Objektumorientált programozás

Java nyelvi alapok 2.

Goldschmidt Balázs

balage@iit.bme.hu

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;}
```

Egyszerű függvény

- Számoljunk négyzetet!
  - definíció

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

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

  - használat

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

```java
//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 isPrime(int x) {
    if (x < 2) return false;
    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 (isPrime(n)) {
        System.out.println(n);
        cnt++;
      }
    }
  }
}
```

Függvények és változóik

- 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 kasztolás
    double q = negyzet(12); // OK: int -> double
    negyzet(2.3); // nem használjuk az eredményt, de OK
  }
}
```

Memóriakezelés

Memória kezelés alapjai

- Java (és python) beépített szemétgyűjtővel
  - 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;
}
```

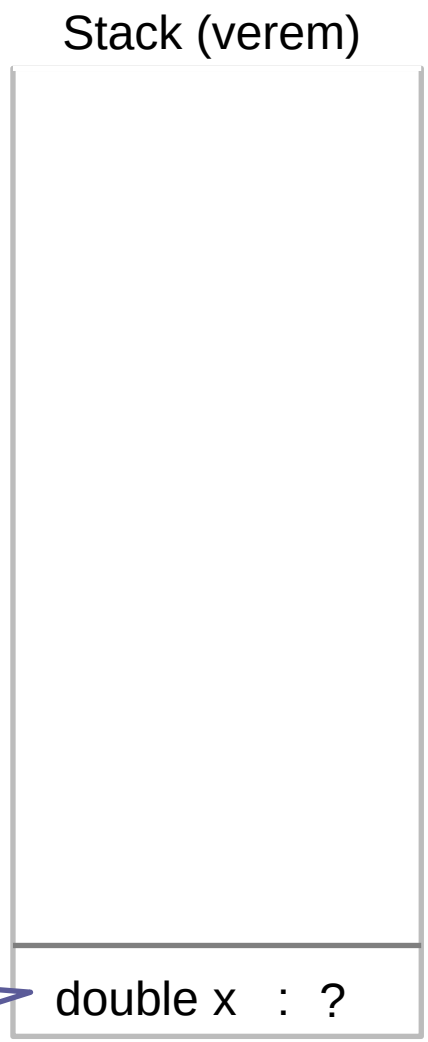
```
//Java (osztályon belül), foo függvény
double x = dist(3,0, 0,4);
System.out.println(x);
```

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);
```



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);
```

dist lokális változói

foo lokális változói

Stack (verem)

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);
```

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
                   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)

pythagoras lokális változói

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

dist lokális változói

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

foo lokális változói

double x	5

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);
```

Stack (verem)

System.out.println lokális változói

Object x : 5
??????

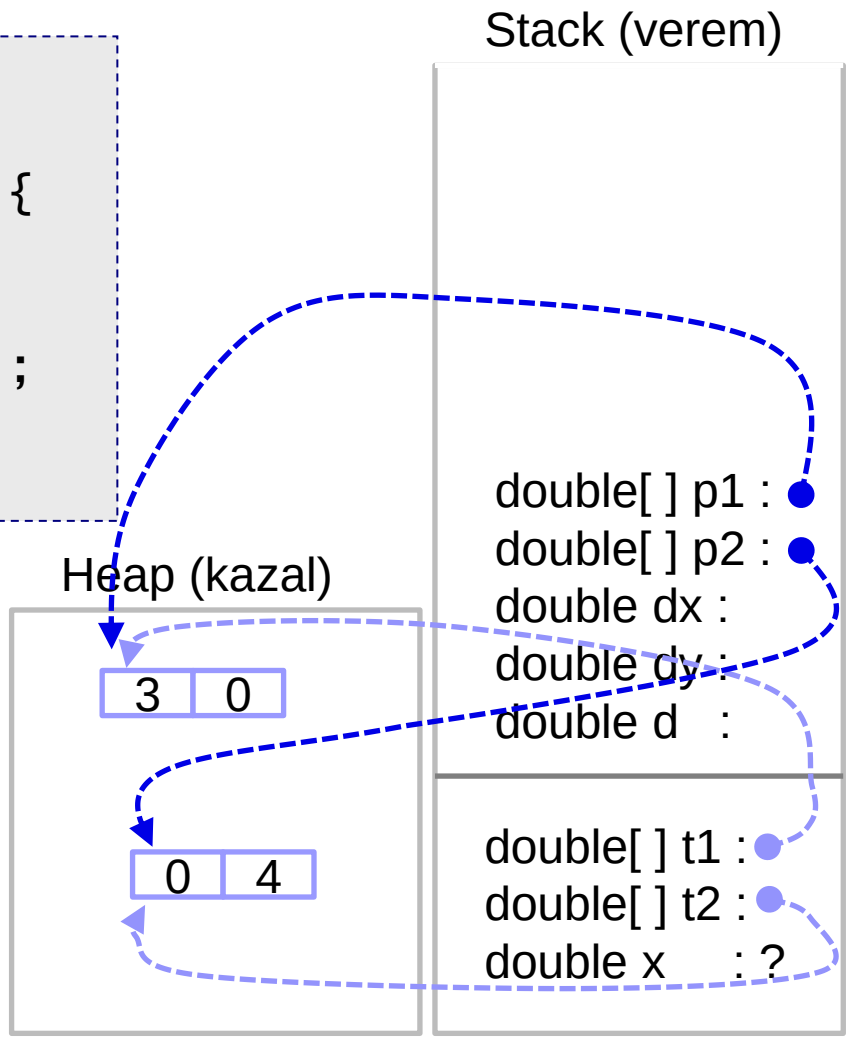
foo lokális változói

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]-p1[1];
    double d = pythagoras(dx,dy);
    return d;
}
```

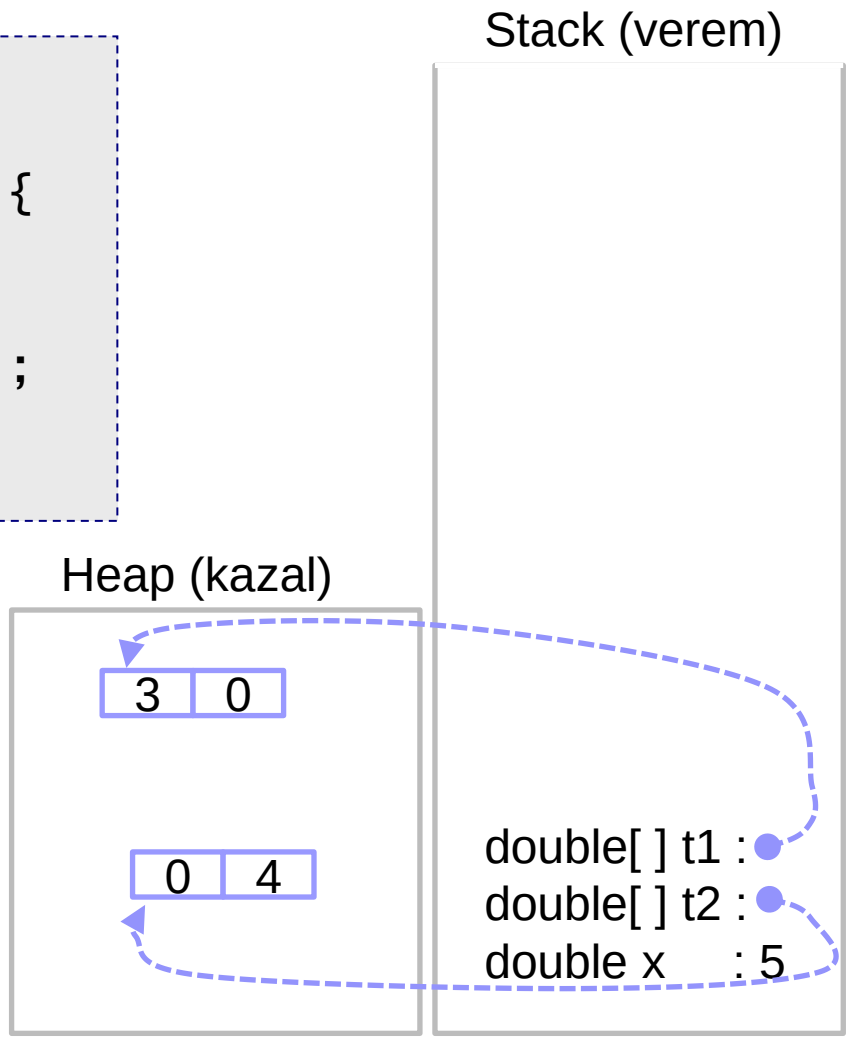
```
//Java (osztályon belül), foo
double[] t1 = new double[2];
double[] t2 = new double[2];
//... tömbök feltöltése
double x = dist(t1, t2);
System.out.println(x);
```



Tömbök a memóriában 1

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

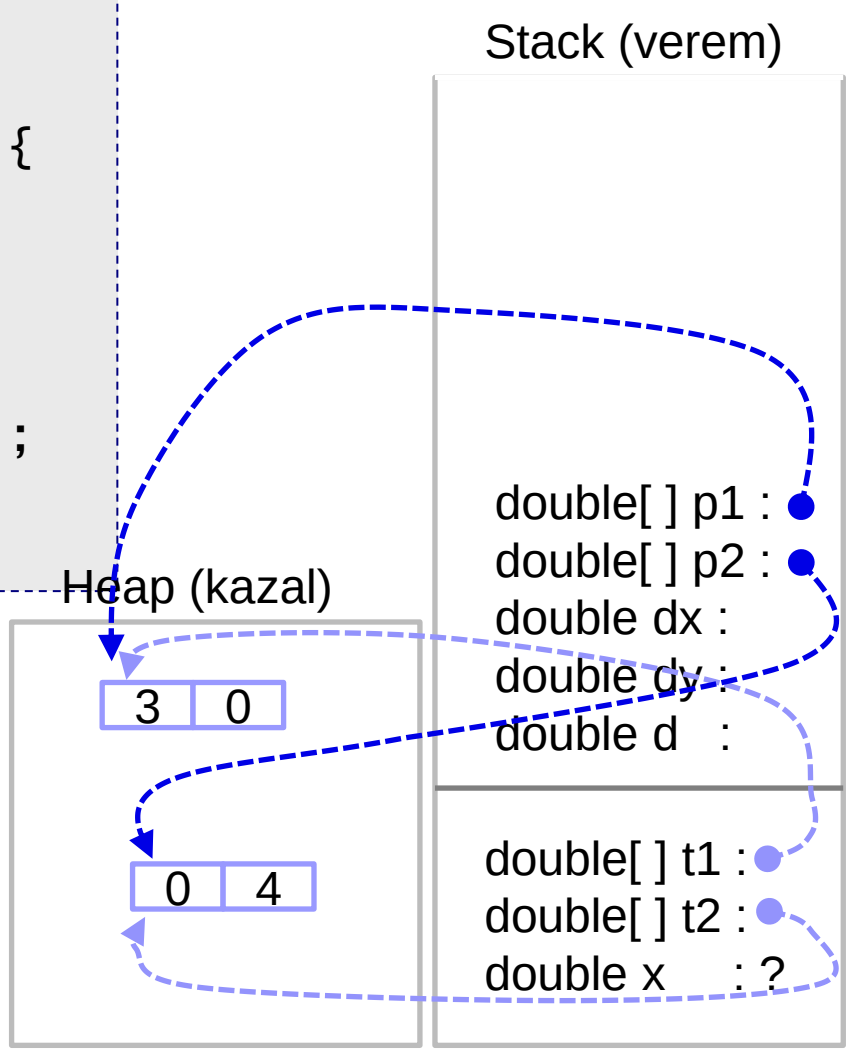
```
//Java (osztályon belül), foo
double[] t1 = new double[2];
double[] t2 = new double[2];
//... tömbök feltöltése
double x = dist(t1, t2);
System.out.println(x);
```



Tömbök a memóriában 2

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

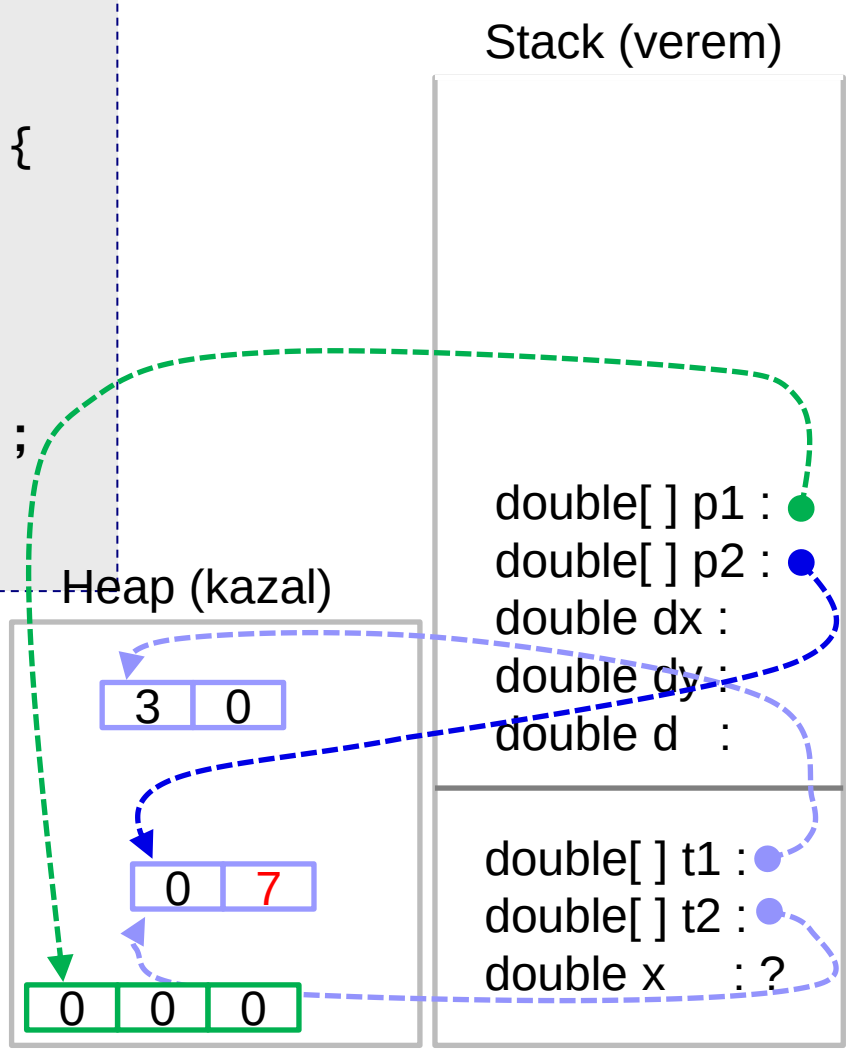
```
//Java (osztályon belül), foo
double[] t1 = new double[2];
double[] t2 = new double[2];
//... tömbök feltöltése
double x = dist(t1, t2);
System.out.println(x);
```



Tömbök a memóriában 2

```
//Java (osztályon belül)
static double dist(double[] p1,
                   double[] p2) {
    double dx = p2[0]-p1[0];
    double dy = p2[1]-p1[1];
    p1 = new double[3];
    p2[1] = 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
#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
//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)
  - python *"""docstring"""*

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

Köszönöm a figyelmet!