

# Objektumorientált programozás

Java nyelvi alapok

*Goldschmidt Balázs*

*balage@iit.bme.hu*



# ***Bevezető***

# Programozás kurzusok

- Programozás alapjai: *Strukturált programozás*
  - Változók, vezérlés, függvények, adatszerkezetek, algoritmusok, stb.
  - Nyelv: *python*
- Objektorientált programozás: *OO fogalmak*
  - Objektumok, osztályok, egységbezárás, öröklés, polimorfizmus, stb.
  - Nyelv: *Java*
- Eseményvezérelt és vizuális programozás
  - Nyelv: *C#*

# TIOBE Index (népszerűség, 12 havi átlag)

© 2016 TIOBE software BV

Programming Language	2016	2011	2006	2001	1996	1991
Java	1	1	1	3	17	-
C	2	2	2	1	1	1
C++	3	3	3	2	2	2
C#	4	5	6	11	-	-
Python	5	6	7	25	23	-
PHP	6	4	4	8	-	-
JavaScript	7	9	8	7	21	-
Visual Basic .NET	8	29	-	-	-	-
Perl	9	8	5	4	3	-
Ruby	10	10	21	32	-	-

# J2SE keretrendszer

- Java alapjai egyszerűek
  - egyértelmű szintaxis
  - hatalmas API
- Java programozás olyan, mint a lego
  - meglévő elemekből építkezünk
  - minden implementálva van
    - általában jobban, mint ahogy mi csinálnánk
  - a valós tudás az API ismerete
  - szintaxis és API eltér a verziók között
    - utolsó verzió: 11 (2018-09-25)

JDK

JRE

Java SE  
API

Compact  
Profiles

<u>Java Language</u>	Java Language						
	java	javac	javadoc	jar	javap	jdeps	Scripting
<u>Tools &amp; Tool APIs</u>	Security	Monitoring	JConsole	VisualVM	JMC	JFR	
	JPDA	JVM TI	IDL	RMI	Java DB	Deployment	
	Internationalization		Web Services		Troubleshooting		
<u>Deployment</u>	Java Web Start			Applet / Java Plug-in			
	JavaFX						
<u>User Interface Toolkits</u>	Swing		Java 2D	AWT	Accessibility		
	Drag and Drop		Input Methods	Image I/O	Print Service	Sound	
<u>Integration Libraries</u>	IDL	JDBC	JNDI	RMI	RMI-IIOP	Scripting	
<u>Other Base Libraries</u>	Beans	Security	Serialization	Extension Mechanism			
	JMX	XML JAXP	Networking	Override Mechanism			
	JNI	Date and Time	Input/Output	Internationalization			
	lang and util						
<u>lang and util Base Libraries</u>	Math	Collections	Ref Objects	Regular Expressions			
	Logging	Management	Instrumentation	Concurrency Utilities			
	Reflection	Versioning	Preferences API	JAR	Zip		
<u>Java Virtual Machine</u>	Java HotSpot Client and Server VM						



# ***Java alapok***

# Java alapok

- Minden osztály vagy objektum
  - nincsenek globális függvények és változók
  - alkalmazás struktúrája:
    - python: (globális tér >) függvény > utasítás
    - java: (globális tér > csomag >) osztály > metódus > utasítás
- Kétfajta típus (mint pythonban)
  - primitív (int, double, boolean, ...)
    - a változó értéket tárol
  - objektum (String, tömb, List, ...)
    - a változó referenciát tárol
    - memóriamodell megfelel a pythonban megszokottnak



# Első program: Hello world

```
# üdvözlét, python  
print("Helló, világ!")
```

Egysoros  
megjegyzés

```
// üdvözlét, Java (Hello.java)  
public class Hello {  
    static public void main(String[] args) {  
        System.out.println("Helló, világ!");  
    }  
}
```

Osztály

Metódus

A rendszer

Szabványos kimenet

Utasítás vége

# Alapvető szintaktikai eltérések

- Minden változónak van típusa
  - első használat előtt meg KELL adni
  - továbbiakban nem változhat
- Minden függvénynek
  - van visszatérési típusa és paraméter-típusai
    - fejlécben meg KELL adni
- Minden blokk zárójelezve
  - nincs szintaktikai jelentősége az indentálásnak
- Minden utasítás végén pontosvessző


# Fordítás és futtatás

## ■ Ökölszabály

- minden osztály saját fájlba

  - `class Hello` →  `Hello.java`

- minden osztályhoz saját bytecode (class) fájlt készít a fordító

- `> javac Hello.java` →  `Hello.class`

## ■ A JVM a kiválasztott osztály *main* metódusát futtatja

- JVM (Java Virtual Machine): a byte-code-ot értelmező program

- `> java Hello`



# ***Alaptípusok, operátorok, utasítások***

# Egyszerű típusok és változók

## ■ Egyszerű típusok

- ***boolean***: logikai
- *char*: karakter (16bit unicode)
- *byte*, *short*, ***int***, *long*: 8, 16, 32, 64 bites egész
- *float*, ***double***: 32 és 64 bites lebegőpontos valós

## ■ Változók deklarációja

- típus, név és opcionális kezdőérték

```
boolean flag; // inicializálatlan!!!  
int a = 13;  
double d = 3.14, f; // d inicializált, f nem az
```

# Példa változók használatára

```
#python
nev = "Gáz Géza"
print("Szia, " + nev + "!")

a = 3
b = 4
c = math.sqrt(a*a + b*b)
```

Gyakori,  
összetett típus

```
//Java, (metódus belsejében!!!)
String nev = "Gáz Géza";
System.out.println("Szia, " + nev + "!");
```

Értékkadás

```
double a = 3, b = 4, c;
c = Math.sqrt(a*a + b*b);
```

Inicializálás

# Beolvasás

- Javaban a kiírás kicsit körülményes
  - kiírás szabványos kimenetre: *System.out.println(...)*
  - oka: minden függvény osztályban, nincs globális fv
- Beolvasás még körülményesebb
  - szabványos bemenet: *System.in*
  - ebből nehéz kényelmesen olvasni
  - segítség: *Scanner*
    - speciális osztály a kényelmes beolvasáshoz
    - minden primitív és fontos típushoz saját függvényvel: *nextInt, nextDouble, nextString, stb.*

# Beolvasás példa

```
#python
import math
print("Mennyi a kör sugara?")
sugar = float(input())
print("Kerület =", 2 * sugar * math.pi)
print("Terület =", sugar**2 * math.pi)
```

```
//Java (metódusban!!!)
Scanner input = new Scanner(System.in);
System.out.println("Mennyi a kör sugara?");
double sugar = input.nextDouble();
System.out.println("Kerület =" + (2 * sugar * Math.PI));
System.out.println("Terület =" + (sugar*sugar * Math.PI));
```

String-  
összeadás

Nincs hatványozó  
operátor



# Teljes program: kör és sugár

```
// Java (Circle.java)
import java.util.Scanner;

public class Circle {
    static public void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.println("Mennyi a kör sugara?");
        double sugar = input.nextDouble();
        System.out.println("Kerület ="
            + (2 * sugar * Math.PI));
        System.out.println("Terület ="
            + (sugar*sugar * Math.PI));

    }
}
```

Scanner és csomagja.  
Az IDE automatikusan  
generálja az importot

Parancssori  
argumentumok

# Műveletek számokkal

## ■ Aritmetikai

- **+**, **-**, **\***, **/**, **%** : mint pythonban
- nincs: **\*\***, **//**

## ■ Inkrementáló, dekrementáló

- **++**, **--** : egy értékkel növel vagy csökkent
- elöl áll: azonnali hatás
- hátul áll: késleltetett hatás

```
int a = 3, b = 2, c = 5;
double d;
c = a/b; // c==1
d = a/b; // d==1.0
d = 1.0*a/b; // d=1.5
```

```
int e = 3, f;
f = e++; // f==3, e==4
f = ++e; // f==5, e==5
```

# Műveletek

- Összehasonlítás

- `==`, `!=`, `<`, `>`, `<=`, `>=` : mint pythonban

- Típusváltás (kasztolás)

- **(típus)** : kifejezés típusát megváltoztatja

```
double a = 3, b = 2, c = 5;

c = a/b; // c = 1.5
c = (int)a/(int)b; // c = 1.0
```

# Automatikus konverzió

- Műveletek különböző típusok között
  - ökölszabály: "nagyobb" típusra konvertál a fordító
    - byte és short → short
    - int és long → long
    - int és double → double
  - "kicsinyítés" explicit módon
    - explicit kasztolás kell, különben fordítási hiba

```
double d1 = 1.0, d2 = 2.2;  
int i1 = 3, i2 = 2;  
d1 = i1/i2; //d1==1.0  
i1 = d1*d2; //HIBA  
i1 = (int)(d1*d2); //i1==2
```

int → double

double → int

# Műveletek bitekkel

## ■ Bitenkénti műveletek

- `&`, `|`, `^`, `~` : mint pythonban (és, vagy, kiz.vagy, negálás)

## ■ Eltolás

- `<<`, `>>`, `>>>`: balra, ismétlődő ill. nullával töltött jobbra

```
byte a = 3;    // 00000011
byte b = 5;    // 00000101
byte c;

c = a & b;    // c==00000001
c = a | b;    // c==00000111
c = a ^ b;    // c==00000110
c = ~b;       // c==11111010
```

```
byte n, m = -6; // 11111010
n = m << 3;    // n== 11010000
n = m >> 2;    // n==11111110
n = m >>> 2;   // n==00111110
```

# Műveletek logikai értékekkel

## ■ Lusta kiértékelés (lazy)

- **&&**, **||** : and, or
- jobb oldal csak akkor értékelődik, ha számít

## ■ Teljes kiértékelés (mellékhatásoknál fontos)

- **&**, **|**, **^**, **!** : and, or, *xor*, not

```
if (a<b && b<c) { ... }
```

*b<c* -t csak akkor értékeli, ha *a<b* igaz

```
if (a<b & b<c) { ... }
```

*b<c* -t mindig kiértékeli

# Műveletek értékadással

- `=` : egyszerű értékadás
- `~=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=` : összetett értékadás (mint pythonban)
  - pl: `x *= a+b; → x = x*(a+b)`
- a teljes kifejezés értéke az értékül adott kifejezés értéke

```
int a=1, b=2, c=3, d=5, e=6;
a = b;           // a == 2
b = c = e;      // b==6, c==6
b *= a+d;       // b==42
e = a += b;     // a==44, e==44
```

# Egyéb műveletek

## ■ Mezőkiválasztás (attribútum, metódus)

- `.` : mint python objektumok esetén

```
System.out.println("Helló");
```

## ■ Elágazás

- `? :` : feltételtől függő érték

```
int x, a=5, b=10;  
x = (a<b) ? a+1 : b+2; // x==6
```

feltétel

igaz ág

hamis ág



# Műveletek precedenciája

	Művelet	Precedencia
1	posztfix	<i>expr++ expr--</i>
2	egy operandusú	<i>++expr --expr</i> <i>+expr -expr</i> <i>~ !</i>
3	multiplikatív ->	<i>* / %</i>
4	additív ->	<i>+ -</i>
5	eltolás ->	<i>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</i>
6	relációs ->	<i>&lt; &gt; &lt;= &gt;=</i> <i>instanceof</i>
7	azonosság ->	<i>== !=</i>

	Művelet	Precedencia
8	bitenkénti AND ->	<i>&amp;</i>
9	bitenkénti XOR ->	<i>^</i>
10	bitenkénti OR ->	<i> </i>
11	logikai AND ->	<i>&amp;&amp;</i>
12	logikai OR ->	<i>  </i>
13	elágazó	<i>? :</i>
14	értékadás <-	<i>= += -= *=</i> <i>/= %=</i> <i>&amp;= ^=  =</i> <i>&lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</i>

**-> és <- az  
asszociativitás iránya**

# Vezérlési szerkezetek

- Nagy különbség python és Java között
  - indentálás csak szebbé tesz
    - fordítóra nem hat
    - használjuk úgy, mint pythonban!
  - utasítások végén pontosvessző
  - blokk-képzés kapcsos zárójellel

```
#python
szam = int(input())
if szam % 2 == 0:
    print("páros")
else:
    print("páratlan")
```

```
//Java (metódusban)
//... (scanner, stb)
int szam = input.nextInt();
if (szam % 2 == 0) {
    System.println("páros");
} else {
    System.println("páratlan");
}
```

# Elágazás: feltételes végrehajtás

**if** (logikai kifejezés) *utasítás* **else** *utasítás*

- *utasítás* lehet egy sor vagy egy blokk
- *logikai kifejezés* mindig boolean típusú

```
#python  
  
szam = int(input())  
if szam % 2 == 0:  
    print("páros")  
else:  
    print("páratlan")
```

```
//Java (metódusban)  
//... (scanner, stb)  
int szam = input.nextInt();  
if (szam % 2 == 0) {  
    System.println("páros");  
} else {  
    System.println("páratlan");  
}
```

# Ciklus: ismétlődő végrehajtás

**while** (*logikai kifejezés*) *utasítás*

Elöltesztelő

**do** *utasítás* **while** (*logikai kifejezés*);

Hátultesztelő

□ *utasítás* lehet egy sor vagy egy blokk

□ *logikai kifejezés* mindig boolean típusú

```
#python
```

```
n = int(input())
x = 1
while x <= n:
    print(x, end=" ")
    x = x+1

print(".")
```

```
//Java (metódusban)
//... (scanner, stb)
int n = input.nextInt();
int x = 1;
while (x <= n) {
    System.out.print(x+" ");
    x = x+1;
}
System.out.println(".");
```

# Ciklus: ismétlődő végrehajtás

**for** (*inicializálás; logikai kifejezés; léptetés*) *utasítás*

- *utasítás* lehet egy sor vagy egy blokk
- *logikai kifejezés* mindig boolean típusú

```
#python
```

```
n = int(input())
```

```
for x in range(1, n, 2):  
    print(x, end=" ")
```

```
print(". ")
```

```
//Java (metódusban)
```

```
//... (scanner, stb)
```

```
int n = input.nextInt();
```

```
for (int x = 1; x <= n; x+=2) {  
    System.out.print(x+" ");
```

```
}
```

```
System.out.println(". ");
```

# Ciklus: ismétlődő végrehajtás

**for** (*inicializálás; logikai kifejezés; léptetés*) *utasítás*

□ általánosított léptető while-ciklus

```
//Java (metódusban)
//... (scanner, stb)
int n = input.nextInt();
int x = 1;
while (x <= n) {
    System.out.print(x+" ");
    x += 2;
}
System.out.println(".");
```

```
//Java (metódusban)
//... (scanner, stb)
int n = input.nextInt();
for (int x = 1; x <= n; x+=2) {
    System.out.print(x+" ");
}
System.out.println(".");
```

# Ciklusban ugrás

## ■ Ciklusból kiugrás

- **break** : mint pythonban
- példa: számok beolvasása és összegzése 0-ig

```
//Java (metódusban)
//... (scanner, stb)
int sum = 0;
while (true) {
    int n = input.nextInt();
    if (n == 0) break;
    sum += n;
}
System.out.println(sum);
```

# Ciklusban folytatás

## ■ Ciklus elejére ugrás

- **continue** : mint pythonban
- példa: 1-100 közötti hárommal osztható számok kiírása és összege

```
//Java (metódusban)
int sum = 0;
for (int i = 1; i < 100; i++) {
    if (i%3 == 0) continue;
    sum += i;
    System.out.println(i);
}
System.out.println("Szum:"+sum);
```



# Egymásba ágyazott ciklusok

## ■ Belső ciklusból kilépés

- példa: keressük a szorzótáblában az első előfordulást

```
//Java (metódusban)
//... (scanner, stb)
int n = input.nextInt();
int i=0,j=0;
boolean megvan = false;
for (i = 1; i <= 10; i++) {
    for (j = 1; j <= 10; j++) {
        if (i*j == n) { megvan = true; break; }
    }
    if (megvan) break;
}
System.out.println((i*j!=121) ? i+", "+j : "Nincs");
```

# Egymásba ágyazott ciklusok

- Belső ciklusból kilépés
  - címke használatával

```
//Java (metódusban)
//... (scanner, stb)
int n = input.nextInt();
int i=0,j=0;

ciklus: for (i = 1; i <= 10; i++) {
    for (j = 1; j <= 10; j++) {
        if (i*j == n) break ciklus;
    }
}

System.out.println((i*j!=121) ? i+", "+j : "Nincs");
```

# Többszörös elágazás

- C-s örökség, ritkán használjuk
  - főleg állapotgépekben
    - egy állapot egy elágazás
- Struktúra
  - *switch* a fő blokk
  - *case* egész típus :
    - lehet char, String, enum is
  - *break* minden külön eset után!

```
//Java (metódusban)
//... (scanner, stb)
int n = input.nextInt();
switch (n) {
    case 1: a = 3;
           b = 2;
           break;
    case 2: a = 2;
           b = 3;
    case 3: c = 1;
           break;
    default: a = b = c = 0;
}
```



# ***Összetett típusok, tömbök***

# Összetett típusok

- Tömbök, Stringek stb. mind összetettek
- Változó csak referenciát tárol
  - mint pythonban
  - python *None* Javaban *null*
- Értékadás a referenciát módosítja
  - referált objektum nem változik

```
String s = "12345"; // String literál  
s = "hello"; // korábbi referencia elveszik,  
              // de az "12345" String a  
              //memóriában marad  
String q = null;
```

# Tömbök

## ■ Egyszerű tömb definiálása

```
int a[] = new int[13]; // 13 elemű, int-ek vannak benne  
double[] d = new double[20]; // 20 db double
```

## ■ Tömbök merevek

méretük konstans

■ létrehozáskor rögzül

■ *length* attribútummal lekérdezhető

indexelés 0-tól

alapértelmezetten nullára inicializálódnak az elemek

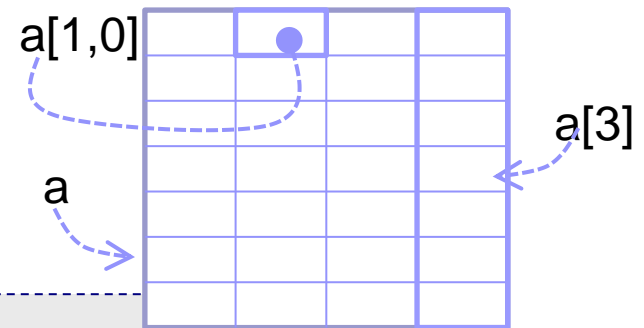
■ referencia esetén *null*

# Tömbök

## ■ Többdimenziós tömb

- tömbök tömbje
- mindig referencia a tömb is
- eltérhet az altömbök mérete

```
int[][] a = new int[4][7];  
  
int[][] b = new int[4][];  
for (int i = 0; i < b.length; i++) {  
    b[i] = new int[i*2+1];  
}  
b[2][3] = 5;
```



Indexek

b

0,0	1,0	2,0	3,0
	1,1	2,1	3,1
	1,2	2,2	3,2
		2,3	3,3
		2,4	3,4
			3,5
			3,6

Értékek

0	0	0	0
	0	0	0
	0	0	0
		5	0
		0	0
			0
			0

39

# Tömbök használata

## ■ Feladat:

- olvassunk be 10 számot, és írjuk ki az átlagnál nagyobbakat!

```
int[] t = new int[10];
int n, sum;
for (int i = 0; i < 10; i++) {
    n = input.nextInt();
    sum += n;
    t[i] = n;
}
for (int i = 0; i < 10; i++) {
    if (t[i]*10 > sum)
        System.out.println(t[i]);
}
```



# Tömbök használata

## ■ Feladat:

- olvassunk be 10 számot, és írjuk ki az átlagnál nagyobbakat!
- 10 csak egyszer!

```
int[] t = new int[10];
int n, sum;
for (int i = 0; i < t.length; i++) {
    n = input.nextInt();
    sum += n;
    t[i] = n;
}
for (int i = 0; i < t.length; i++) {
    if (t[i]*t.length > sum)
        System.out.println(t[i]);
}
```

# Tömbök használata

## ■ Feladat:

- olvassunk be 10 számot, és írjuk ki az átlagnál nagyobbakat!
- 10 csak egyszer!
- iteráljunk az elemeken!

**Foreach** ciklus.  
Aktuális elem: *x*  
Tároló: *t*  
Pythonhoz hasonló

```
int[] t = new int[10];
int n, sum;
for (int i = 0; i < t.length; i++) {
    n = input.nextInt();
    sum += n;
    t[i] = n;
}
for (int x : t) {//python for x in t:
    if (x *t.length > sum)
        System.out.println(x );
}
```



# *Függvények alapjai*

# Függvények definiálása

- Függvények Java megnevezése: *metódus*
  - mindig egy osztályban kell legyen
  - szerkezet
    - láthatóság és módosítók (később, most csak *static*) [1]
    - visszatérési érték típusa (*void*, ha nincs) [2]
    - név [3]
    - paraméterek típussal (üres, ha nincs) [4]
    - várható kivételek (ha vannak; később) [5]
    - törzs (blokk) [6]

```
static double sum(double x, double y) {return x+y;}
```

1

2

3

4

4

6

# Egyszerű függvény

## ■ Számoljunk négyzetet!

### □ definíció

```
#python  
def negyzet(x):  
    return x*x
```

```
//Java (osztályon belül)  
static double negyzet(double x) {  
    return x*x;  
}
```

### □ használat

```
#python  
print(negyzet(2.3))
```

```
//Java (metóduson belül)  
System.out.println(negyzet(2.3));
```

# Függvény példa

## ■ Teljes példa 1

- Írjunk Java programot, ami kiírja az első 10 négyzetszámot!

```
//Java (Main.java)
public class Main {
    static double negyzet(double x) {
        return x*x;
    }
    public static void main(String[] args) {
        for (int n = 1; n <= 10; n++) {
            System.out.println(negyzet(n));
        }
    }
}
```

# Függvény példa

- Teljes példa 2.1

- Írjunk Java programot, ami kiírja az első 10 prímszámot!

```
//Java (Main.java)
public class Main {
    static boolean prime(int x) {
        for (int i = 2; i <= x/2; i++) {
            if (x%i == 0)
                return false;
        }
        return true;
    }
    //...
```

# Függvény példa

## ■ Teljes példa 2.2

- Írjunk Java programot, ami kiírja az első 10 prímszámot!

```
//...
public static void main(String[] args) {
    for (int cnt = 0, n = 2; cnt <= 10; n++) {
        if (prime(n)) {
            System.out.println(n);
            cnt++;
        }
    }
}
```



# Függvények és változók

## ■ Változók élettartama

### lokális változó

- míg a függvény (blokk) véget nem ér

### paraméter

- primitív: amíg a függvény vissza nem tér
- objektumreferencia: a hívónál megmarad

### visszatérési érték

- amíg a hívó használja

## ■ Referencia típusok

- mindig *new*-val jön létre a referencia

# Függvények paraméterezése

- Hívónak minden paramétert meg kell adnia
  - a sorrend és a típus számít
    - nincs név szerinti paraméter-értékadás
    - nincs default érték
- Paraméterek helyes típussal
  - fordító ellenőrzi a típus-kompatibilitást
    - pl. *double* helyett adhatunk *int*-et, de fordítva nem
- Visszatérési érték
  - nem kötelező felhasználni
  - ha átvesszük, típus-kompatibilitás fontos

# Paraméterek: típus-kompatibilitás

```
//Java (Main.java)
public class Main {
    static double negyzet(double x) {
        return x*x;
    }

    public static void main(String[] args) {

        for (int n = 1; n <= 10; n++) {
            System.out.println(negyzet(n)); //OK, int->double
        }
        int i = negyzet(12); // fordítási hiba: double->int
        int k = (int)negyzet(12); // OK: explicit kasztozás
        double q = negyzet(12); // OK: int -> double
        negyzet(2.3); // nem használjuk az eredményt, de OK
    }
}
```



# *Memóriakezelés*

# Memória kezelés alapjai

- Java (és python) beépített szemétygyűjtővel
  - garbage collector (GC)
  - **new** : objektum lefoglalása a heap-en
    - élettartam: amíg van rá referáló változó vagy attribútum
- GC összegyűjti és törli a nem referáltakat
  - automatikus
  - okos algoritmus, köröket, hurkokat is jól kezeli
  - nem okoz objektum-vesztést

# Lokális változók a memóriában

```
//Java (osztályon belül)
static double pythagoras(double a,
                          double b) {
    double c2 = a*a+b*b;
    double c = Math.sqrt(c2);
    return c;
}
static double dist(double x1, double y1,
                   double x2, double y2) {
    double dx = x2-x1, dy = y2-y1;
    double d = pythagoras(dx,dy);
    return d;
}
```

```
//Java (osztályon belül), foo függvény
double x = dist(3,0, 0,4);
System.out.println(x);
```

Stack (verem)



# Lokális változók a memóriában

```
//Java (osztályon belül)
static double pythagoras(double a,
                        double b) {
    double c2 = a*a+b*b;
    double c = Math.sqrt(c2);
    return c;
}
static double dist(double x1, double y1,
                  double x2, double y2) {
    double dx = x2-x1, dy = y2-y1;
    double d = pythagoras(dx,dy);
    return d;
}
```

```
//Java (osztályon belül), foo függvény
double x = dist(3,0, 0,4);
System.out.println(x);
```

Stack (verem)



foo lokális változói

double x : ?

# Lokális változók a memóriában

```
//Java (osztályon belül)
static double pythagoras(double a,
                          double b) {
    double c2 = a*a+b*b;
    double c = Math.sqrt(c2);
    return c;
}
static double dist(double x1, double y1,
                   double x2, double y2) {
    double dx = x2-x1, dy = y2-y1;
    double d = pythagoras(dx,dy);
    return d;
}
```

```
//Java (osztályon belül), foo függvény
double x = dist(3,0, 0,4);
System.out.println(x);
```

Stack (verem)

double x1 : 3  
double y1 : 0  
double x2 : 0  
double y2 : 4  
double dx : ?  
double dy : ?  
double d : ?

double x : ?

*dist* lokális változói

*foo* lokális változói



# Lokális változók a memóriában

```
//Java (osztályon belül)
static double pythagoras(double a,
    double b) {
    double c2 = a*a+b*b;
    double c = Math.sqrt(c2);
    return c;
}
static double dist(double x1, double y1,
    double x2, double y2) {
    double dx = x2-x1, dy = y2-y1;
    double d = pythagoras(dx,dy);
    return d;
}
```

```
//Java (osztályon belül), foo függvény
double x = dist(3,0, 0,4);
System.out.println(x);
```

*Math.sqrt* lokális változói

*pythagoras* lokális változói

*dist* lokális változói

*foo* lokális változói

Stack (verem)

double x : 25  
?????

double a : -3  
double b : 4  
double c2 : 25  
double c : ?

double x1 : 3  
double y1 : 0  
double x2 : 0  
double y2 : 4  
double dx : -3  
double dy : 4  
double d : ?

double x : ?

# Lokális változók a memóriában

```
//Java (osztályon belül)
static double pythagoras(double a,
                          double b) {
    double c2 = a*a+b*b;
    double c = Math.sqrt(c2);
    return c;
}
static double dist(double x1, double y1,
                  double x2, double y2) {
    double dx = x2-x1, dy = y2-y1;
    double d = pythagoras(dx,dy);
    return d;
}
```

```
//Java (osztályon belül), foo függvény
double x = dist(3,0, 0,4);
System.out.println(x);
```

Stack (verem)

double a : -3  
double b : 4  
double c2 : 25  
double c : 5

double x1 : 3  
double y1 : 0  
double x2 : 0  
double y2 : 4  
double dx : -3  
double dy : 4  
double d : 5

double x : 5

*pythagoras* lokális változói

*dist* lokális változói

*foo* lokális változói

# Lokális változók a memóriában

```
//Java (osztályon belül)
static double pythagoras(double a,
                          double b) {
    double c2 = a*a+b*b;
    double c = Math.sqrt(c2);
    return c;
}
static double dist(double x1, double y1,
                   double x2, double y2) {
    double dx = x2-x1, dy = y2-y1;
    double d = pythagoras(dx,dy);
    return d;
}
```

```
//Java (osztályon belül), foo
double x = dist(3,0, 0,4);
System.out.println(x);
```

*System.out.println*  
lokális változói

*foo* lokális változói

Stack (verem)

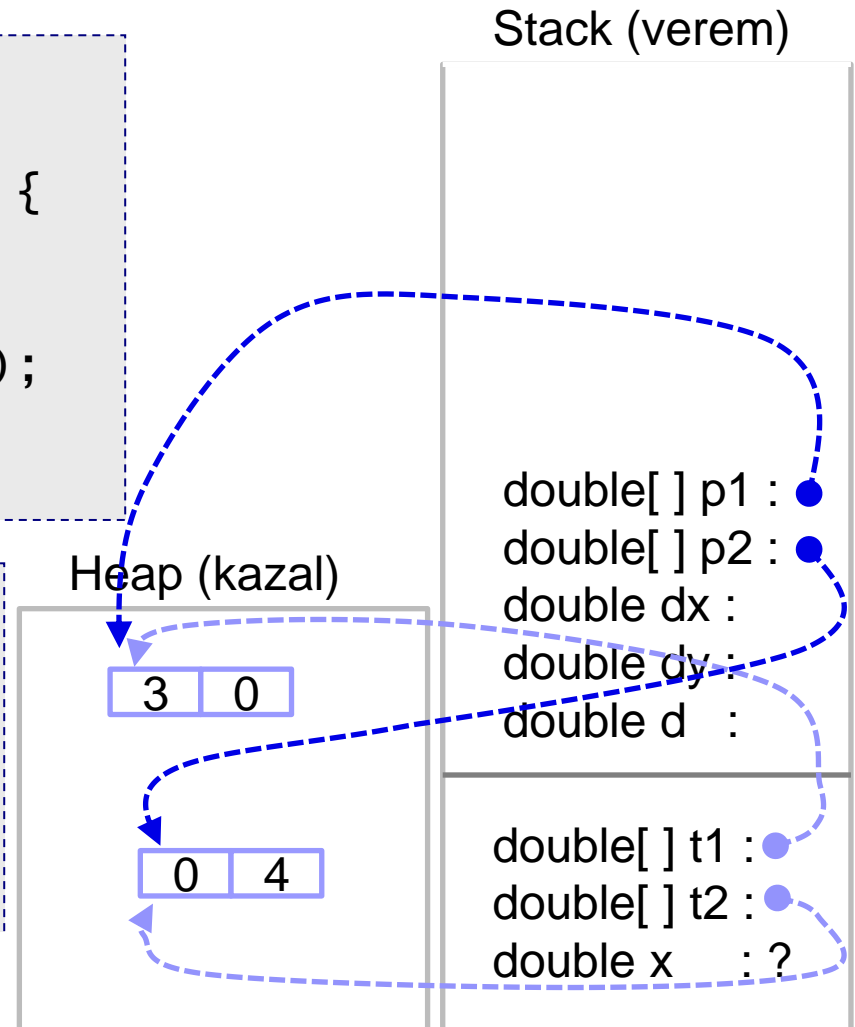
Object x : 5  
??????

double x : 5

# Tömbök a memóriában 1

```
//Java (osztályon belül)
static double dist(double[] p1,
                  double[] p2) {
    double dx = p2[0]-p1[0],
    double dy = p2[1]-p2[0];
    double d = pythagoras(dx,dy);
    return d;
}
```

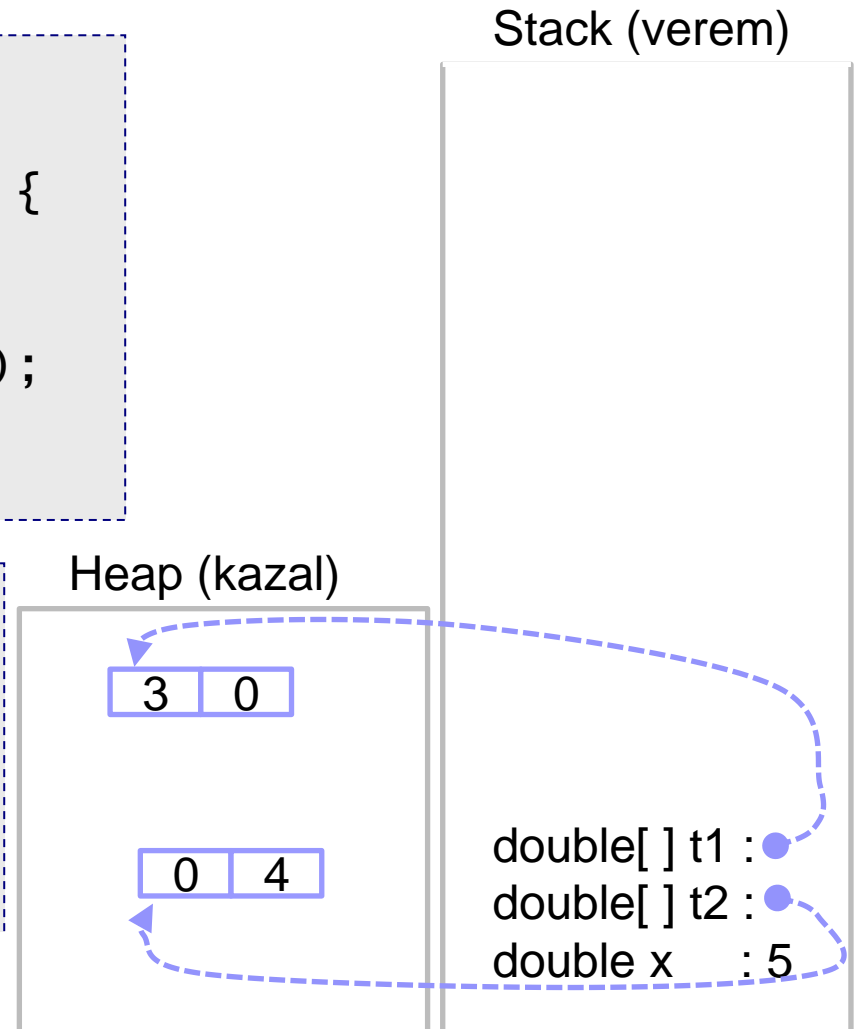
```
//Java (osztályon belül), főprogram
double[] t1 = new double[2];
double[] t2 = new double[2];
//... tömbök feltöltése
double x = dist(t1, t2);
System.out.println(x);
```



# Tömbök a memóriában 1

```
//Java (osztályon belül)
static double dist(double[] p1,
                  double[] p2) {
    double dx = p2[0]-p1[0];
    double dy = p2[1]-p2[0];
    double d = pythagoras(dx,dy);
    return d;
}
```

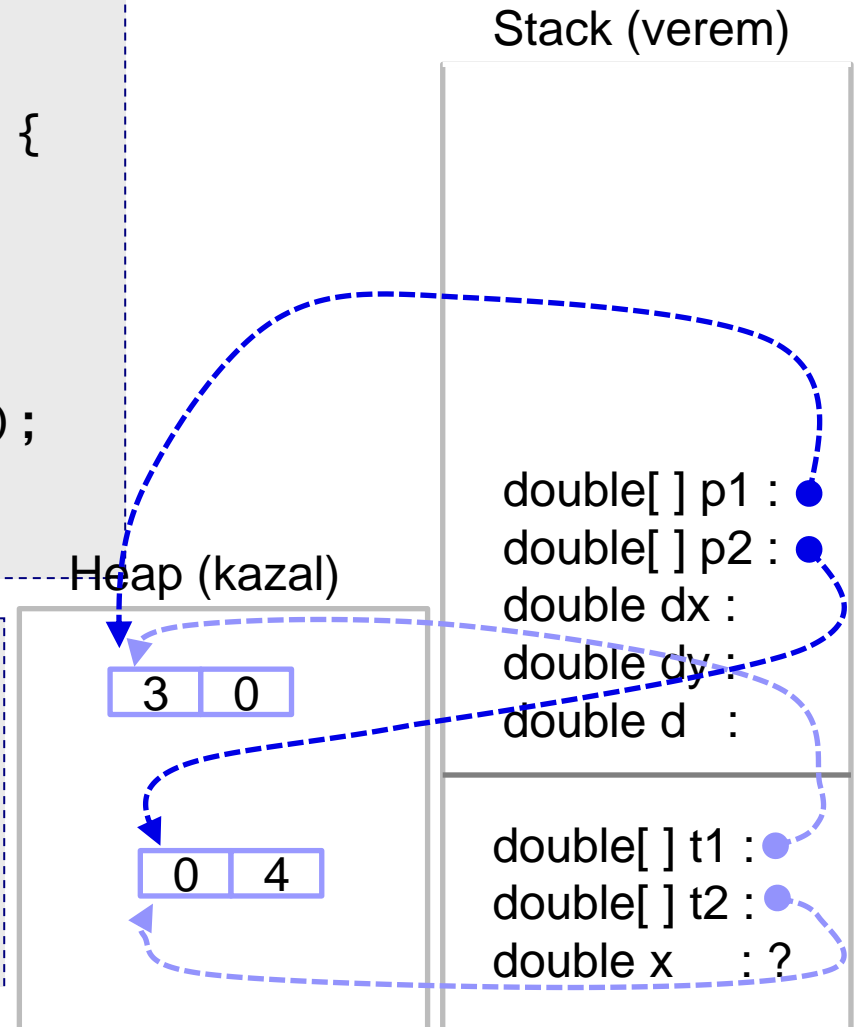
```
//Java (osztályon belül), foo
double[] t1 = new double[2];
double[] t2 = new double[2];
//... tömbök feltöltése
double x = dist(t1, t2);
System.out.println(x);
```



# Tömbök a memóriában 2

```
//Java (osztályon belül)
static double dist(double[] p1,
                  double[] p2) {
    double dx = p2[0]-p1[0];
    double dy = p2[1]-p2[0];
    p1 = new double[3];
    p2[0] = 7;
    double d = pythagoras(dx,dy);
    return d;
}
```

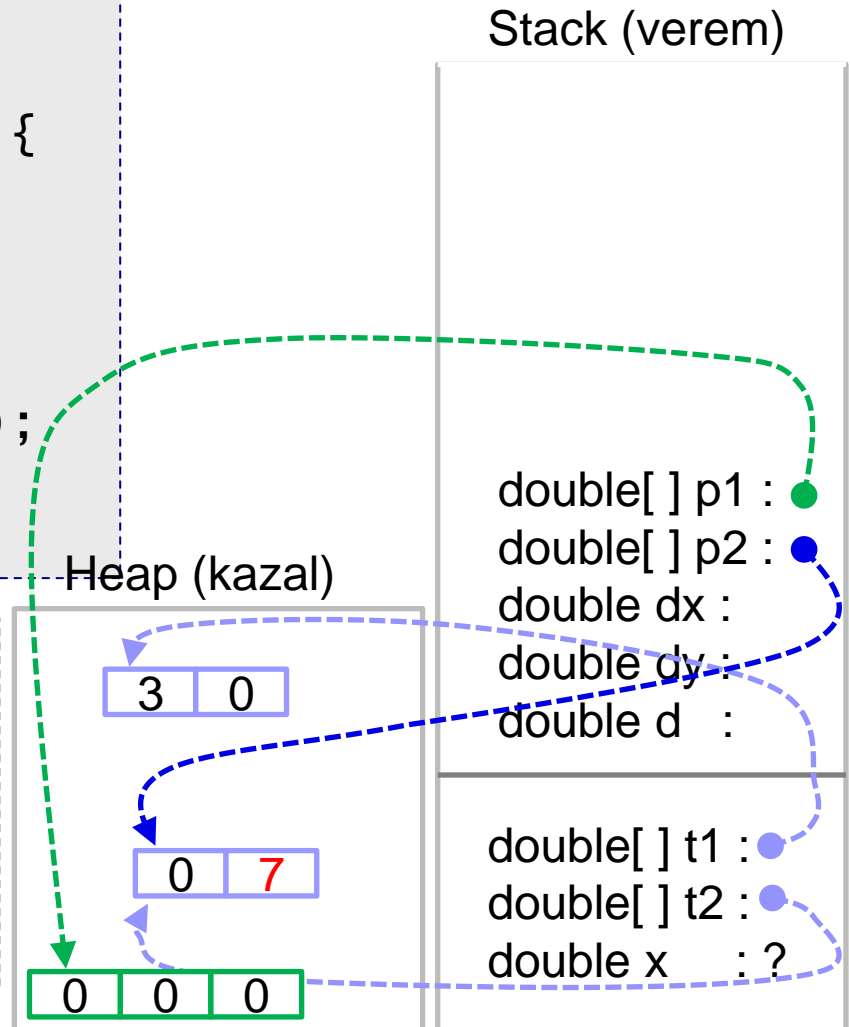
```
//Java (osztályon belül), foo
double[] t1 = new double[2];
double[] t2 = new double[2];
//... tömbök feltöltése
double x = dist(t1, t2);
System.out.println(x);
```



# Tömbök a memóriában 2

```
//Java (osztályon belül)
static double dist(double[] p1,
                  double[] p2) {
    double dx = p2[0]-p1[0];
    double dy = p2[1]-p2[0];
    p1 = new double[3];
    p2[0] = 7;
    double d = pythagoras(dx,dy);
    return d;
}
```

```
//Java (osztályon belül), foo
double[] t1 = new double[2];
double[] t2 = new double[2];
//... tömbök feltöltése
double x = dist(t1, t2);
System.out.println(x);
```



# Flexibilis tömb: *ArrayList*

- Pythonban megismert lista flexibilis
  - nyújtózkodik (append)
- Java sok tároló-osztályt ismer
  - részletek később
- Egyik leggyakoribb: *ArrayList*
  - nyújtózkodik
  - törölhetők az elemei
  - stb.



# ArrayList használata

```
#python
szamok = []
while True:
    n = int(input())
    if n < 0:
        break
    szamok.append(n)
for n in szamok:
    print(n)
```

Tárolt típus.  
(Primitív típusoknál  
nagybetűvel)

```
//Java, metódusban, input van
ArrayList<Integer> szamok =
    new ArrayList<Integer>();
while (true) {
    int n = input.nextInt();
    if (n < 0) { break; }
    szamok.add(n);
for (int n : szamok) {
    System.out.println(n);
}
```

# ArrayList főbb függvényei

- `ArrayList<T>` esetén (T a tárolt típus)
  - `boolean add(T x)`: végéhez fűz (mindig true)
  - `void add(int i, T x)`: i. elem elé beszúr (0. az első)
  - `boolean contains(T x)`: igaz, ha x benne van
  - `int indexOf(T x)`: x első előfordulása egyébként -1
  - `T get(int i)`: i. elemet visszaadja
  - `T remove(int i)`: i. elemet visszadja és törli a listából
  - `boolean remove(T x)`: egyszer törli x-et, ha benne van
  - `clear()`: minden elemet töröl a listából
  - `int size()`: tárolt elemek száma
  - `boolean isEmpty()`: üres-e?
  - ...

# *Csomagok*

# Csomagok alapjai

- Hierarchikus névteret definiálnak
  - hasonló a *module*-okhoz pythonban
- Csomag-hierarchia mappa-hierchiával
  - nevek azonosak, de több gyökérmappánk is lehet
  - forráskód fájljai a megfelelő nevű mappában legyenek
- Forráskódban
  - kötelező definiálni a kódban is
    - `package foo.bar.baz;`
  - importálni csomagnévvel vagy osztálynévvel lehet

# Csomagok és osztályok

- Teljes név: csomag+osztály
  - `foo.bar.baz.MyClass`
- Nevek használata importtal
  - importtal csomagnév használata nélkül elérhető
    - megadja, hogy egy azonosítót hol keressen a fordító
  - névütközés esetén teljes nevet kell használni
    - pl. *List* benne van a *java.util* és *java.awt* csomagban is
  - statikus import mezőkre működik

```
import foo.bar.baz.*;  
import mypack.MyClass;  
import static Math.sin;
```



# ***Kódolási konvenciók***

# Azonosítók írásmódja

- Változók, attribútumok és metódusok
  - `camelCase`, kezdőbetű kicsi, új szavak naggyal
    - `getSecondBiggestNumber()`
    - `int importantVariable;`
- Osztálynevek
  - `CamelCase`, kezdőbetű nagy, új szavak naggyal
    - `StringBuffer`
- Csomagnév
  - kisbetűs
    - `java.util`

# Zárójelezés

## ■ Blokkok esetén

- nyitó zárójel a sor végén

```
while (true) {
```

- zárás új sorban, folytatás a sorban

```
if (a<b) {
```

```
    ...
```

```
} else {
```

```
    ...
```

```
}
```



# Kommentek

## ■ Egysoros

- sor végéig tart

```
int x; // sor végéig tart
```

## ■ Többsoros

- nem szabad egymásba ágyazni

```
int /* ez már többsoros,  
ez is az, még mindig tart  
itt lesz vége: */ y;
```

## ■ Dokumentáló

- mint a többsoros, de dokumentációba bekerül (JavaDoc)

```
/** A pozíció X koordinátája */  
double x;  
/** A pozíció Y koordinátája */  
double y;
```

- python `"""docstring"""`