Objektumorientált programozás

Osztályok és objektumok

Goldschmidt Balázs

balage@iit.bme.hu

Komplex számokat akarunk!

- Valós számok
  - van rá típus! (float, double)
- Komplex számok
  - *pl: 3.0 + 4.0i*
    - $i = \sqrt{-1}$
  - nincs rá támogatás ☹
- Komplex szám szerkezete
  - $x + yi$, ahol x a valós és y a képzetes rész

Feladat

- Írjunk programot, ami beolvas 10 komplex számot, és kiírja a legnagyobb abszolút értékűt!

- Mire van szükségünk?
  - komplex szám tárolása
  - beolvasás
  - abszolút érték számítása
  - maximumkiválasztás
  - kiíratás

Komplex szám hibás tárolása

- Ötlet
  - egy komplex szám két valósból áll
  - tároljuk két valós tömbben!

```
double re[], im[];
re = new double[10];
im = new double[10];
```

- Problémák
  - szétcsúszhat a két tömb
    - csak az indexek kötik össze az összetartozó értékeket

Komplex szám helyes tárolása

- Használjunk rekordot, struktúrát

```
# python (complex.py)
class Complex:
  def __init__(self, re, im): # konstruktor
    self.re = re
    self.im = im
```

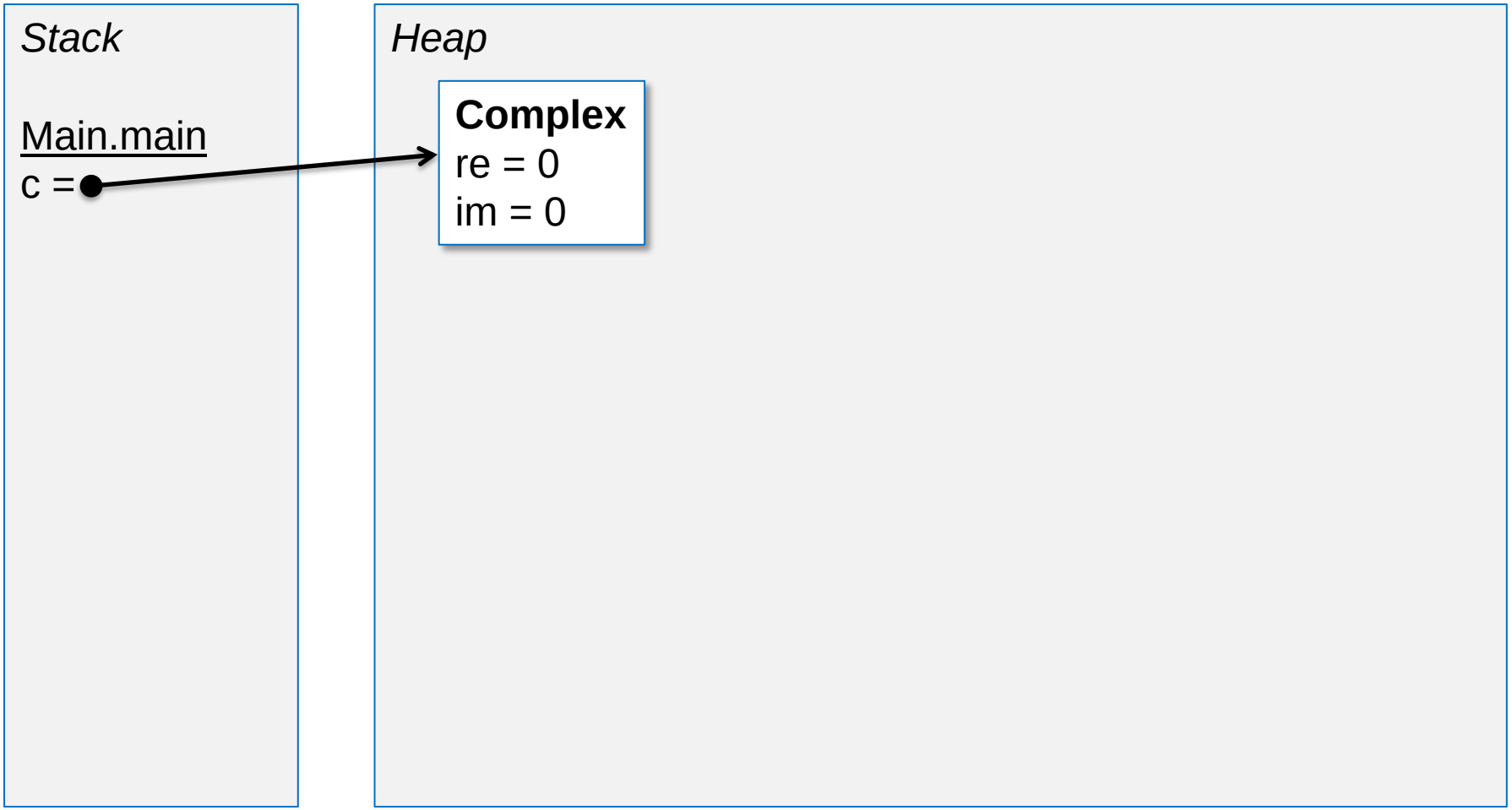
```
// Java (Complex.java)
public class Complex {
    public double re, im;
}
```

tagváltozó v. attribútum

```
// Java (main metódusban)
Complex c = new Complex();
```

objektum létrehozása

Komplex a memóriában



Complex beolvasása

```
// Java (Main.java)
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Complex c[] = new Complex[10];
        for (int i = 0; i < 10; i++) {
            c[i] = new Complex();
            c[i].re = sc.nextDouble();
            c[i].im = sc.nextDouble();
        }

        //...

    }
}
```

Belépési pont

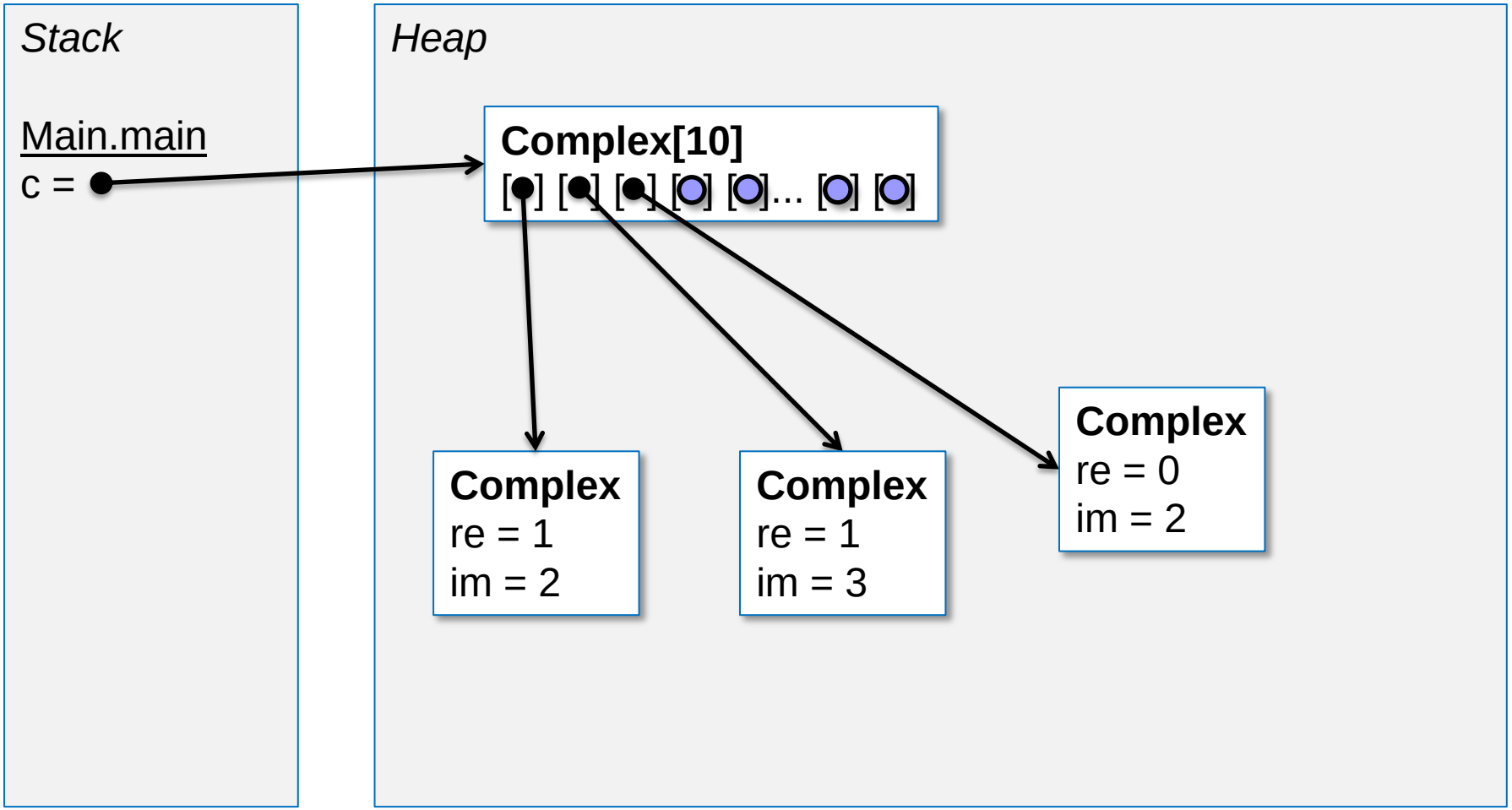
Std inputról olvasunk

Tömb létrehozása

Complex létrehozása

Érték beolvasása

Komplex a memóriában (tömb)



Abszolút érték számítása

- Python: függvény

```
# python (hello.py)
def abs(c):
  # ...
```

- Java: metódus Main-ben

```
// Java (Main.java)
public class Main {
    public static double abs(Complex c) { ... }
    public static void main(String[] args) { ... }
}
```

Abszolút érték számítása

- Python: függvény

```
# python (hello.py)
def abs(c):
  return math.sqrt(c.re*c.re+c.im*c.im)
```

- Java: metódus a Main-ben

```
...
    // abs(c) = √(c.re²+c.im²)
    public static double abs(Complex c) {
        return Math.sqrt(c.re*c.re + c.im*c.im);
    }
...
```

Maximumkiválasztás és kiírás

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Complex c[] = new Complex[10];
    for (int i = 0; i < 10; i++) {
        c[i] = new Complex();
        c[i].re = sc.nextDouble();
        c[i].im = sc.nextDouble();
    }
    int max = 0;
    for (int i = 1; i < 10; i++) {
        if (abs(c[max]) < abs(c[i])) {
            max = i;
        }
    }
    System.out.println(c[max].re+"+"+c[max].im+"i");
}
```

Adatrejtés

- Jó-e, hogy a *re* és *im* kívülről látszik?
  - válasz: nem ☺
  - miért?
- Mit lehet tenni, hogy ne látsszon?
  - válasz: priváttá kellene tenni
    - fordító ne engedje a hozzáférést (hibajelzés)

```
public class Complex {
  private double re, im;
}
```

- És akkor hogyan használjuk a *Complex*-et?
  - válasz: saját metódusai kell legyenek

Egységbezárás

- Az egységbezárás az OO alapvető elve
  - adatok és műveletek együtt definiálva
    - ez az osztály!
  - attribútumok csak belül látszanak (*private*)
    - az osztály metódusai férnek csak hozzá, adatrejtés
  - metódusok kívülről is (*public*)
    - lehetnek privát metódusok is (tkp. segédfüggvények)
- Attribútumok: *állapot*
- Metódusok: *viselkedés*

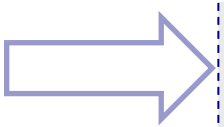
Egységbezárás hol sérül?

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Complex c[] = new Complex[10];
    for (int i = 0; i < 10; i++) {
        c[i] = new Complex();
        c[i].re = sc.nextDouble();
        c[i].im = sc.nextDouble();
    }
    int max = 0;
    for (int i = 1; i < 10; i++) {
        if (abs(c[max]) < abs(c[i])) { // Complex.abs(c[i])
            max = i;
        }
    }
    System.out.println(c[max].re+"+"+c[max].im+"i");
}
```

Complex metódusai

- Milyen viselkedést várunk a komplextől?
  - értéket lehessen adni

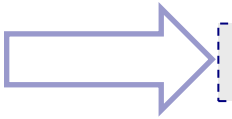
```
c[i].re =
  sc.nextDouble();
c[i].im =
  sc.nextDouble();
```



```
c[i].setRe(sc.nextDouble());
c[i].setIm(sc.nextDouble());
```

  - tudjon abszolútértéket számolni

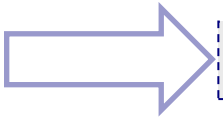
```
abs(c[max])
```



```
c[max].abs()
```

  - ki lehessen íratni (*String* konverzió)

```
(c[max].re+"+"
 +c[max].im+"i")
```

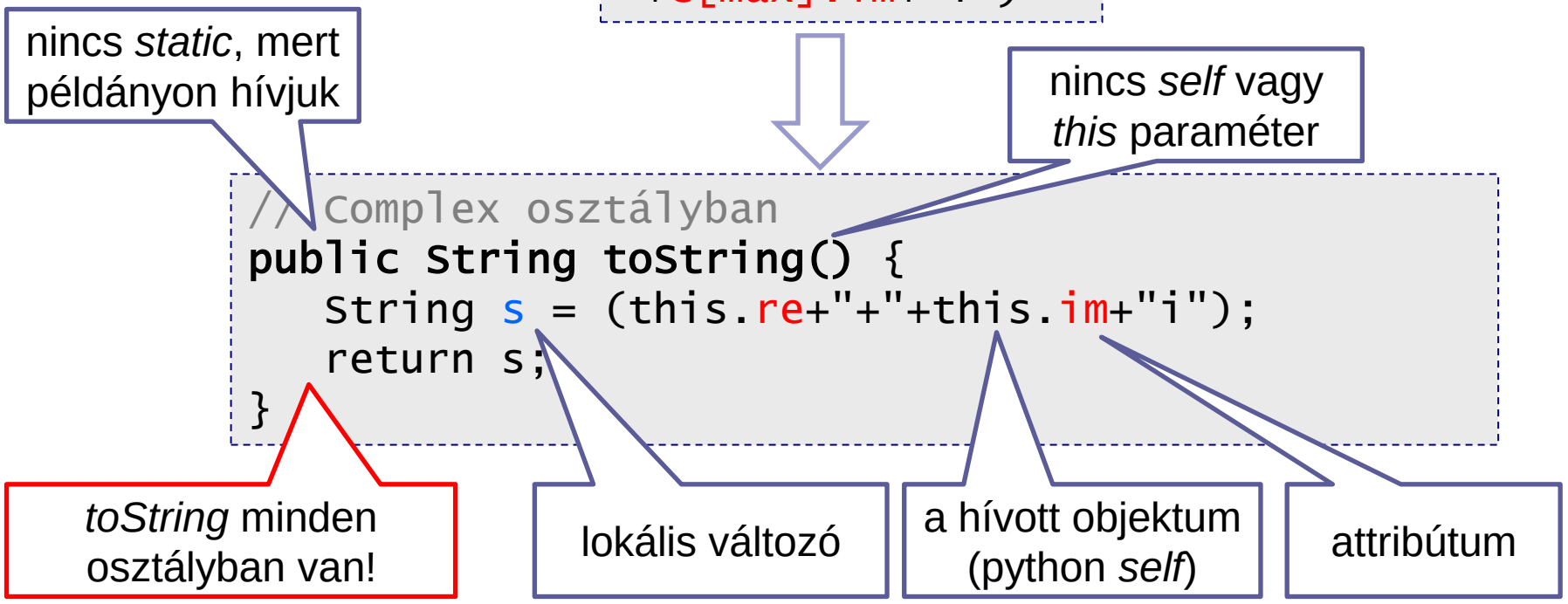


```
(c[max].toString())
```

Complex kiíratása

- Stringgé kell konvertálni, *String*-et visszaadó metódus kell

```
(c[max].re+"+"
 +c[max].im+"i")
```



Abszolút érték számítása

- Metódus kell, ami *double*-t ad vissza

```
Math.abs(c[max])
```

nincs *static*, mert példányon hívjuk

```
// Complex osztályban
public double abs() {
        double d = Math.sqrt(re*re+im*im);
        return d;
}
```

this elhagyható, ha triviális

Értékadás

- *Setter* metódusok

```
// Complex osztályban, külön metódusok
public void setRe(double re) {
    this.re = re;
}
public void setIm(double im) {
    this.im = im;
}
```

this kell a két *re* (*im*) megkülönböztetéséhez

Egységbezárás után

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Complex c[] = new Complex[10];
    for (int i = 0; i < 10; i++) {
        c[i] = new Complex();
        c[i].setRe(sc.nextDouble());
        c[i].setIm(sc.nextDouble());
    }
    int max = 0;
    for (int i = 1; i < 10; i++) {
        if (c[max].abs() < c[i].abs()){
            max = i;
        }
    }
    System.out.println(c[max].toString());
}
```

Attribútum, állapot lekérdezése

- *Getter* metódusok

```
public double getRe() {
    return re;
}
public double getIm() {
    return im;
}
```

  - abszolút érték is lehetne *getAbs()*, de hagyomány
  - az állapotról adnak információt
  - boolean értéknél *get* helyett *is*
    - *pl. isEmpty()*

Attribútumok állítása

- *Setter* metódusok ✓
- Inicializáló metódusok
  - python alapján: *init*

```
public void init(double re, double im) {
        this.re = re;
        this.im = im;
}
```

a hívott objektum referenciája (python *self*)

  - már létező objektumot módosít! ☹

Inicializálás létrehozáskor

- Konstruktor Javaban

```
public Complex(double re, double im) {
        this.re = re;
        this.im = im;
}
public Complex() { this(0,0); } // default ctr
```

delegálás egy másik konstruktornak

  - a neve mindig az osztály neve
    - pythonban *__init__*
  - nincs visszatérési értéke
  - egy objektumon mindig csak egyszer hívható
    - ha nincs, akkor alapértelmezve
    - minden attribútum gyárilag 0-ra inicializálva
  - *best practice*: paraméterlista sose legyen hosszú

Konstruktorral

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Complex c[] = new Complex[10];
    for (int i = 0; i < 10; i++) {
       c[i] =
          new Complex(sc.nextDouble(), sc.nextDouble());
    }
    int max = 0;
    for (int i = 1; i < 10; i++) {
        if (c[max].abs() < c[i].abs()){
            max = i;
        }
    }
    System.out.println(c[max].toString());
}
```

Újabb műveletek

- Implementáljunk további komplex műveleteket!
  - összeadás
  - kivonás
  - szorzás
  - stb.

Összeadás

- Mit szeretnénk?

```
Complex c1 = new Complex(1,0);
Complex c2 = new Complex(0,2);
Complex c3 = c1.add(c2); // c3 = c1+c2;
System.out.println(c3.toString());
```

- Kell egy metódus
  - név: *add*
  - paraméter: *másik Complex (c2)*
  - visszatérés: *az összeg (Complex)*

Összeadás tagfüggvénnyel

- Megvalósítása

```
public class Complex {
    private double re,im;
    //... korábbi metódusok

    public Complex add(Complex c) {
        Complex ret =
            new Complex(re+c.re, im+c.im);
        return ret;
    }

}
```

Új *Complex* objektum kell

Complex eléri más *Complex private* tagjait

Feladat

- Írjunk programot, ami beolvas 10 komplex számot, és kiírja az összegüket!

- Mire van szükségünk?
  - komplex szám tárolása: OK
  - beolvasás: OK
  - összeadás: OK
  - kiíratás: OK

Megoldás

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    Complex c = new Complex(); // aktuálisan beolvasott
    Complex sum = new Complex(0,0); // összeg inicializálva

    for (int i = 0; i < 10; i++) {
        c.setRe(sc.nextDouble());
        c.setIm(sc.nextDouble());
        sum = sum.add(c); // mindig nő a sum
    }

    System.out.println(sum.toString());
}
```

ehhez kell a default ctr

Egységbezárás: láthatóságok

- Privát
  - *private* előtag
  - csak az osztály látja (objektumok egymást is)
- Csomag
  - nincs előtag
  - azonos csomagban levők látják
- Publikus
  - *public* előtag
  - bárki látja

Láthatóság példa

```
package education;
public class Student {
  private String address;S
  double average;         ST
  public String name;     STC
  ...
}
```

```
package education;
public class Teacher {
  private String address; T
  String room;            TS
  public String name;     TSC
  ...
}
```

Látja:
T Theacher
S Student
C Cop

```
package police;
public class Cop {
  private String address; C
  int rank;               C
  public String name;     CTS
  ...
}
```

Metódusok típusai

- Konstruktor
  - objektum inicializálása
  - objektum születésekor
- Lekérdező (pl. *getter*, *behaviour*)
  - lekérdezi az állapotot
  - nem módosít
- Módosító (pl. *setter*, *modifier*)
  - változtat(hat) az állapoton

Egy népszerű osztály: *String*

- Létezik literálja
  - `"hello world"`
- Van **+** operátora
  - `"hello" + " world"`
- Nem módosul (*immutable*)
  - a kezdőértéke később nem változtatható
- Bármiből készíthető
  - `toString()`

Egy népszerű osztály: *String*

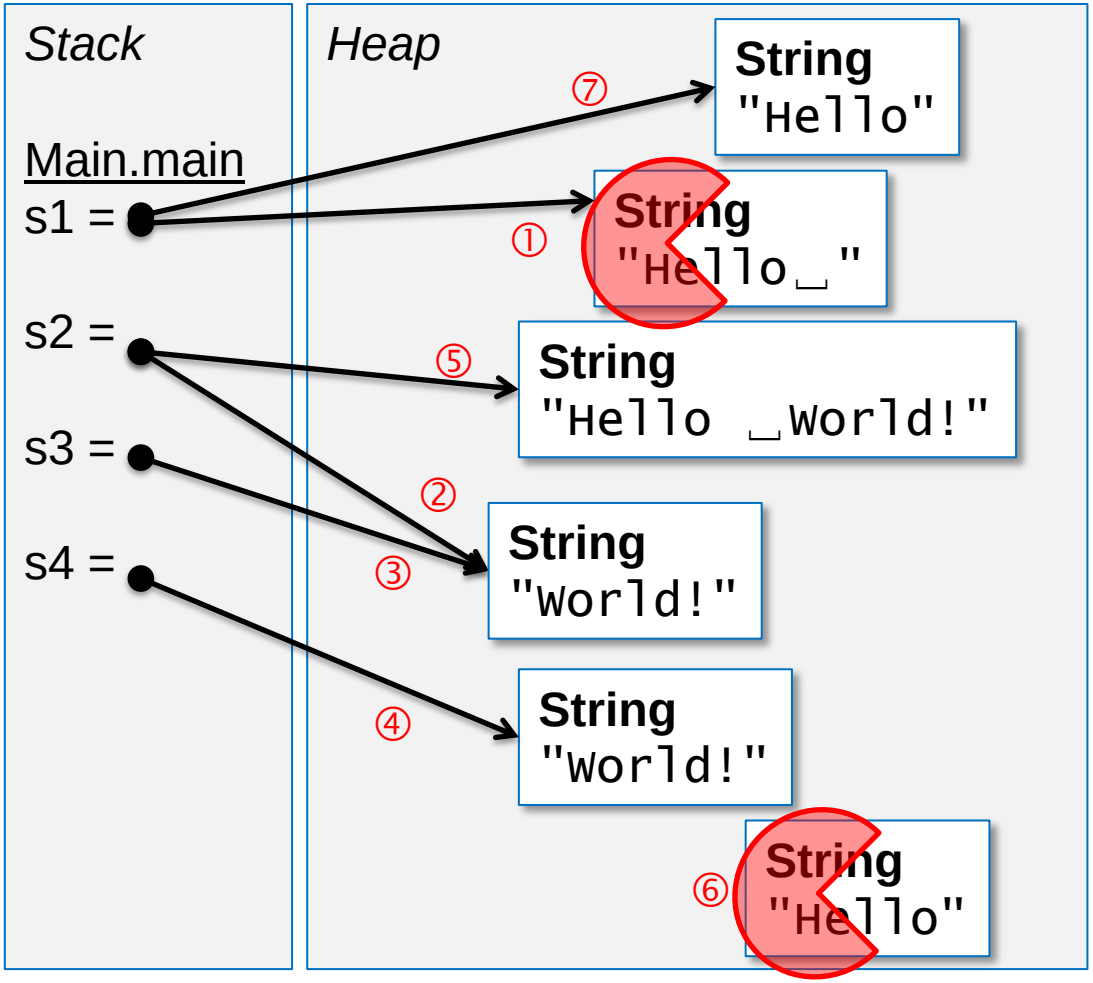
- Szokásos string-műveletek elérhetők
  - `length()`, `equals()`, `startsWith()`
  - `substring()`, `trim()`, `split()`, `concat()`
  - `toUpperCase()`, `toLowerCase()`, `replace()`
  - `charAt()`, `indexOf()`, `lastIndexOf()`
  - `valueOf()`
  - ...
- Immutable!
  - pl. *String trim()* új Stringet hoz létre, a régi nem módosul

String példák

```
String s1 = "Hello␣"; // s1 literálra referál
String s2 = new String("World!"); // új string
String s3 = s2;

s3 == s2 ; // true
s3.equals(s2); // true

String s4 = new String("World!"); // új string!
s3 == s4 ; // false
s3.equals(s4); // true

s2 = s1 + s2; // mi s3 tartalma?
// s1: "Hello ", s2: "Hello World!", s3: "World!"

s1.trim(); // s1: "Hello␣"
s1 = s1.trim(); // s1: "Hello"
```

Stringek a memóriában

```
①String s1 = "Hello␣";
②String s2 =
  new String("World!");
③String s3 = s2;
 s3 == s2 ; // true
 s3.equals(s2); // true
④String s4 =
  new String("World!");
 s3 == s4 ; // false
 s3.equals(s4); // true
⑤s2 = s1 + s2;
⑥s1.trim();
⑦s1 = s1.trim();
```



Ismétlés: tömbök is objektumok!

```
int[] a = new int[10]; // 1
int[] b;               // 2
b = a;                 // 3
```

- Példányosítani kell (*new*, 1. sor)
- Referenciával használjuk
  - *b* változó nem mutat sehova, inicializálatlan (2. sor)
  - *b* változó a-ra mutat, *a* és *b* "mögött" ugyanaz az tömb van (3. sor)
- Van attribútuma
  - *length*: megadja a tömb hosszát
  - tömb hossza sosem változhat