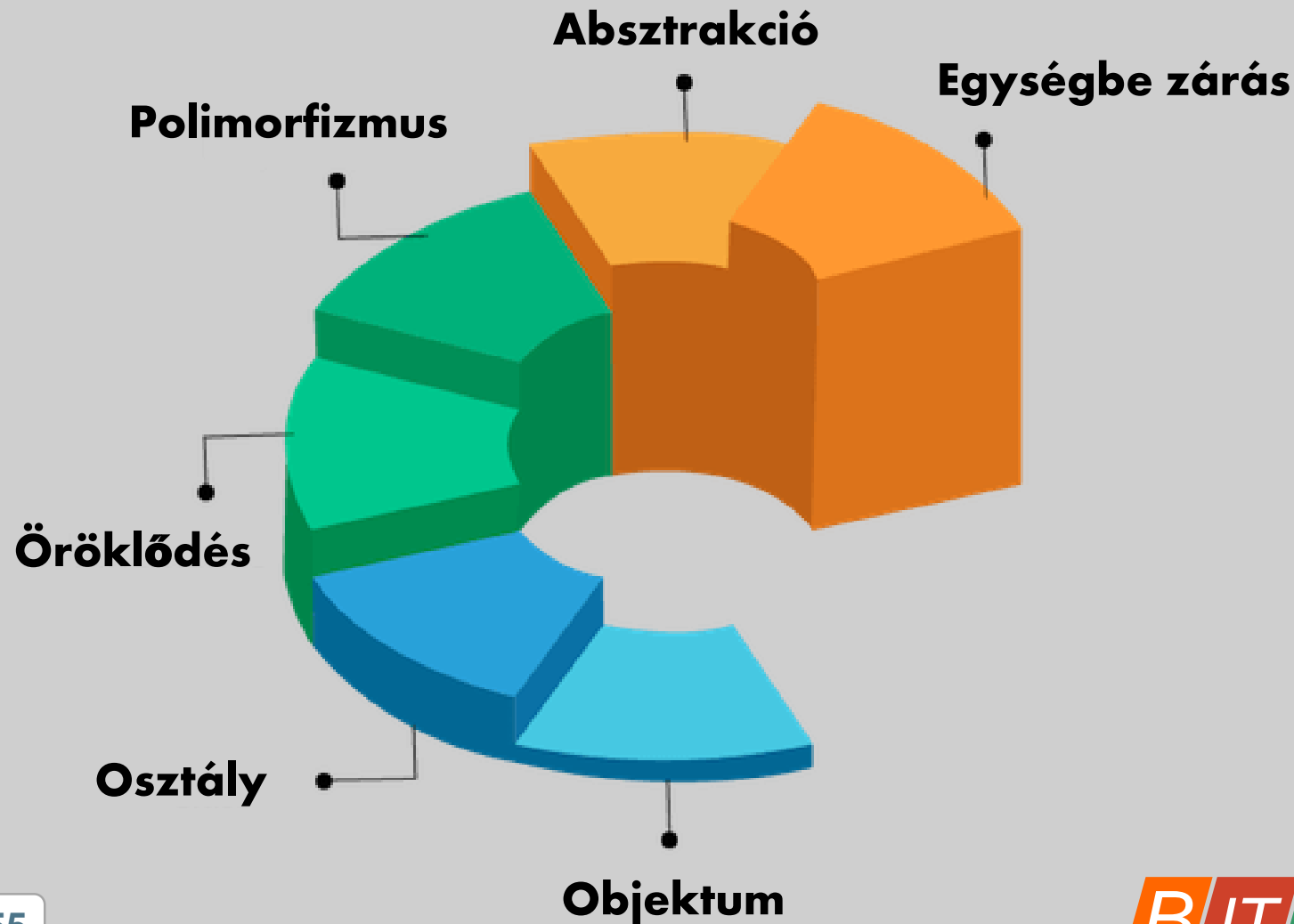
Numbers: [4, 5, 1, 6]

4

5

# Objektum Orientált Programozási Rendszer

---



# További OOP alapfogalmak

---

- **Objektum:** Egy osztály egy működőképes példánya. Egy adott osztályban definiált tulajdonságokkal tetszőleges számú objektum példányosítható. Minden objektum természeténél fogva különbözik az összes többitől. Egy adott osztályból példányosított valamennyi objektumnak ugyanolyan lehetséges viselkedési módjai vannak, de mindnek saját állapota van.
- **Osztály:** Objektumok közös tulajdonságait definiálja. Programozástechnikai szempontból egy típus. Építőkövei:
  - adattagok (attribútumok, jellemzők, tulajdonságok),
  - metódusok (műveletek, viselkedés).
- **Üzenet:** Az objektummal való kommunikáció módja. A módszerek (metódusok) aktivizálását jelenti.

# OOP alapelvek

- **Egységbe zárás** (**encapsulation**): Az adatok és a hozzájuk kapcsolódó tevékenységek együtt egy egységet alkotnak, ennek alapegysége az **osztály**, melynek építőkövei:
  - adattagok,
  - metódusok.
- **Öröklődés** (**inheritance**): az osztályok egy részét már meglévő osztályok továbbfejlesztésével hozzuk létre, úgy, hogy további adattagokkal, illetve metódusokkal bővítjük.
- **Többalakúság** (**polimorphism**): A származtatott osztályokban lehetnek ugyanolyan elnevezésű, de más tevékenységű metódusok, mint az ős osztályban => áttekinthető kód, nem kell az elnevezéseket variálnunk.



PIE

Polimorphism  
Inheritance  
Encapsulation

# OOP módszerek

---

- **Absztrakció** (**abstraction**): Elvonatkoztatás. Segítségével privát implementációkat rejthetünk el egy nyilvánosinterfész mögé. Pl. absztrakt osztályok, vagy ugyanazt az interfészt megvalósító osztályok, melyeket ugyanolyan módszerekkel lehet kezelni.
- **Információ rejtés**: Egy objektum adatai a külvilág számára hozzáférhetetlenek. Egy objektum a külvilággal csak az interfészén keresztül tarthatja a kapcsolatot. (Interface: a külvilág számára elérhető módszerek –metódusok– együttese.) A módszerek implementációja rejtett.

# Öröklődés

---

## ■ Megadása:

```
class osztálynév extends ősosztály {
 ...
}
```

- Ha nem adunk meg szülő osztályt, akkor automatikusan a `java.lang.Object` osztály a szülő.
- Az `Object` tehát minden osztály őse, tagjait minden osztály örökli!
- A gyermek osztály örökli szülője tagjait, de:
  - a `private` tagokhoz nem fér hozzá,
  - a félnyilvános tagokhoz pedig csak akkor, ha ugyanabban a csomagban van (ezeket a szülő osztály hozzáférhető tagjainak segítségével kezelheti)
  - a `protected`, `public` tagokhoz közvetlenül hozzáfér.

# Konstruktorok az öröklődés során

---

- **A konstruktor nem öröklődik.** Mind az ős osztály, mind a leszármazott osztály rendelkezhet konstruktorral (akár többel is). Egy leszármazott objektum példányosításánál tisztázni kell:
  - A konstruktorok végrehajtási sorrendjét
  - Azt, hogy hogyan választhatjuk ki az ősosztály konstruktorai közül a végrehajtandót
- **Végrehajtási sorrend:** először mindig az ősosztály, majd a leszármazott osztály konstruktora hajtódik végre. A pontos sorrend:
  1. Az ős osztály adattagjainak inicializálása
  2. Az ős osztály konstruktorának végrehajtódása
  3. A gyermek osztály adattagjainak inicializálása
  4. A gyermek osztály konstruktorának végrehajtódása



```
public class Szemely {
 String nev;
 int kor;
```

```
Dolgozo d = new Dolgozo("tanár", 550);
```

```
 public Szemely(String nev, int kor){
 this.nev = nev;
 this.kor = kor;
```

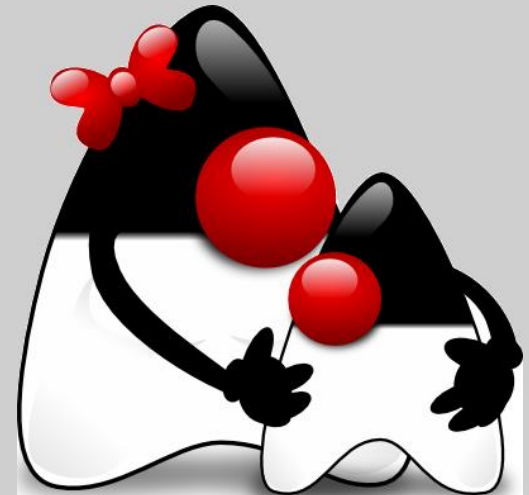
```
 }
 public Szemely(){
 this("Kis Béla", 25);
```

```
 }
}
```

```
public class Dolgozo extends Szemely {
 String munkakor;
 int fizetes;
```

```
 public Dolgozo(String munkakor,int fizetes){
 super();
 this.munkakor = munkakor;
 this.fizetes = fizetes;
```

```
 }
}
```



```
c:\Java8>java Prg
Kis Béla - 25 - tanár - 550
```

# Öröklődés

---

## ■ Adattagok öröklése:

- A leszármazott örökli az összes adattagot, és új adattagokat is definiálhat
- Definiálhat egy örökölt adattaggal megegyező nevű adattagot is, ilyenkor az örökölt tagot elrejtí (hide)
- Az elrejtett tagra a **super** kulcsszóval hivatkozhatunk:
  - `super.adattag`
  - `super.super.adattag` hívás nem lehetséges!

# Adattagok öröklődéskor

```
public class Szemely {
 String nev;
 int fizetes;

 public Szemely(String nev, int fizetes){
 this.nev = nev;
 this.fizetes = fizetes;
 }
}
```

```
public class Dolgozo extends Szemely {
 int fizetes;

 public Dolgozo(String nev, int fiz1, int fiz2){
 super(nev, fiz1);
 fizetes = fiz2;
 }
}
```

```
c:\Java8>java Prg
Béla - 200
Béla - 550
```

```
public class OuterP {
 public static void main(String args[]) {
 Dolgozo d = new Dolgozo("Béla", 550, 200);
 System.out.println(d.nev+" - "+d.fizetes);
 Szemely sz = d;
 System.out.println(sz.nev+" - "+sz.fizetes);
 }
}
```

# Öröklődés

---

## ■ Metódusok öröklése

- A leszármazott örökli az összes nem `private` metódust és újakat is definiálhat, valamint megváltoztathat örökölt metódust
- Esetek:
  1. *Új metódus*: Az örökölt metódusoktól eltérő nevű metódus definiálása
  2. *Új metódus örökölt metódus túlterhelésével* (*overloading*): az egyik örökölt metódussal megegyezik a név, de eltér a paraméter szignatúra.
  3. *Példányszintű metódus felüldefiniálása* (*override*): az egyik örökölt példányszintű metódussal megegyezik a szignatúra és a visszatérési érték típus

**Paraméter szignatúra:** A formális paraméterek száma és típus-sorrendje. A visszatérési érték típusa ebből a szempontból közömbös.

# Metódusnév túlterhelés

```
public class Szemely {
 String név;
 int fizetés;
 int getFizetés(){
 return fizetés;
 }
 void fizetéstEmel(int fizetés){
 this.fizetés += fizetés;
 }
}
```

```
public class Alkalmazott extends Szemely {
 String beosztás;
 void fizetéstEmel(){
 fizetés += 20000;
 }
 void fizetéstEmel(Alkalmazott a){
 if (a.getFizetés() > fizetés)
 fizetés = a.getFizetés();
 }
}
```

Az **Alkalmazott** osztályban a `fizetéstEmel` metódus három különböző paraméterszignatúrával is használható.

# Metódusnév felüldefiniálás

```
public class Alkalmazott {
 String név;
 int nyelvekSzama;
 public int potlek() {
 return nyelvekSzama*20000;
 }
}
```

```
public class Fonok extends Alkalmazott {
 int beosztottakSzama;
 public int potlek() {
 return super.potlek() + beosztottakSzama*10000;
 }
}
```

Az őssztály **potlek()** metódusának meghívása

# Statikus és dinamikus típus

---

- Egy változónak van **statikus** és **dinamikus** típusa.
- Egy változó **statikus** típusa az, amelyet a deklarációjában megadtunk
  - Ez a változó teljes élete alatt **változatlan**
  - Ez határozza meg, hogy **milyen műveleteket végezhetünk** a referenciával hivatkozott objektummal
- Egy változó **dinamikus** típusa az általa éppen hivatkozott objektum tényleges típusa.
  - Csak olyan típus lehet, amely rendelkezik ugyanazokkal az adatokkal és műveletekkel, mint a statikus típus, ezért a változó dinamikus típusa csak **a statikus típus** vagy **annak leszármazottja lehet**.
  - A dinamikus típus a program futása során **bármikor változhat**

# Statikus és dinamikus típus

---

```
Gyümölcs x = new Alma("Jonatán");
```

## Statikus típus

- Sosem változik
- Vannak bizonyos tulajdonságai:
  - adattagok
  - metódusok

## Dinamikus típus

- Változhat
- Rendelkeznie kell a statikus típus minden tulajdonságával, de lehetnek további tulajdonságai is, emiatt lehet a statikus típus, vagy annak valamelyik leszármazottja.

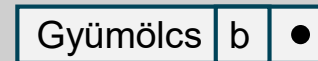
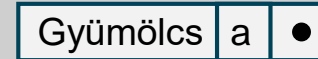


# Statikus és dinamikus típus



Egyszerű példa: miből mi lehet?

Gyümölcs a, b;



Deklarálás  
Statikus  
értékadás

a = new Alma("Jonatán");



Példányosítás  
Dinamikus  
értékadás

b = new Gyümölcs("Kiwi");



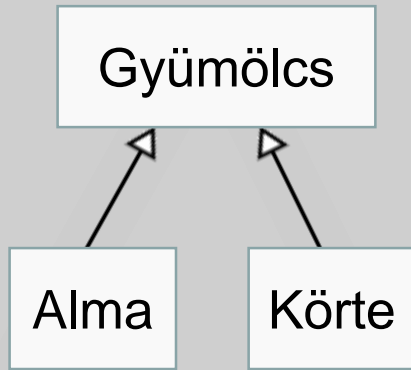
Értékadás  
Dinamikus  
értékváltás

a = b;



Újra  
példányosítás  
Dinamikus  
értékadás

a = new Körte("Vilmos");



- Egy gyümölcs példány lehet Gyümölcs, Alma, vagy Körte típusú.
- Alma példány csak Alma típusú lehet.
- Körte példány csak Körte típusú lehet.

# Statikus és dinamikus típus

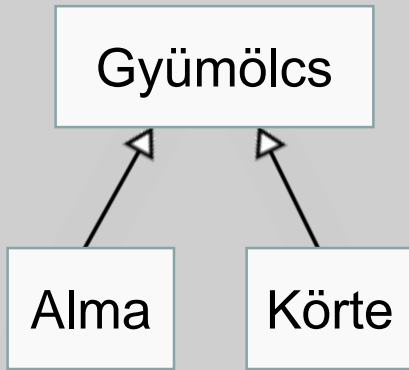


Mi történik?

**Gyümölcs** a = new Körte("Vilmos");



**Körte** c = new Körte("Piros");



- Egy gyümölcs példány lehet Gyümölcs, Alma, vagy Körte típusú.
- Alma példány csak Alma típusú lehet.
- Körte példány csak Körte típusú lehet.

**a = c;**

A **Gyümölcs** típusú **statikus változóba** berakunk egy **Körte** típusú **statikus változót**! Működik? Igen, mert a Gyümölcs lehet Körte

**c = a;**

A **Körte** típusú **statikus változóba** beraknánk egy **Gyümölcs** típusú **statikus változót**! Működik? Nem, mert a Körte csak Körte lehet!

Értékadáskor változik a referencia (a dinamikus típus), de csak olyan érték adható át, amelynek a statikus típusa megfelelő!

# Statikus és dinamikus típus



Mi történik?

**Gyümölcs** *a* = new Körte("Vilmos");

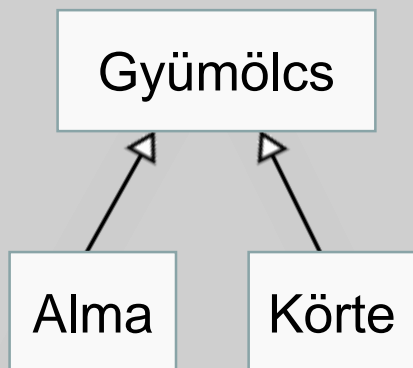


**Körte** *c* = new Körte("Piros");



**c = (Körte)a;**

Ha értékadáskor a jobb oldalon álló **dinamikus típus megegyezik** a bal oldalon álló statikus típussal, akkor konvertálással megoldható az értékadás.



- Egy gyümölcs példány lehet Gyümölcs, Alma, vagy Körte típusú.
- Alma példány csak Alma típusú lehet.
- Körte példány csak Körte típusú lehet.

Értékadáskor változik a referencia (a dinamikus típus), de csak olyan érték adható át, amelynek a statikus típusa megfelelő!

# Interfész

---

- Egy osztály interfészén a nyilvános elemeinek összességét értjük, azokat, amelyek az osztály megfelelő használatához szükségesek
- Az információ rejtés elve miatt az interfész általában csak metódusokból áll.
- Szintaktikája hasonló az osztályhoz, de a `class` kulcsszó helyett `interface` kulcsszót kell használni.
- Az interfészt tartalmazó fájl nevének meg kell egyeznie a kódban szereplő interfész névvel.

```
interface Nyomtathato{
 void nyomtat();
}
```

```
class Konyv implements Nyomtathato{
 String tartalom = "ABC";
 public void nyomtat(){
 System.out.println(tartalom);
 }
}
```

# Kivételkezelés

---

- Fogalma: A hibakezelés egy formája, amelyben elválasztható a hibalekezelő kódrészlet a normál működés kódjától, valamint a jelezhetjük a kódunkban előforduló hibákat és előírhatjuk kezelésüket a kódunkat használó kód számára.
- Kivételek keletkezése
  - Kivétel létrejöhet egy végrehajtás során előforduló hiba hatására (I/O hiba, tömbindex túllépés, stb.), valamint saját magunk is létrehozhatunk kivételeket.
  - A kivétel létrejöttét a kivétel megdobódásának nevezzük.
  - A kivétel elkapását a kivétel lekezelésének nevezzük.

# Kivételkezelés

- A védett kódot **try** blokkban helyezzük el:  
**try** { utasítások }
- Az első kivételt kiváltó utasítással befejeződik a **try** blokk végrehajtása, létrejön egy **kivétel** objektum. Fontos: a try blokk kivételt kiváltó utasítása utáni utasítások tehát mindig kimaradnak!

```
try {
```

**X** kód, ami kiválthat egy kivételt  
további utasítások

```
} catch (Kivétel típusa Azonosító) {
```

Kód belépési pont

Kód végrehajtás  
kivétel esetén

**try {**

Kód végrehajtás  
hibamentes  
esetben



kód, ami kiválthat egy kivételt

**} catch (** Kivétel típusa Azonosító **) {**



kód, ami a kivétel kiváltódásakor lefut

**} finally {**

**}**



kód, ami mindig lefut

Kód kilépési pont



Ő Tinku.

Mexikói cserediák.

A faterja kinyírja, ha  
nem tanul meg  
tökéletesen  
programozni!

Most tőlünk kér  
segítséget!

Keressük meg a kódjaiban  
a hibát, és javítsuk ki!



# Kódtalány

1.



Tinku írta a kódot. Mi a hiba?

```
Error: '{' expected
class Test extends X, Y {
 ^
```

```
class X {
 int a;
 X(int a){
 this.a=a;
 }
}

class Y {
 int b;
 Y(int b){
 this.b=b;
 }
}

class Test extends X, Y {
 int c;
 Test(int a, int b, int c){
 super(a);
 super(b);
 this.c=c;
 }
}
```

Tinku írta a kódot, de nem érti, miért azt írja ki, amit.

1. Mit ír ki a kód?

10  
B

2. Miért azt?

```
class A {
 int i = 10;
 void Print(){System.out.println("A");}
}

class B extends A {
 int i = 20;
 void Print(){System.out.println("B");}
}

public class Test {
 public static void main(String[] args) {
 A a = new B();
 System.out.println(a.i);
 a.Print();
 }
}
```



Tinku már itt elakadt. Mi a hiba? Hogy lehet kijavítani?

```
public class A {
 public A(int i) {
 System.out.println(i);
 }
}

class B extends A {
 void B(int i) {
 System.out.println(i);
 }
}
```

```
Error: constructor A in class A cannot be applied to given types;
class B extends A {
^
 required: int
 found: no arguments
 reason: actual and formal argument lists differ in length
```

Így már lefordítható a kód?

Nem

```
public class A {
 public A(int i) {
 System.out.println(i);
 }
}
class B extends A {
 void B(int i) {
 super(i);
 System.out.println(i);
 }
}
```

```
Error: constructor A in class A cannot be applied to given types;
class B extends A {
^
 required: int
 found: no arguments
 reason: actual and formal argument lists differ in length
```

Így már lefordítható a kód?

Igen

```
public class A {
 public A(int i) {
 System.out.println(i);
 }
}
class B extends A {
 public Test(int i) {
 super(i);
 }
 void B(int i) {
 System.out.println(i);
 }
}
```



Tinku megint elakadt. Mi a hiba?  
Hogy lehet kijavítani?

```
class X {
 private int m = 48;
}

class Test extends X {
 void method() {
 System.out.println(m);
 }
}
```

```
Error: m has private access in X
 System.out.println(m);
 ^
```

```
class A {
 public A(int i) {
 System.out.println(1);
 }
 public A() {
 this(10);
 System.out.println(2);
 }
 void A() {
 A(10);
 System.out.println(3);
 }
 void A(int i) {
 System.out.println(4);
 }
}

public class Test {
 public static void main(String[] args) {
 new A().A();
 }
}
```

## Kódtalány 16.

Tinku írta a kódot,  
és működik :-)

Mit ír ki?

1  
2  
4  
3



# 5

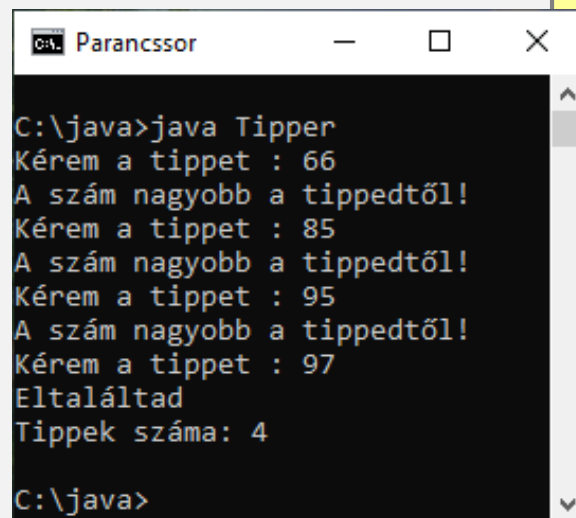
**Ennyi. Jöhet egy  
kis gyakorlat**



Készítsünk egy komplett konzolos alkalmazást, mely generál egy számot 0 és 100 között (mindkét határt beleértve), feladatunk a szám kitalálása.

Beírunk egy tippet, és kapunk egy üzenetet:

- A szám kisebb a tippedtől!
- A szám nagyobb a tippedtől!
- Eltaláltad!



```
C:\java>java Tipper
Kérem a tippet : 66
A szám nagyobb a tippedtől!
Kérem a tippet : 85
A szám nagyobb a tippedtől!
Kérem a tippet : 95
A szám nagyobb a tippedtől!
Kérem a tippet : 97
Eltaláltad
Tippek száma: 4
C:\java>
```

A programot írjuk meg, ahogy tudjuk, aztán alakítsuk át az oop elveknek megfelelően, végül alakítsuk át az MVC szemléletnek megfelelően.

# Magyarázatok

## Programlogika:

1. Generálunk egy véletlen számot (rndNum)
2. Bekérünk a konzolról egy számot
3. Megvizsgáljuk a bekért számot:
  - Írtak be valamilyen adatot?
  - Az adat számmá konvertálható?
4. Ha az adat számmá konvertálható, átkonvertáljuk (tipp)
5. Ha nincs beírt adat, vagy nem konvertálható számmá, hibaüzenetet írunk, és visszatérünk a 2. lépéshez
6. Vizsgáljuk a tipp és az rndNum viszonyát, kiírjuk a kiértékelést
7. Ha nem találtuk ki, visszatérünk a 2. lépéshez
8. Ha kitaláltuk, leáll a program

Véletlen szám generálás:

**1. lehetőség:** `java.lang.Math` osztály `double random()` metódussal, mely 0-1 között generál véletlen számot:

```
int rnd = (int)(101*Math.random());
```

**2. lehetőség:** `java.util.Random` osztály `int nextInt(n)` metódusával, mely a  $0 \leq x < n$  tartományban generál véletlen számot:

```
Random rnd = new Random();
int rndNum = rnd.nextInt(101);
```

## Beolvasás konzolról:

Használjuk a `java.util.Scanner` osztály `nextLine()` utasítását:

```
Scanner sc = new Scanner(System.in);
String tippS = sc.nextLine();
```

## A bekért adat vizsgálata:

Hossz ellenőrzése:

```
if (tippS.length()==0)
```

Adat konvertálása számmá:

```
try {
 int x = Integer.valueOf(tippS);
} catch (NumberFormatException nfe) { ... }
```

```
int x = Integer.parseInt(tippS);
```

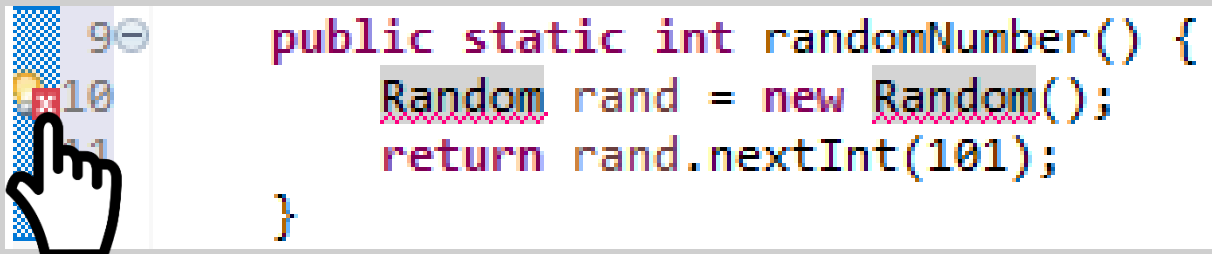
# Feladat – OOP technológia

Próbáljuk úgy kialakítani a kódot, hogy az "üzleti logikát" metódusokba szervezzük. Létrehozandó metódusok:

1. Véletlen egész számot előállító metódus, legyen a neve `randomNumber`;
2. Adatot beolvasó metódus (neve: `readTip`), mely kiírja, hogy "Kérem a tippet :", beolvassa azt, és visszaadja a stringet.
3. Adat ellenőrző (`check` névvel), mely ellenőrzi, hogy üres-e az adat, és hogy számmá alakítható-e. Logikai értéket adjon vissza.
4. Stringet egészzé alakító metódus, legyen a neve `stringToInt`.
5. Kiértékelő metódus (`evaluate` névvel), mely megkapja a leellenőrzött beolvasott stringet, számmá alakítja, és kiértékeli a tippet (az eredményt kiírja a képernyőre), és számolja a tippeket. Igaz értéket adjon vissza, ha eltaláltuk a számot, hamisat, ha nem.
6. A metódusokat felhasználva írjuk meg a főprogramot!

# Megoldás Eclipsben

1. Készítsünk egy új Java projektet, legyen a neve: [ozd\\_1](#)
2. Készítsünk a projekthez egy [Tipper](#) nevű programosztályt
3. Írjuk meg a [randomNumber](#) metódust:



```
9 public static int randomNumber() {
10 Random rand = new Random();
11 return rand.nextInt(101);
 }
```

A hand cursor icon points to the `Random` variable on line 10. The IDE shows two red squiggly lines under the word `Random`, indicating unresolved references.

Multiple markers at this line

- Random cannot be resolved to a type
- Random cannot be resolved to a type

A hiba  
elhárítása

Jobb klikk a nem egyértelműen felismerhető elemre, aztán  
Source menü, végül [Organize imports](#)

# Megoldás Eclipsben

---

4. Írjuk meg a `readTip` metódust:

```
public static String readTip() {
 System.out.print("Kérem a tippet : ");
 Scanner sc = new Scanner(System.in);
 String s = sc.nextLine();
 return s;
}
```

# Megoldás Eclipsben

5. Írjuk meg a **check** metódust:

```
public static boolean check(String s) {
 boolean out = true;
 if (s.length()==0) {
 System.out.println("Nem írtál be adatot!");
 out = false;
 }
 if (out) {
 try {
 int x=Integer.valueOf(s);
 } catch (NumberFormatException nfe) {
 System.out.println("Hibás a beírt adat formátuma!");
 out = false;
 }
 }
 return out;
}
```



# Megoldás Eclipsben

6. Írjuk meg a `stringToInt` metódust:

```
public static int stringToInt (String s) {
 int x = -1;
 try {
 x=Integer.valueOf(s);
 } catch (NumberFormatException nfe) {
 System.out.println("stringToInt: "+nfe.getMessage());
 }
 return x;
}
```

7. Útálom leírni, hogy `System.out.println`, ráadásul sokszor szerepel ezután is, ezért írok egy a `SM` nevű metódust:

```
public static void SM(String msg) {
 System.out.println(msg);
}
```

8. Az összes helyen a kódban írjuk át a `System.out.println`-t `SM`-re!

# Megoldás Eclipsben

1. Írjuk meg az `evaluate` metódust:

```
public static boolean evaluate(int rndNum, String s) {
 boolean ok = false;
 String mS = "";
 int tipp = stringToInt(s);
 if (rndNum == tipp) {
 mS="Eltaláltad!";
 ok = true;
 }
 if (rndNum < tipp) {
 mS="A szám kisebb a tippedtől!";
 }
 if (rndNum > tipp) {
 mS="A szám nagyobb a tippedtől!";
 }
 SM(mS);
 return ok;
}
```

# Megoldás Eclipsben

Adok időt!



## 1. Próbáljuk meg megírni a főprogramot!

```
public class Tipper {
 static boolean success = false;

 public static void main(String[] args) {
 int rndNum = randomNumber();

 while (!success) {
 String tippS = readTip();
 if (check(tippS)) {
 success = evaluate(rndNum, tippS);
 }
 }
 }
}
```

```
Kérem a tippet : 55
A szám nagyobb a tippedtől!
Kérem a tippet : 73
A szám nagyobb a tippedtől!
Kérem a tippet : 92
A szám kisebb a tippedtől!
Kérem a tippet : 81
A szám nagyobb a tippedtől!
Kérem a tippet : 86
Eltaláltad!
```

# Továbbfejlesztés

---

1. Tároljuk a tippeket valamilyen adatszerkezetben, és ha eltaláltuk a számot, írjuk ki a tippeket egy szöveges `output.txt` nevű fájlba.
  - Milyen típusú adatokat tároljunk?
  - Milyen adatszerkezetben tároljuk a tippeket?

# Továbbfejlesztés

---

1. A `fileWriter` metódus kódja:

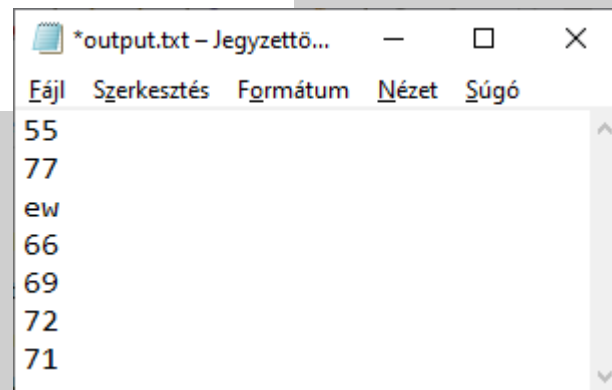
```
public static void fileWriter(ArrayList<String> al) {
 try {
 FileWriter writer = new FileWriter("output.txt");
 for(String str: al) {
 writer.write(str + System.LineSeparator());
 }
 writer.close();
 } catch (Exception e) {
 SM("fileWriter: "+e.getMessage());
 }
}
```

# Továbbfejlesztés

## 1. A módosított főprogram:

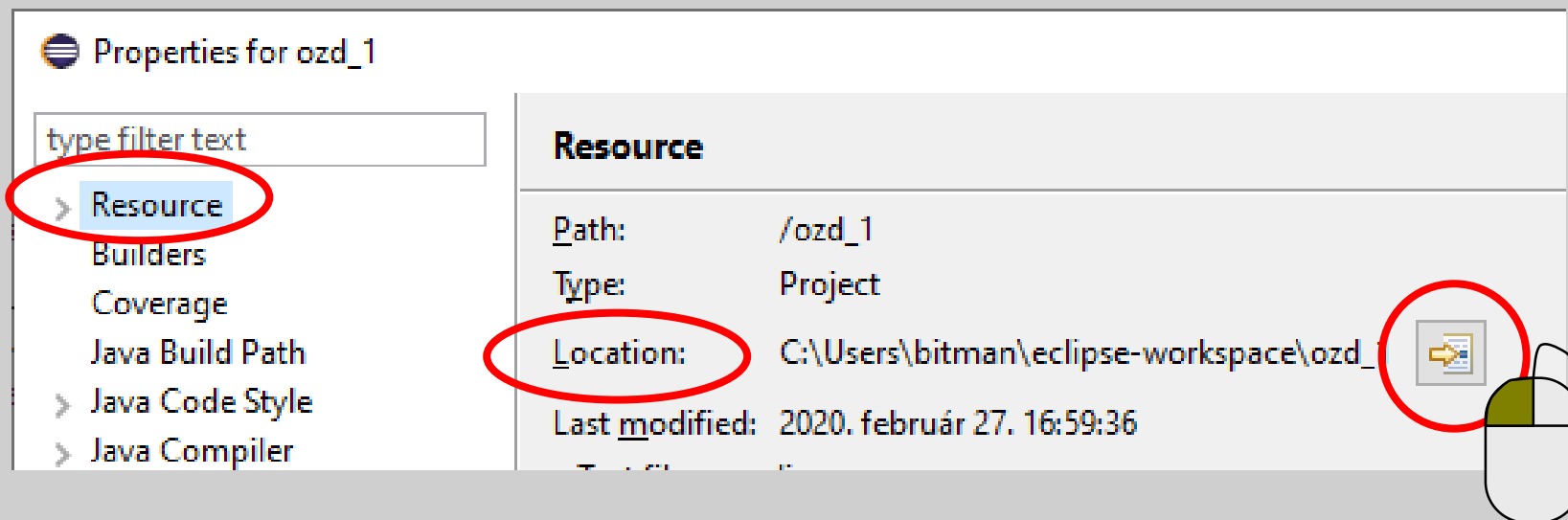
```
public static void main(String[] args) {
 int rndNum = randomNumber();
 ArrayList<String> ts = new ArrayList<String>();

 while (!success) {
 String tippS = readTip();
 ts.add(tippS);
 if (check(tippS)) {
 success = evaluate(rndNum, tippS);
 }
 }
 fileWriter(ts);
}
```



# A fájl megkeresése

- Projekt nevén jobb klikk
  - Menüből: Properties (legalul)





Question  
van?



# VÉGE

