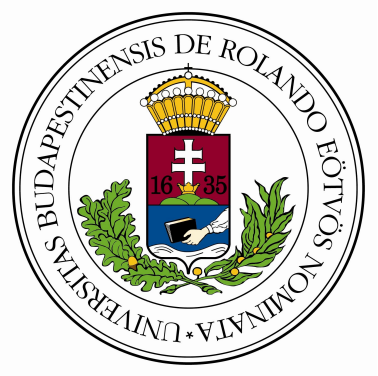
Programozási nyelvek – Java

Második előadás



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Absztrakció

- Egységbe zárás
- Információelrejtés

Egységbe zárás

```
class Time {
    int hour;
    int minute;
    Time( int hour, int minute ){
        this.hour = hour;
        this.minute = minute;
    }
    void aMinutePassed(){
        if( minute < 59 ){
            ++minute;
        } else { ... }
    }
}
```

```
Time morning = new Time(6,5);
morning.aMinutePassed();
int hour = morning.hour;
```

Típusinvariáns

```
class Time {
    int hour;                                   // 0 <= hour < 24
    int minute;                                 // 0 <= minute < 60
    Time( int hour, int minute ){
        this.hour = hour;
        this.minute = minute;
    }
    void aMinutePassed(){
        if( minute < 59 ){
            ++minute;
        } else { ... }
    }
}
```

Értelmetlen érték létrehozása

```
class Time {
    int hour;
    int minute;
    Time( int hour, int minute ){
        this.hour = hour;
        this.minute = minute;
    }
    void aMinutePassed(){
        if( minute < 59 ){
            ++minute;
        } else { ... }
    }
}
```

```
Time morning = new Time(6,5);
morning.aMinutePassed();
int hour = morning.hour;

morning.hour = -1;
morning = new Time(24,-1);
```

Létrehozásnál típusinvariáns biztosítása

```
class Time {
    int hour;                                   // 0 <= hour < 24
    int minute;                                 // 0 <= minute < 60
    Time( int hour, int minute ){
        if (0 <= hour && hour < 24 && 0 <= minute && minute < 60){
            this.hour = hour;
            this.minute = minute;
        }
    }
    void aMinutePassed(){
        if( minute < 59 ){
            ++minute;
        } else { ... }
    }
}
```

Kerüljük el a „silent failure” jelenséget

```
class Time {
    int hour;                                   // 0 <= hour < 24
    int minute;                                 // 0 <= minute < 60
    Time( int hour, int minute ){
        if (0 <= hour && hour < 24 && 0 <= minute && minute < 60){
            this.hour = hour;
            this.minute = minute;
        } else {
            throw new IllegalArgumentException("Invalid time!");
        }
    }
    void aMinutePassed(){
        ...
    }
}
```

Kivétel

- Futás közben lép fel
- Problémát jelezhetünk vele
  - `throw` utasítás
- Jelezhet „dinamikus szemantikai hibát”
- Program leállását eredményezheti
- Lekezelhető a programban
  - `try–catch` utasítás

Futási hiba

```
class Main {
    public static void main( String[] args ){
        Time morning = new Time(24,-1);
    }
}
```

```
$ javac Time.java
$ javac Main.java
$ java Main
Exception in thread "main" java.lang.IllegalArgumentException:
Invalid time!
    at Time.<init>(Time.java:9)
    at Main.main(Time.java:16)
$
```

A mezők közvetlenül manipulálhatók

```
class Time {
    int hour;                          // 0 <= hour < 24
    int minute;                        // 0 <= minute < 60
    ...
}
```

```
class Main {
    public static void main( String[] args ){
        Time morning = new Time(6,5);
        morning.aMinutePassed();

        morning.hour = -1;             // ajjaj!
    }
}
```

Mező elrejtése: private

```
class Time {
    private int hour;                    // 0 <= hour < 24
    private int minute;                  // 0 <= minute < 60
    ...
}
```

```
class Main {
    public static void main( String[] args ){
        Time morning = new Time(6,5);
        morning.aMinutePassed();

        morning.hour = -1;               // fordítási hiba
    }
}
```

Idióma: privát állapot csak műveleteken keresztül

```
class Time {
    private int hour;                   // 0 <= hour < 24
    private int minute;                 // 0 <= minute < 60
    Time( int hour, int minute ){ ... }
    int getHour(){ return hour; }
    int getMinute(){ return minute; }
    void setHour( int hour ){
        if( 0 <= hour && hour <= 23 ){
            this.hour = hour;
        } else {
            throw new IllegalArgumentException("Invalid hour!");
        }
    }
    void setMinute( int minute ){ ... }
    void oneMinutePassed(){ ... }
}
```

Getter-setter konvenció

Lekérdező és beállító művelet neve

```
class Time {
    private int hour;                    // 0 <= hour < 24
    int getHour(){ return hour; }
    void setHour( int hour ){
        if( 0 <= hour && hour <= 23 ){
            this.hour = hour;
        } else {
            throw new IllegalArgumentException("Invalid hour!");
        }
    }
    ...
}
```

Reprezentáció változtatása

```
class Time {
    private short minutes;
    Time( int hour, int minute ){
        if (0 <= hour && hour < 24 && 0 <= minute && minute < 60){
            minutes = 60*hour + minute;
        } else {
            throw new IllegalArgumentException("Invalid time!");
        }
    }
    int getHour(){ return minutes / 60; }
    int getMinute(){ return minutes % 60; }
    void setHour( int hour ){
        if( 0 <= hour && hour <= 23 ){
            minutes = 60 * hour + getMinute();
        } else {
            throw new IllegalArgumentException("Invalid hour!");
```

Információ elrejtése

- Osztályhoz szűk interfész
  - Ez „látszik” más osztályokból
  - A lehető legkevesebb kapcsolat
- Priváttá tett implementációs részletek
  - Segédműveletek
  - Mezők

Előnyök

- Típusinvariáns megőrzése könnyebb
- Kód könnyebb evolúciója (reprezentációváltás)
- Kevesebb kapcsolat, kisebb komplexitás

Outline

Csomag

- Program tagolása
- Összetartozó osztályok összefogása
- Programkönyvtárak
  - Szabványos programkönyvtár

A package utasítás

```
package geometry;

class Point {
    int x, y;
    void move( int dx, int dy ){
        x += dx;
        y += dy;
    }
}
```

- Osztály (teljes) neve: `geometry.Point`
- Osztály rövid neve: `Point`

Hierarchikus névtér

```
package geometry.basics;

class Point {    // geometry.basics.Point
    int x, y;
    void move( int dx, int dy ){
        x += dx;
        y += dy;
    }
}
```

- Szabványos programkönyvtár, pl. `java.net.ServerSocket`
- `hu.elte.kto.teaching.javabsc.geometry.basics.Point`

Fordítás, futtatás

- Munkakönyvtár (working directory)
- Hierarchikus csomagszerkezet $\rightarrow$ könyvtárszerkezet
- Fordítás a munkakönyvtárból
  - Fájlnév teljes elérési úttal
- Futtatás a munkakönyvtárból
  - Teljes osztálynév

```
$ ls -R
.:
geometry

./geometry:
basics

./geometry/basics:
Main.java  Point.java
$ javac geometry/basics/*.java
$ ls geometry/basics
Main.class  Main.java
Point.class  Point.java
$ java geometry.basics.Main
$
```

Névtelen csomag

Default/anonymous package

- Ha nem írunk package utasítást
- Forrásfájl közvetlenül a munkakönyvtárba
- Kis kódbázis esetén rendben van

Láthatósági kategóriák

- `private` (privát, rejtett)
  - csak az osztálydefiníción belül
- semmi (félnyilvános, package-private)
  - csak az ugyanabban a csomagban lévő osztálydefiníciókban
- `public` (publikus, nyilvános)
  - osztály is
  - tagok, konstruktor is

Nyilvános és rejtett tagokat tartalmazó nyilvános osztály

```
package hu.elte.kto.javabsc.eloadas;

public class Time {
    private int hour;                   // 0 <= hour < 24
    private int minute;                 // 0 <= minute < 60
    public Time( int hour, int minute ){ ... }
    public int getHour(){ return hour; }
    public int getMinute(){ return minute; }
    public void setHour( int hour ){ ... }
    public void setMinute( int minute ){ ... }
    public void oneMinutePassed(){ ... }
}
```

Több csomagból álló program

hu/elte/kto/javabsc/eloadas/Time.java

```
package hu.elte.kto.javabsc.eloadas;

public class Time {
    ...
}
```

Main.java

```
// a névtelen csomagban

class Main {
    public static void main( String[] args ){
        hu.elte.kto.javabsc.eloadas.Time morning = new Time(6,5);
        ...
    }
}
```

Az import utasítás

hu/elte/kto/javabsc/eloadas/Time.java

```
package hu.elte.kto.javabsc.eloadas;

public class Time {
    ...
}
```

Main.java

```
import hu.elte.kto.javabsc.eloadas.Time;

class Main {
    public static void main( String[] args ){
        Time morning = new Time(6,5);
        ...
    }
}
```

Minősített név feloldása

- Osztály teljes neve helyett a rövid neve
- `import hu.elte.kto.javabsc.*;`
- Nem tranzitív
- A `java.lang` csomag típusait nem kell
- Névütközés: teljes név kell
  - java.util.List
  - java.awt.List

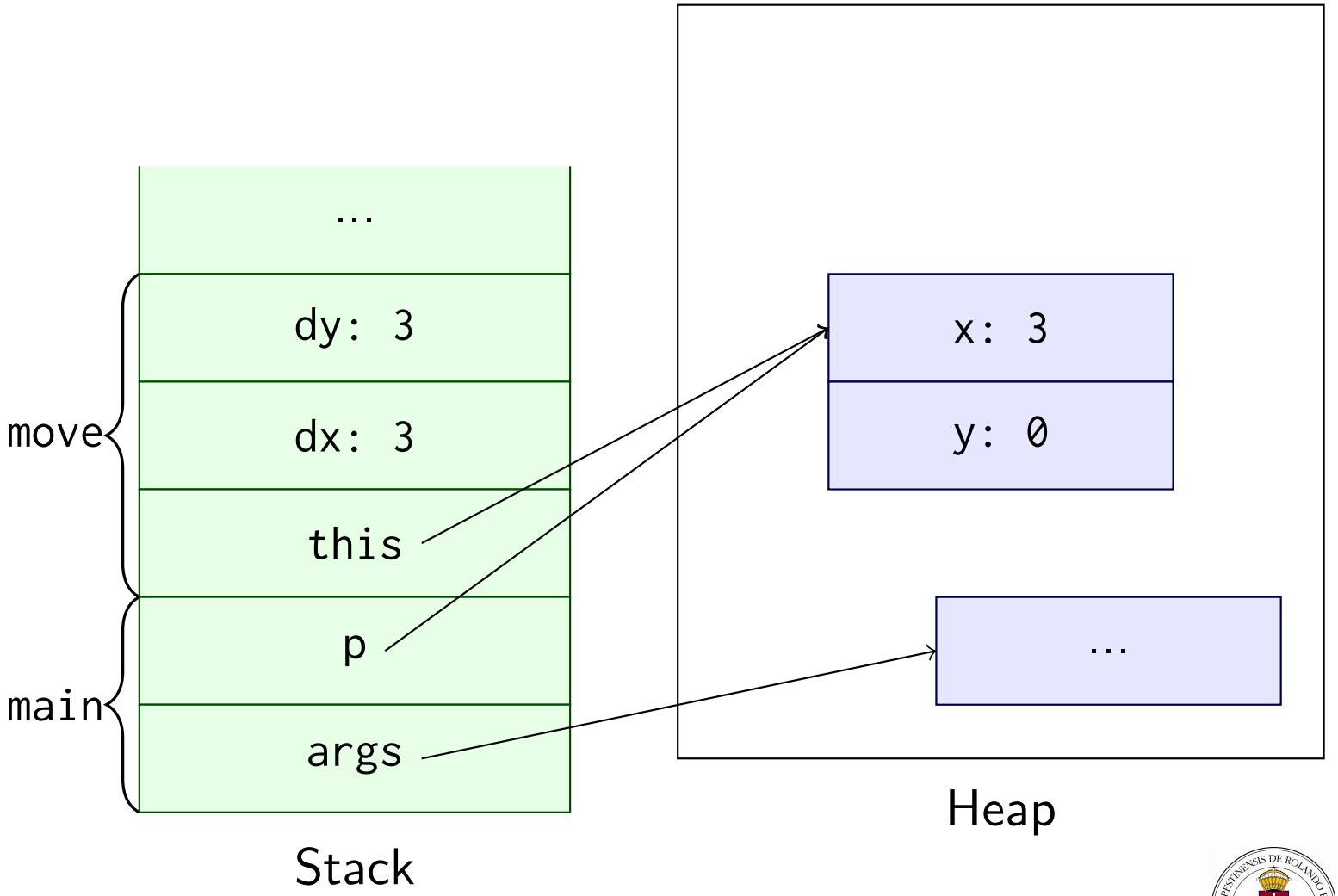
Outline

Referencia

- Osztály típusú változó
- Objektumra hivatkozik
- Heap
- Létrehozás: `new`
- Dereferálás: `.`

```
Point p;
p = new Point();
p.x = 3;
```

Különböző típusú változók a memóriában



Típusok

Primitív típusok

- byte: $[-128..127]$
- short: $[-2^{15}..2^{15}-1]$
- int: $[-2^{31}..2^{31}-1]$
- long: 8 bájt
- float: 4 bájt
- double: 8 bájt
- char: 2 bájt
- boolean: {false,true}

Referenciák

- Osztályok
- Tömb típusok
- ...

Ábrázolás a memóriában

Végrehajtási verem

Lokális változók és paraméterek
(Primitív típusú, referencia)

Heap

Objektumok, mezőik
(Primitív típusú, referencia)

Lokális változók hatóköre és élettartama

- Más nyelvekhez (pl. C) hasonló szabályok
- Lokális változó élettartama: hatókör végéig
- Hatókör: deklarációtól a közvetlenül tartalmazó blokk végéig
- Elfedés: csak mezőt

```
class Point {
    int x = 0, y = 0;
    void foo( int x ){    // OK
        int y = 3;        // OK
        {
            int z = y;
            int y = x;    // Fordítási hiba
            ...
        }
    }
}
```

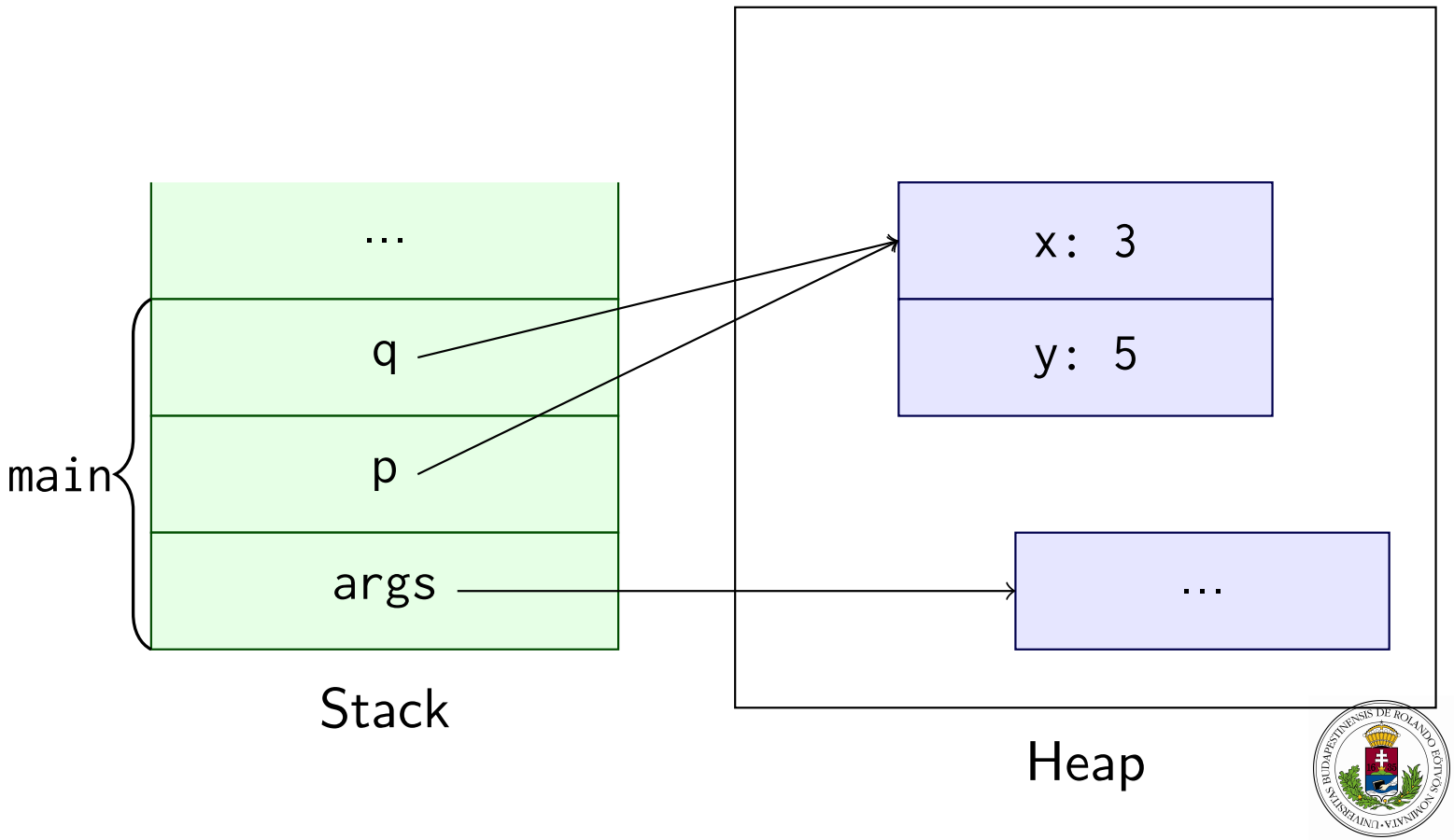
Objektumok élettartama

- Létrehozás + inicializálás
- Referenciák ráállítása
  - Aliasing
- Szemétgyűjtés

```
new Point(3,5)
Point p = new Point(3,5);
Point q = p;
p = q = null;
```

Aliasing

```
Point p = new Point(3,5), q = p;
q.x = 6;
```



Üres referencia

```
Point p = null;
p = new Point(4,6);
if( p != null ){
    p = null;
}
p.x = 3;    // NullPointerException
```

Mezők inicializálása

Automatikusan, nulla-szerű értékre

```
class Point {
    int x = 0, y = 0;
}
```

```
class Point {
    int x, y;
}
```

```
class Point {
    int x, y = 0;
}
```

```
class Point {
    int x, y = x;
}
```

Inicializálás üres referenciára

```
class Hero {
    String name;          // == null
    Hero bestFriend;      // == null
}
```

```
Hero ironMan = new Hero();
ironMan.name = "Iron Man";
// ironMan.bestFriend == null
```

Lokális változók inicializálása

- Nincs automatikus inicializáció
- Explicit értékadás kell olvasás előtt
- Fordítási hiba (statikus szemantikai hiba)

```
public static void main( String[] args ){
    int i;
    Point p;
    p.x = i;    // duplán fordítási hiba
}
```

Lokális változóra garantáltan legyen értékadás, mielőtt az értékét használni próbálnánk.

Garantáltan értéket kapni

- „Minden” végrehajtási úton kapjon értéket
- Túlbiztosított szabály (ellenőrizhetőség)

Példa a JLS-ből (16. fejezet, Definite Assignment)

```
{
    int k;
    int n = 5;
    if (n > 2)
        k = 3;
    System.out.println(k);  /* k is not "definitely assigned"
                               before this statement */
}
```

Szemétgyűjtés

Feleslegessé vált objektumok felszabadítása

Helyes

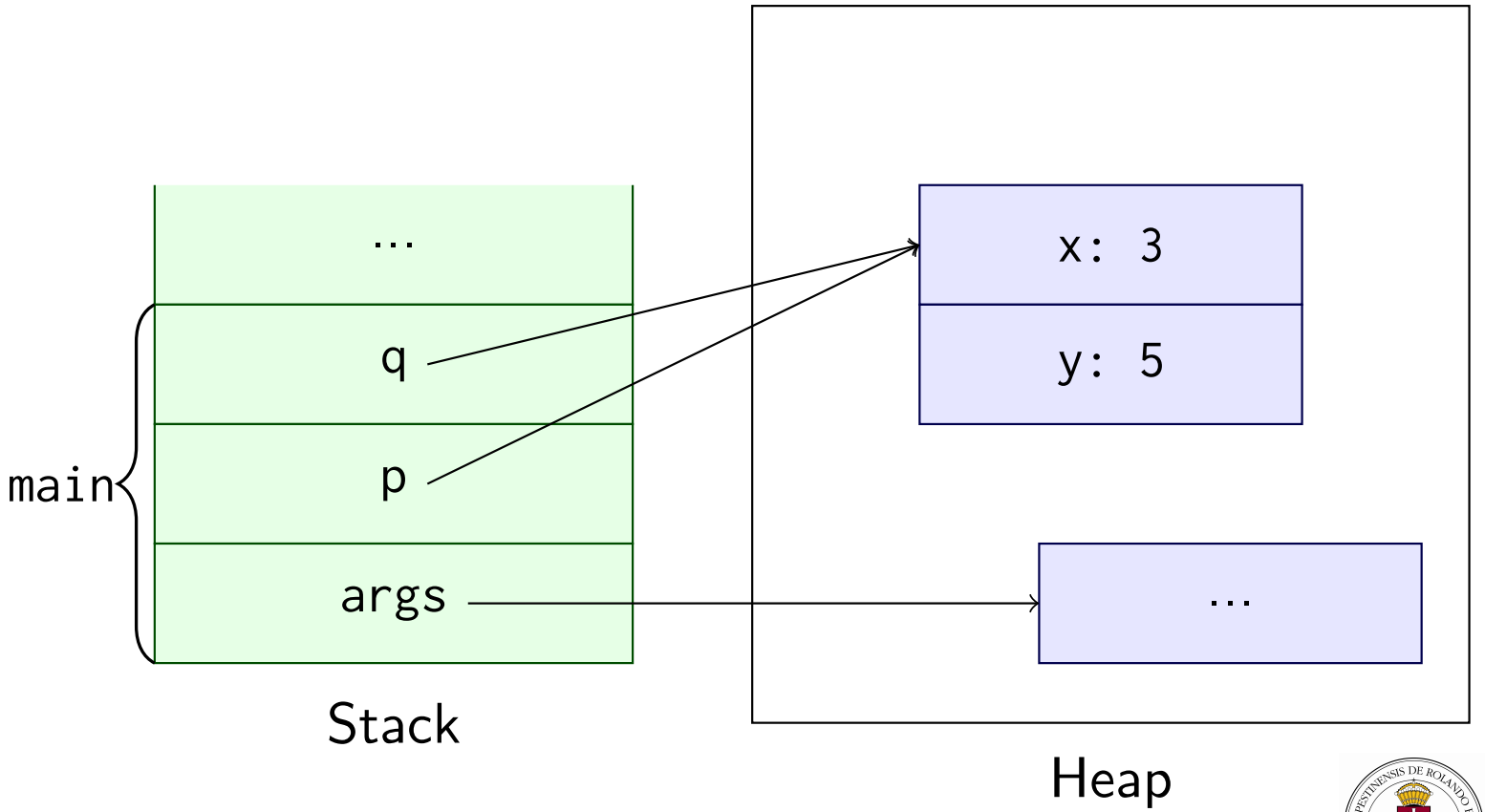
Csak olyat szabadít fel, amit már nem lehet elérni a programból

Teljes

Mindent felszabadít, amit nem lehet már elérni

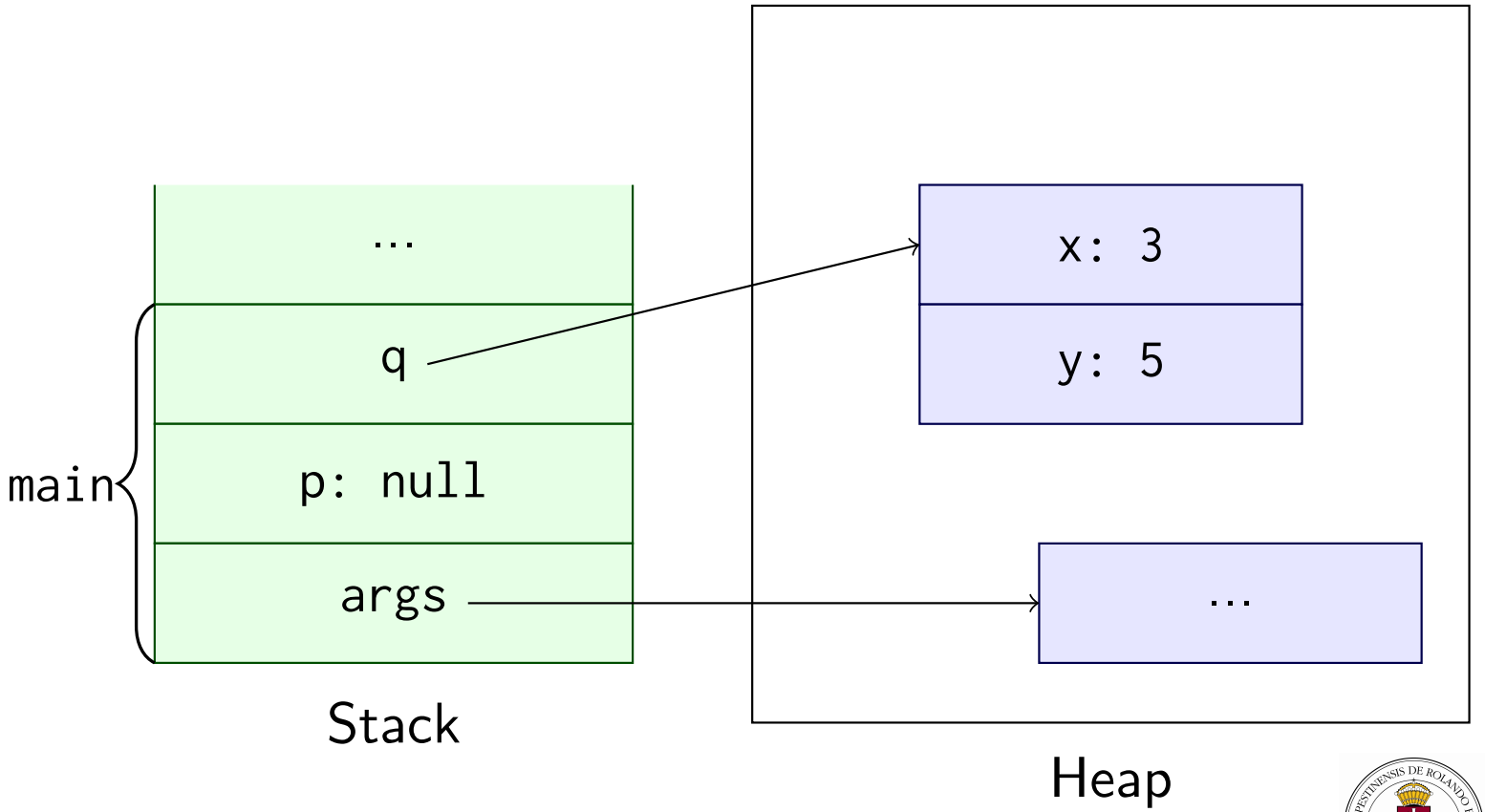
Még nem szabadítható fel

```
Point p = new Point(3,5), q = p;
```



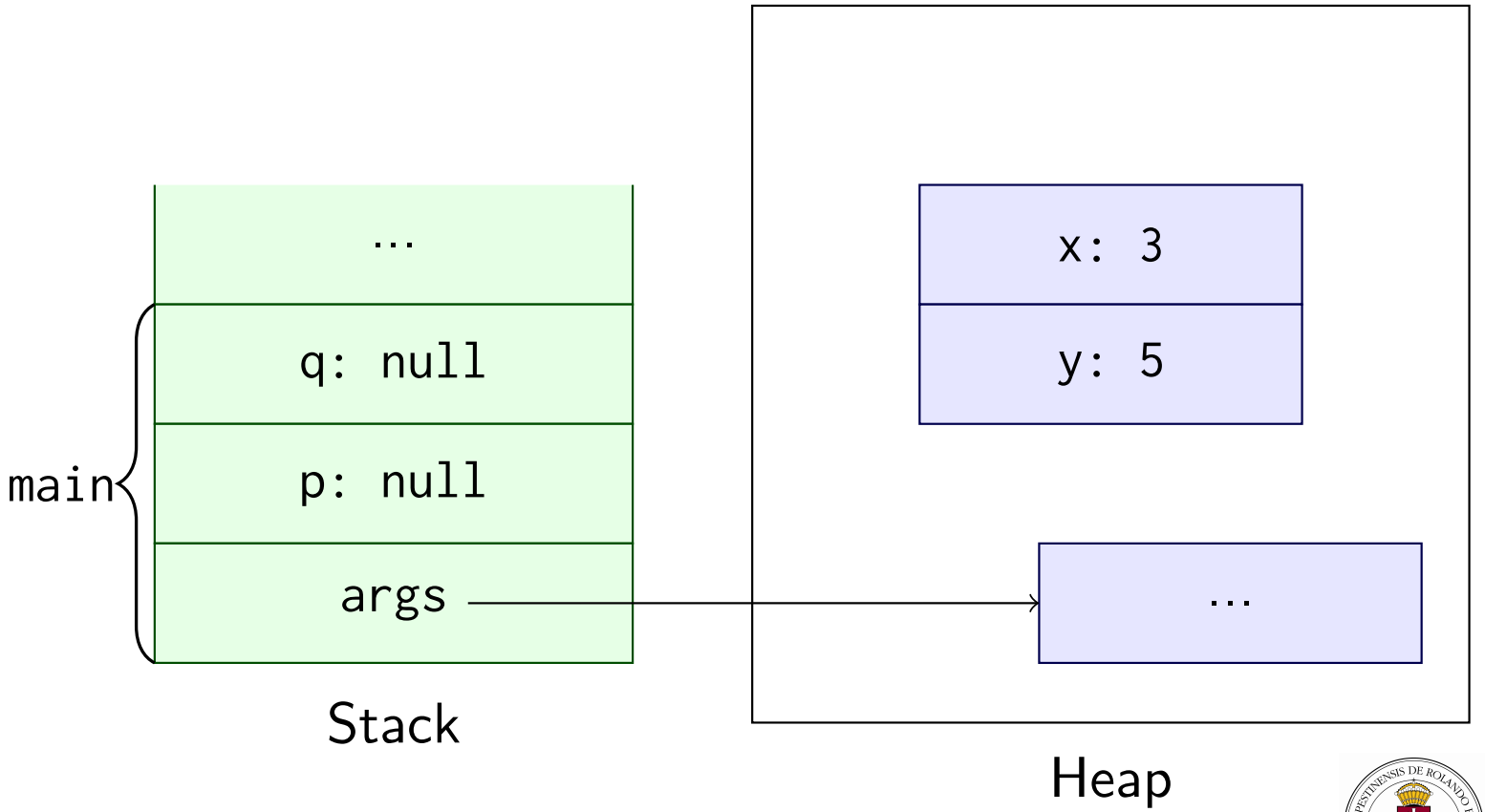
Még mindig nem szabadítható fel

```
p = null;
```



Már felszabadítható

```
q = null;
```

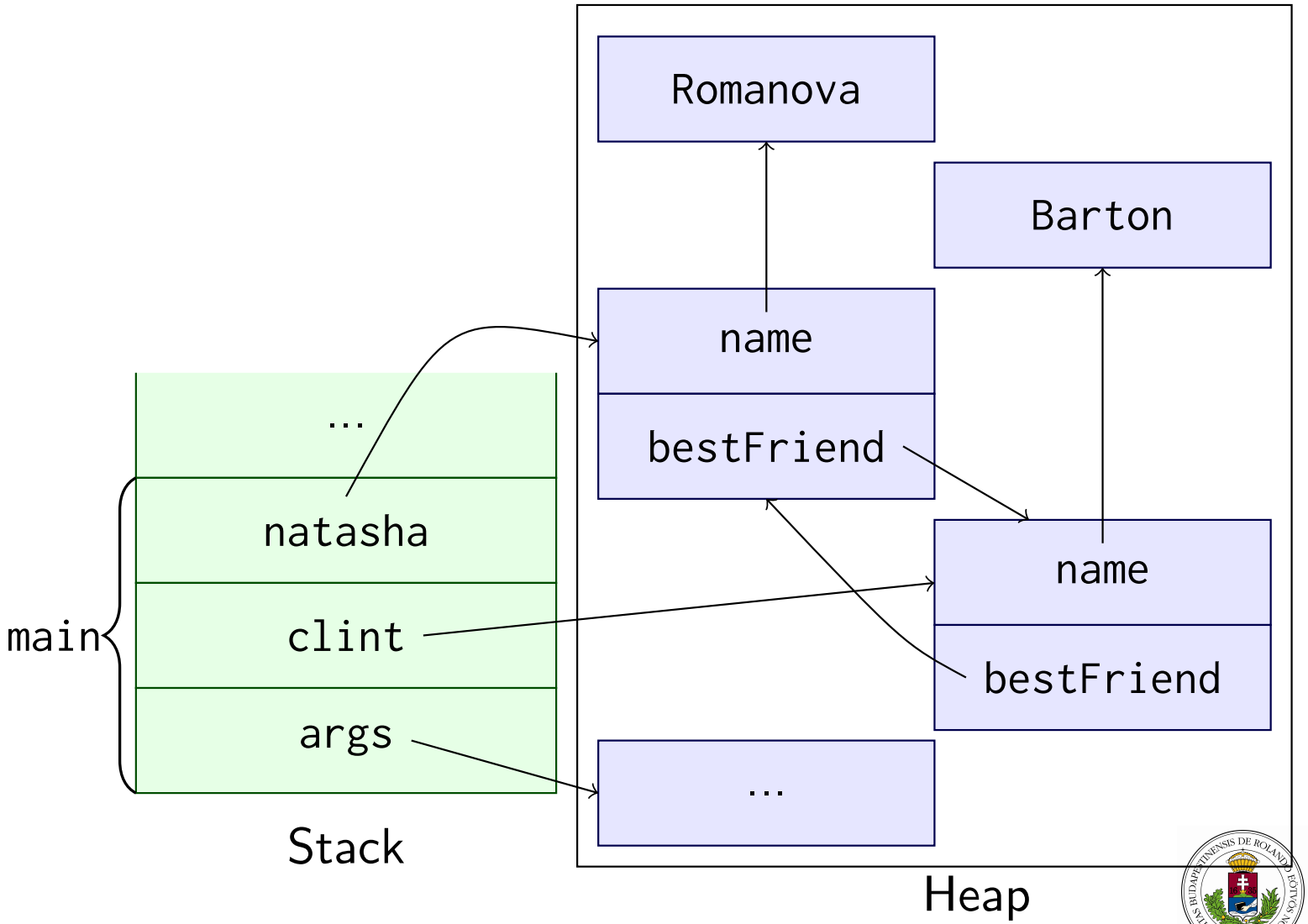


Bonyolultabb példa

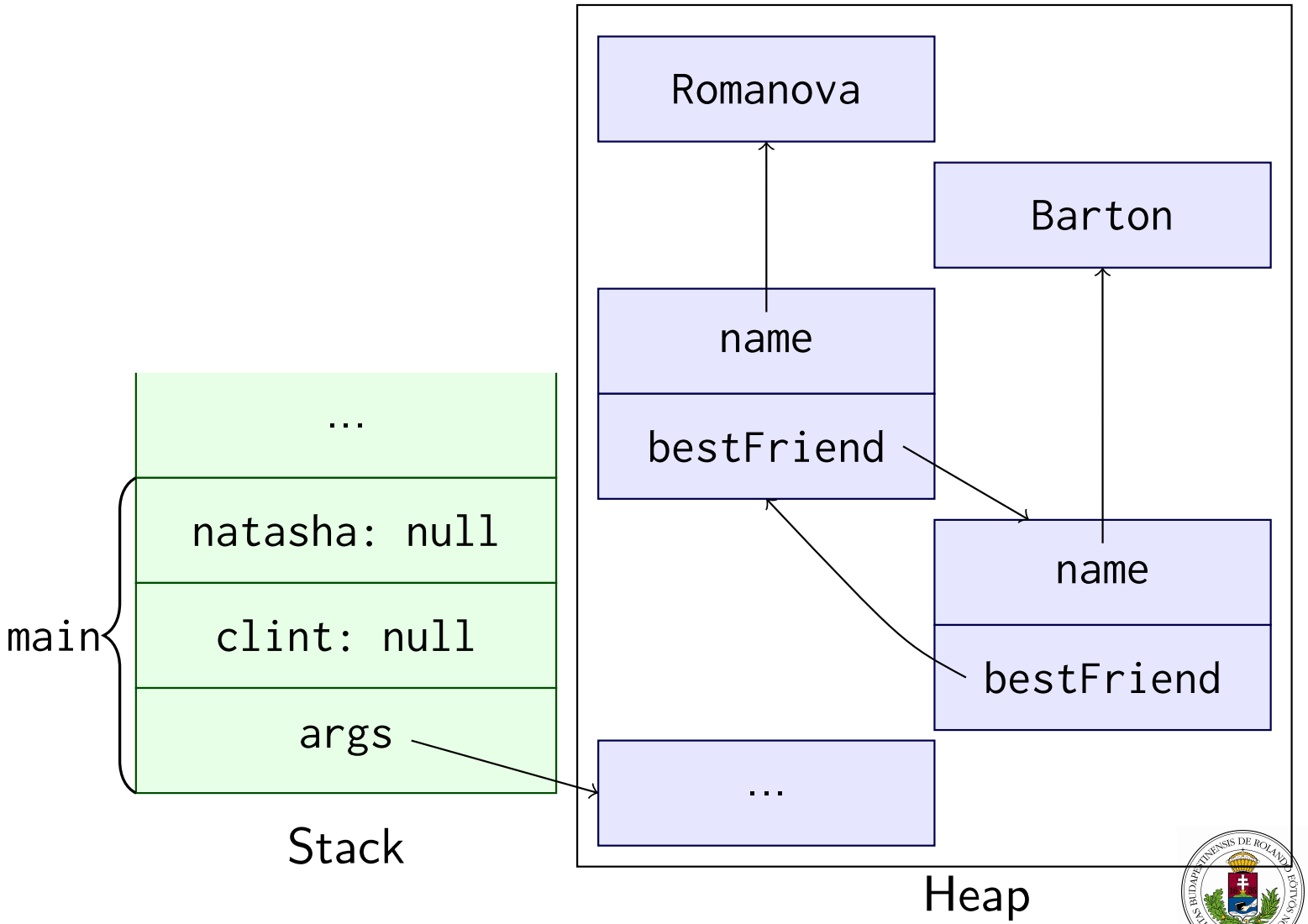
```
class Hero {
    String name;
    Hero bestFriend;
}
```

```
Hero clint = new Hero(),
     natasha = new Hero();
clint.name = "Barton";
natasha.name = "Romanova";
clint.bestFriend = natasha;
natasha.bestFriend = clint;
```

Hősök a memóriában



natasha = clint = null;



Mark-and-Sweep garbage collection

- Kiindulunk a vermen lévő referenciákból
- Megjelöljük a belőlük elérhető objektumokat
- Megjelöljük a megjelöltekből elérhető objektumokat
- Amíg tudunk újabbat megjelölni (tranzitív lezárt)
- A jelöletlen objektumok felszabadíthatók

Statikus mezők

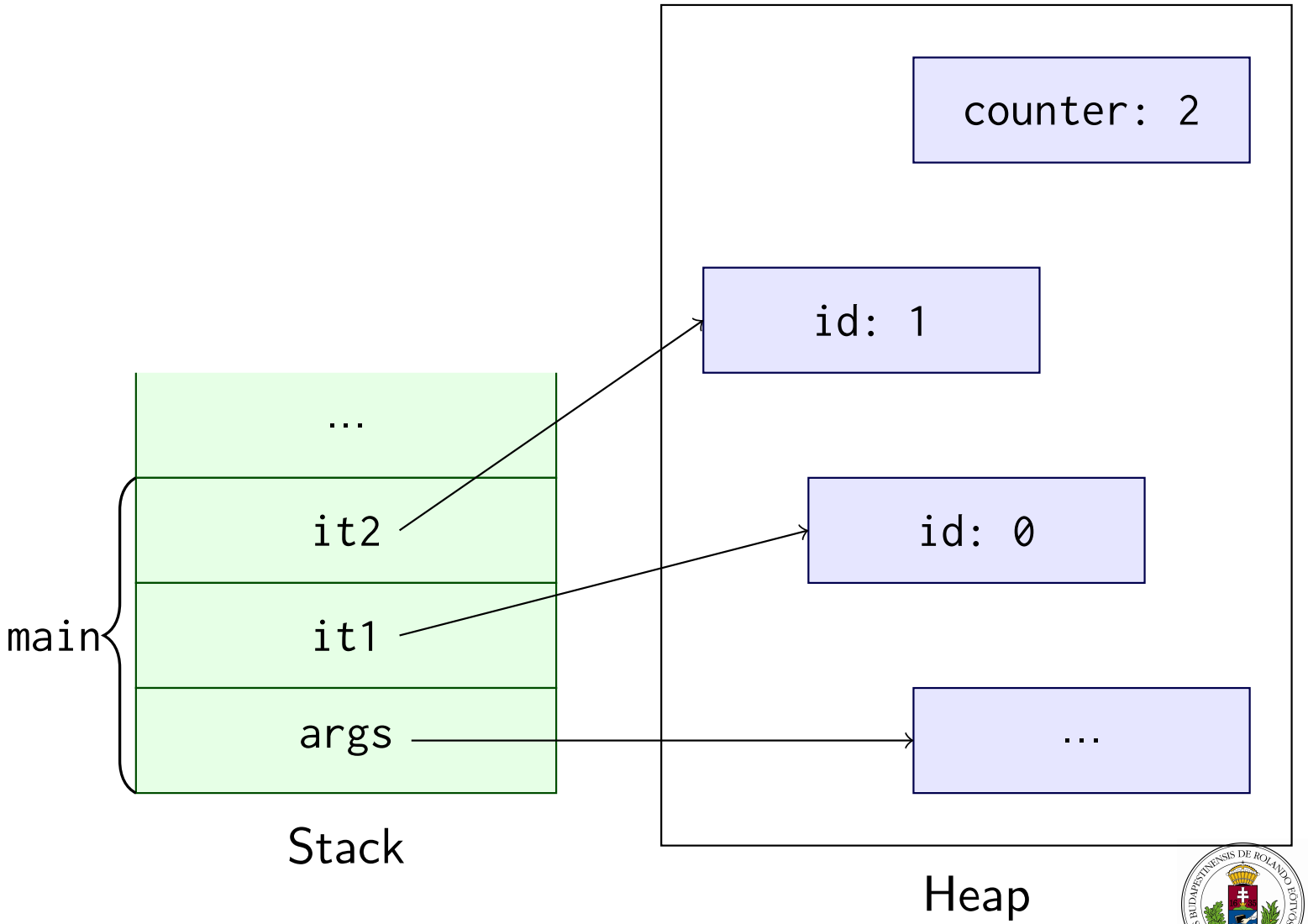
- Hasonló a C globális változóihoz
- Csak egy létezik belőle
- Az osztályon keresztül érhető el
- Mintha *statikus tárhelyen* lenne, nem az objektumokban

```
class Item {
    static int counter = 0;
}

class Main {
    public static void main( String[] args ){
        System.out.println( Item.counter );
    }
}
```

Osztályszintű és példányszintű mezők

```
class Item {
    static int counter = 0;
    int id = counter++;         // jelentése: id = Item.counter++
}

class Main {
    public static void main( String[] args ){
        Item it1 = new Item(), it2 = new Item();
        System.out.println( it1.id );
        System.out.println( it2.id );
        System.out.println( it1.counter );  // csúf, jelentése:
                                            // Item.counter
    }
}
```

`Item it1 = new Item(), it2 = new Item();`



Statikus metódusok

- Hasonló a C globális függvényeihez
- Az osztályon keresztül hívható meg
- Nem kell hozzá objektumot létrehozni
- A statikus mezők logikai párja

```
class Item {
    static int counter = 0;
    static void print(){
        System.out.println( counter );
    }
}
class Main {
    public static void main( String[] args ){
        Item.print();
    }
}
```

Statikus metódusban nincsen this

```
class Item {
    static int counter = 0;
    int id = counter++;
    static void print(){
        System.out.println( counter );
        System.out.println( id );       // értelmetlen
    }
}

class Main {
    public static void main( String[] args ){
        Item.print();
    }
}
```

Tömb

- Adatszerkezet
- Tömbelemek egymás után a memóriában
- Indexelés: hatékony
- Javában is 0-tól indexelünk, `[]`-lel

Tömb típusok

`String[] args`

- Az `args` egy referencia
- A tömbök objektumok
  - A heapen tárolódnak
  - Létrehozás: `new`
- A tömbök tárolják a saját méretüket
  - `args.length`
  - Futás közbeni ellenőrzés
  - `ArrayIndexOutOfBoundsException`

Tömbök bejárása

```
public static void main( String[] args ){
    for( int i = 0; i < args.length; ++i ){
        System.out.println( args[i] );
    }
}
```

ArrayIndexOutOfBoundsException

```
public static void main( String[] args ){
    for( int i = 0; i <= args.length; ++i ){
        System.out.println( args[i] );
    }
}
```

Iteráló ciklus (enhanced for-loop)

```
public static void main( String[] args ){
    for( int i = 0; i < args.length; ++i ){
        System.out.println( args[i] );
    }
}
```

```
public static void main( String[] args ){
    for( String s: args ){
        System.out.println( s );
    }
}
```

Tömbök létrehozása és feltöltése

```
public static void main( String[] args ){
    int[] numbers = new int[args.length];   // 0-kkal feltöltve
    for( int i = 0; i < args.length; ++i ){
        numbers[i] = Integer.parseInt( args[i] );
    }
    java.util.Arrays.sort(numbers);
}
```

Outline

Forráskód felépítése

- fordítási egységek
- típusdefiníciók
- metódusok
- utasítások
- kifejezések
- lexikális elemek
- karakterek

Lexikális elemek

- Kulcsszavak (`while`, `case`, `class`, `new` stb.)
- Azonosítók (pl. `Point`, `move`)
- Operátorok (<=, =, <<< stb.)
- Literálok (pl. `6.022140857E23`, `"hello"`, `'\n'`)
- Zárójelek, speciális jelek
- Megjegyzések (egysoros, többsoros, „dokumentációs”)

Kifejezések

- szintaxis: operátorok arítása, fixitása
- kiértékelés
  - precedencia (`A + B * C`)
  - asszociativitás
  - operandusok kiértékelési sorrendje
  - lustaság
  - mellékhatás

Utasítások

- Értékadások
- Metódushívás
- `return`-utasítás
- Elágazások (`if`, `switch`)
- Ciklusok (`while`, `do–while`, `for`)
- Blokk-utasítás
- Változódeklaráció
- Kivételkezelő és -kiváltó utasítások
- `assert`-utasítás

Típusdefiníciók

- **Osztály** (`class`)
- Interfész (`interface`)
- Felsorolási típus (`enum`)
- Annotáció típus (`@interface`)

Fordítási egység

compilation unit

- opcionális package utasítás
- opcionális `import` utasítások
- típusdefiníciók