Programozási nyelvek Java

Adatábrázolás

Kozsik Tamás

ELTE
IK

Objektumelvű programozás

Object-oriented programming (OOP)

- Objektum
- Osztály
- Absztrakció
  - ◇ Egységbe zárás (enkapszuláció)
  - ◇ Információ elrejtése
- Öröklődés
- Altípusosság, altípusos polimorfizmus
- Felüldefiniálás, dinamikus kötés

ELTE
IK

Egységbe zárás: objektum

Adat és rajta értelmezett alapműveletek (v.ö. C-beli `struct`)

- "Pont" objektum
- "Racionális szám" objektum
- "Sorozat" objektum
- "Ügyfél" objektum

```
p.x = 0;
p.y = 0;
p.move(3,5);
System.out.println(p.x);
```

ELTE
IK

Metódus

```
// Java kód
p.x = 0;
p.y = 0;
p.move(3, 5);

// megfelelője objektumok nélküli nyelvekben (pl. C)
p.x = 0;
p.y = 0;
move(p, 3, 5);
```

ELTE
IK

Osztály

Objektumok típusa

- “Pont” osztály
- “Racionális szám” osztály
- “Sorozat” osztály
- “Ügyfél” osztály

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

ELTE
IK

Példányosítás (instantiation)

- Objektum létrehozása osztály alapján
- Javában: mindig a heapen

```
Point p = new Point();
```

ELTE
IK

Példa: szövegek

```
String   name   = "James Arthur Gosling";
String[] names  = name.split(" ");
String   abbrev = names[names.length-1] + ", "
                  + names[0].charAt(0) + ".";
```

ELTE
IK

Osztály, objektum, példányosítás

Point.java

```
public class Point { // osztálydefiníció
  int x, y;          // mezők
}
```

Osztály, objektum, példányosítás

Point.java

```
public class Point { // osztálydefiníció
  int x, y;          // mezők
}
```

Main.java

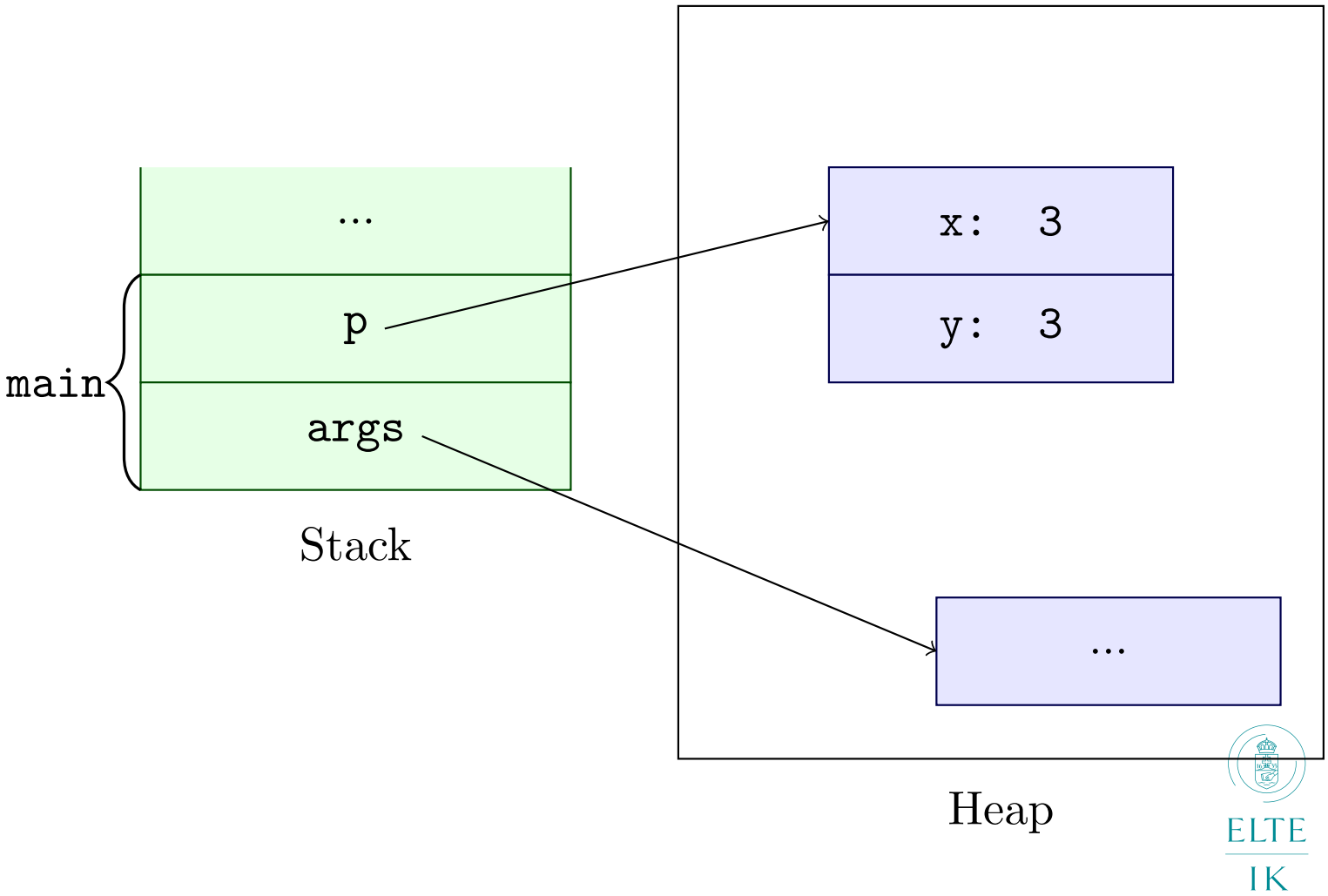
```
public class Main {
  public static void main(String[] args) {   // főprogram
    Point p = new Point(); // példányosítás (heap)
    p.x = 3;               // objektum állapotának
    p.y = 3;               // módosítása
  }
}
```

Fordítás, futtatás

```
$ ls
Main.java  Point.java
$ javac *.java
$ ls
Main.class  Main.java  Point.class  Point.java
$ java Point
Error: Main method not found in class Point, please define
 the main method as:
   public static void main(String[] args)
$ java Main
$
```

ELTE
IK

Stack és heap



Mezők inicializációja

```
class Point {
  int x = 3, y = 3;
}

class Main {
  public static void main(String[] args) {
    Point p = new Point();
    System.out.println(p.x + " " + p.y);   // 3 3
  }
}
```

ELTE
IK

Mező alapértelmezett inicializációja

Automatikusan egy nulla-szerű értékre!

```
class Point {
  int x, y = 3;
}

class Main {
  public static void main(String[] args) {
    Point p = new Point();
    System.out.println(p.x + " " + p.y);    // 0 3
  }
}
```

ELTE
IK

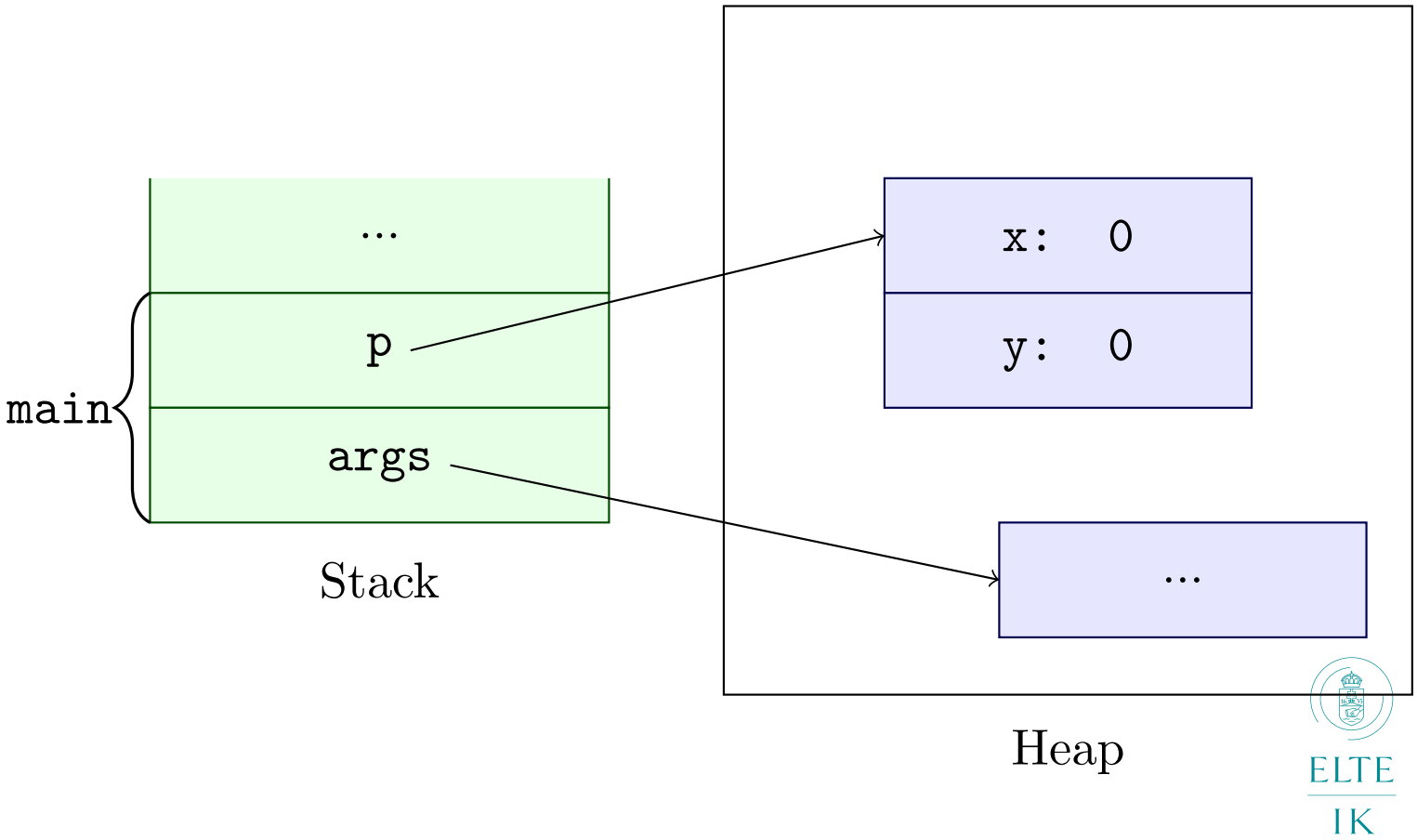
Metódus

```
class Point {
  int x, y;   // kezdetben 0, 0
  void move(/* Point this, */ int dx, int dy) {
            // this: implicit paraméter
    this.x += dx;
    this.y += dy;
  }
}

class Main {
  public static void main(String[] args) {
    Point p = new Point();
    p.move(3,3);                  // p -> this, 3 -> dx, 3 -> dy
  }
}
```

ELTE
IK

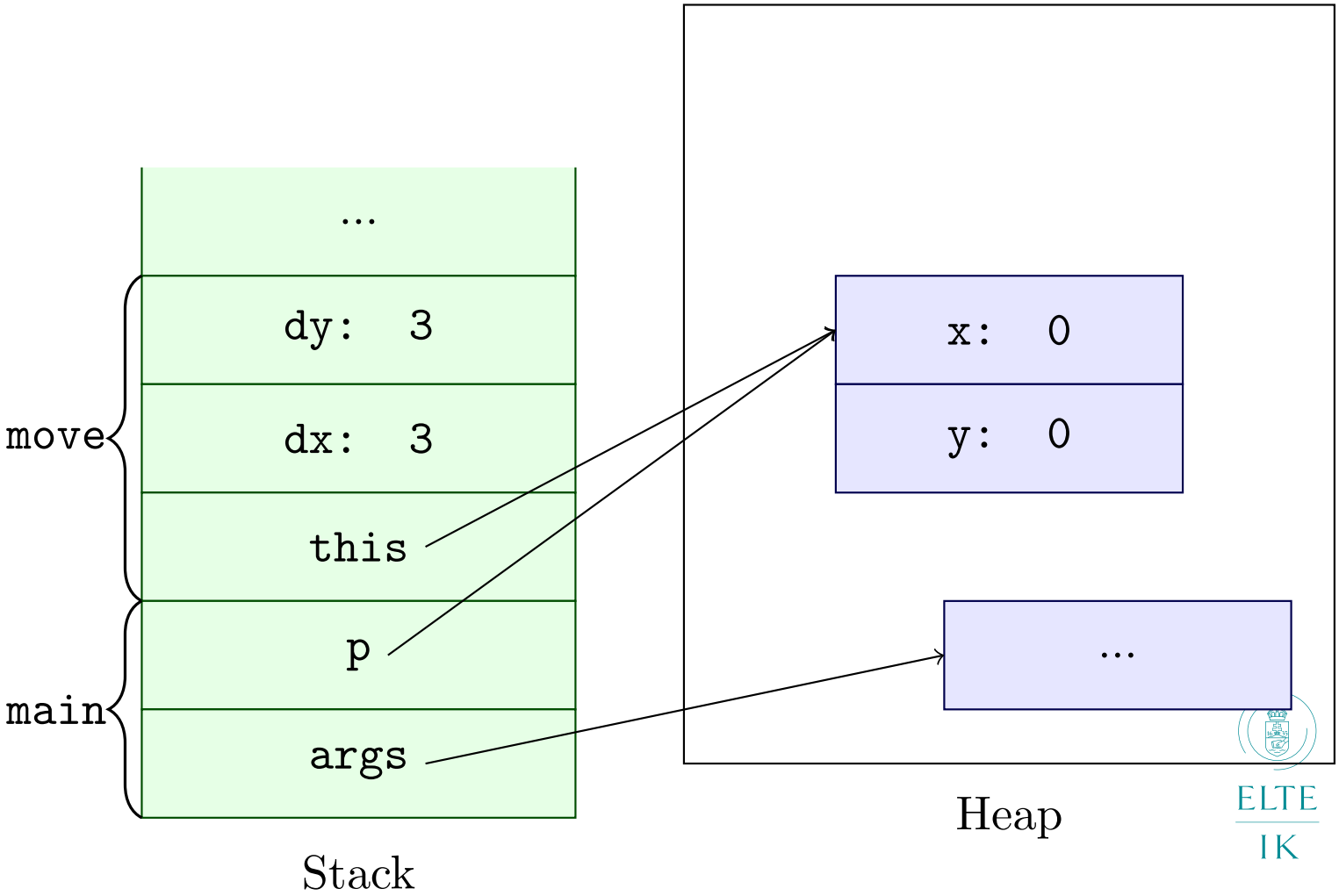
Metódus aktivációs rekordja – 1

```
Point p = new Point();
```



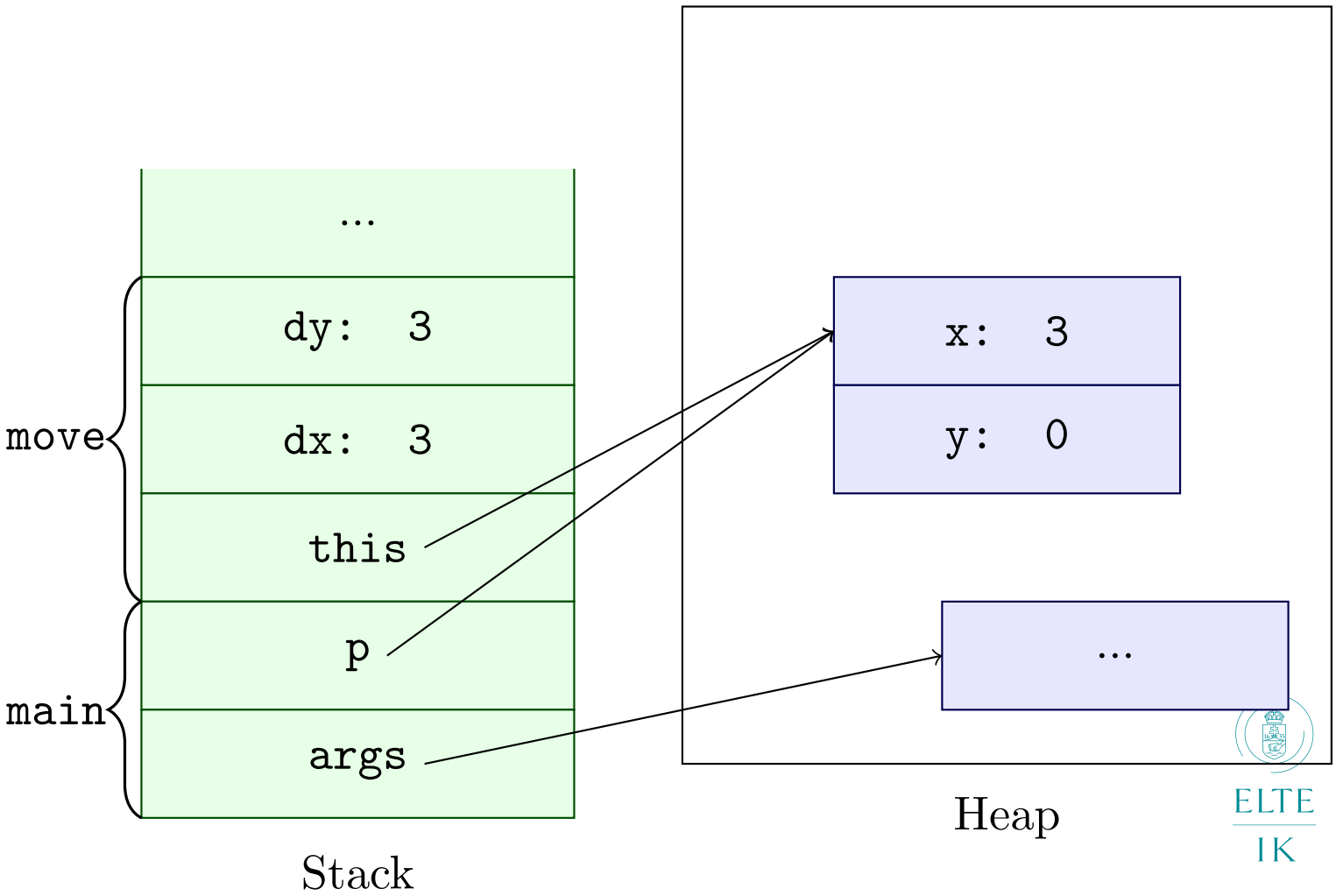
Metódus aktivációs rekordja – 2

```
p.move(3,3);
```



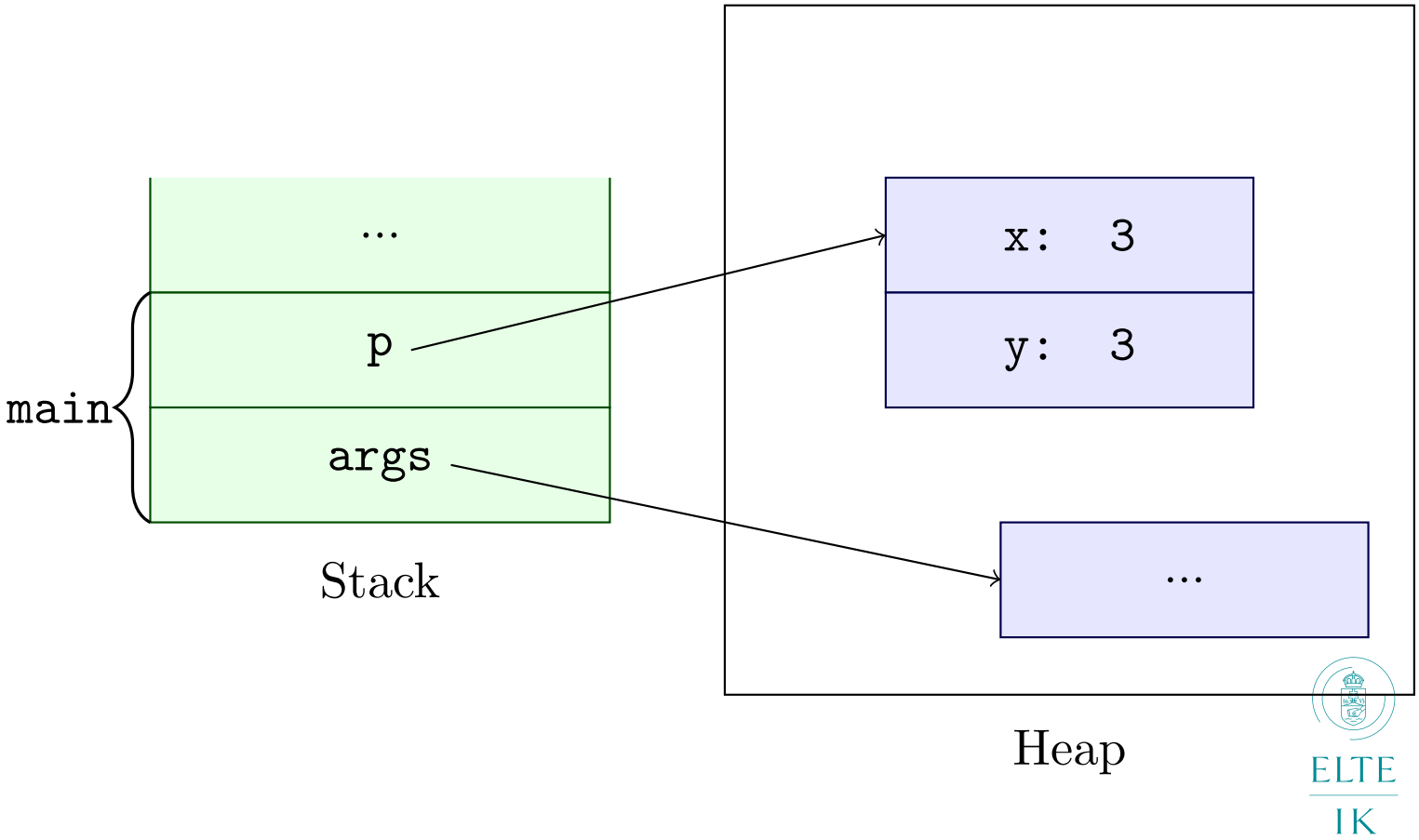
Metódus aktivációs rekordja – 3

```
this.x += dx;
```



Metódus aktivációs rekordja – 4

```
System.out.println(p.x + " " + p.y);
```



A `this` implicit lehet

```
class Point {
  int x, y;   // 0, 0
  void move(int dx, int dy) {
    this.x += dx;
    y += dy;
  }
}

class Main {
  public static void main(String[] args) {
    Point p = new Point();
    p.move(3,3);
  }
}
```

ELTE
IK

Inicializálás konstruktorral

```
public class Point {
  int x, y;
  Point(/* Point this, */ int initialX, int initialY) {
        // this: implicit paraméter
    this.x = initialX;
    this.y = initialY;
  }
}

class Main {
  public static void main(String[] args) {
    Point p = new Point(0,3);
    System.out.println(p.x + " " + p.y);   // 0 3
  }
}
```

ELTE
IK

Inicializálás konstruktorral – a this elhagyható

```
public class Point {
  int x, y;
  public Point(int initialX, int initialY) {
    x = initialX;
    y = initialY;
  }
}

class Main {
  public static void main(String[] args) {
    Point p = new Point(0,3);
    System.out.println(p.x + " " + p.y);    // 0 3
  }
}
```

ELTE
IK

Nevek újrahasznosítása

```
public class Point {
  int x, y;
  Point(int x, int y) { // elfedés
    this.x = x;         // minősített (qualified) név
    this.y = y;         // konvenció
  }
}

class Main {
  public static void main(String[] args) {
    Point p = new Point(0,3);
    System.out.println(p.x + " " + p.y);   // 0 3
  }
}
```

ELTE
IK

Paraméter nélküli konstruktor

```
public class Point {
  int x, y;
  public Point() {}
}

public class Main {
  public static void main(String[] args) {
    Point p = new Point();
    System.out.println(p.x + " " + p.y);    // 0 0
  }
}
```

ELTE
IK

Alapértelmezett (default) konstruktor

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

class Main {
  public static void main(String[] args) {
    Point p = new Point();
    System.out.println(p.x + " " + p.y);   // 0 0
  }
}
```

Generálódik egy paraméter nélküli, üres konstruktor

```
public Point() {}
```

IK

Egységbe zárás

```
public class Time {
  int hour;
  int min;
  Time(int hour, int min) {
    this.hour = hour;
    this.min = min;
  }
  void aMinPassed() {
    if (min < 59) {
      ++min;
    } else { ... }
  } // (C) Monty Python
}
```

```
Time morning = new Time(6,10);
morning.aMinPassed();
int hour = morning.hour;
```

ELTE
IK

Típusinvariáns

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

Értelmetlen érték létrehozása

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

```
Time morning = new Time(6,10);
morning.aMinPassed();
int hour = morning.hour;

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

ELTE
IK

létrehozásnál típusinvariáns biztosítása

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

Kerüljük el a „silent failure'' jelenséget

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

Segédfüggvény

```
public class Time {
  ...
  public Time(int hour, int min) {
    if (isBetween(hour, 0, 24) && isBetween(min, 0, 60)) {
      this.hour = hour;
      this.min = min;
    } else {
       throw new IllegalArgumentException("Wrong time!");
    }
  }
  // segédfüggvény: a kód könnyebb megértését segíti
  private boolean isBetween(int value, int min, int max) {
    return min <= value && value < max;
  }
}
```

„Early return”

```
public class Time {
  public Time(int hour, int min) {
    // early return/throw: a speciális eseteket elöl kezeli
    if (!isBetween(hour, 0, 24) || !isBetween(min, 0, 60)) {
        throw new IllegalArgumentException("Wrong time!");
    }

    // "happy path": a kód szokásos lefutása
    this.hour = hour;
    this.min = min;
  }

  ...
}
```

Kivétel

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

ELTE
IK

Futási hiba

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

```
$ javac Time.java
$ javac Main.java
$ java Main
Exception in thread "main" java.lang.IllegalArgumentException:
Wrong time!
    at Time.<init>(Time.java:9)
    at Main.main(Main.java:3)
$
```

A mezők közvetlenül manipulálhatók

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

```
class Main {
    public static void main(String[] args) {
        Time morning = new Time(6,10);
        morning.aMinutePassed();

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

Mező elrejtése: private

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

```
class Main {
    public static void main(String[] args) {
        Time morning = new Time(6,10);
        morning.aMinutePassed();

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

Idióma: privát állapot csak műveleteken keresztül

```
public class Time {
  private int hour;                    // 0 <= hour < 24
  private int min;                     // 0 <= min  < 60
  public Time(int hour, int min) { ... }
  int getHour() { return hour; }
  int getMin()  { return min;  }
  void setHour(int hour) {
    if (0 <= hour && hour <= 23) {
      this.hour = hour;
    } else {
      throw new IllegalArgumentException("Wrong hour!");
    }
  }
  void setMin(int min) { ... }
  void aMinPassed()    { ... }
}
```

ELTE
IK

Getter-setter konvenció

Lekérdező és beállító művelet neve

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

ELTE
IK

Reprezentáció változtatása

```
public class Time {
  private short mins;
  public Time(int hour, int min) {
    if (...) throw new IllegalArgumentException("Wrong time!");
    mins = 60*hour + min;
  }
  int getHour() { return mins / 60; }
  int getMin() { return mins % 60; }
  void setHour(int hour) {
    if (...) throw new IllegalArgumentException("Wrong hour!");
    mins = 60 * hour + getMin();
  }
  void setmin(int min) { ... }
  void aMinPassed() { ... }
}
```

ELTE
IK

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
- Javasolt továbbá
  - ◇ Erős kohézió (az osztálynak egyetlen, jól megadott célja van)

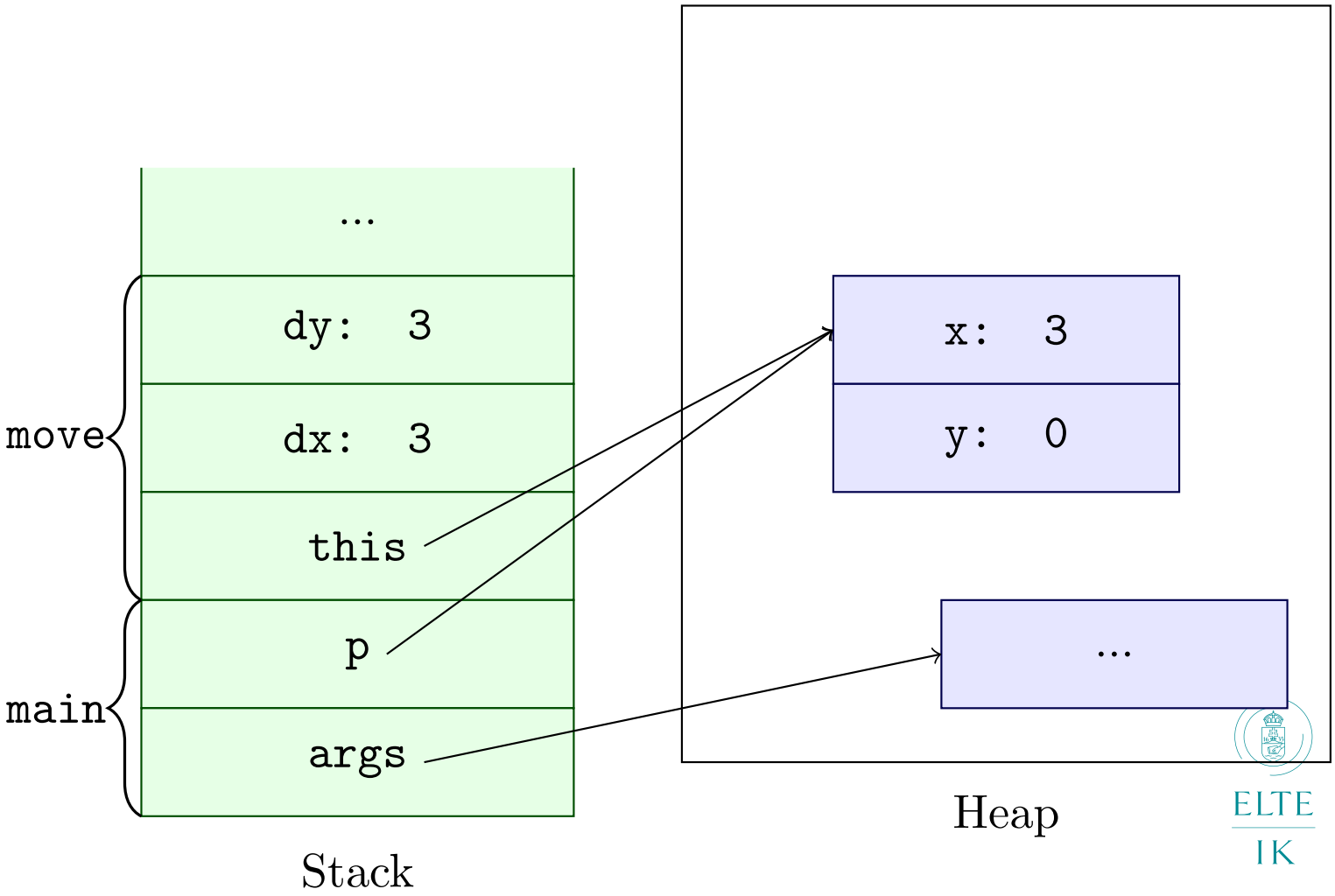
ELTE
IK

Hivatkozás (referencia)

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

- Osztály típusú változó
- Objektumra hivatkozik
- Heap
- Létrehozás: `new`
- Dereferálás (hivatkozás feloldása): `.`

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
- ...

ELTE
IK

Ábrázolás a memóriában

Végrehajtási verem

Lokális változók és paraméterek
(Primitív típusú, referencia)

Heap

Objektumok, mezőik
(Primitív típusú, referencia)

Példányváltozó: a mezőnek megfelelő adattároló az objektumban.

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

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

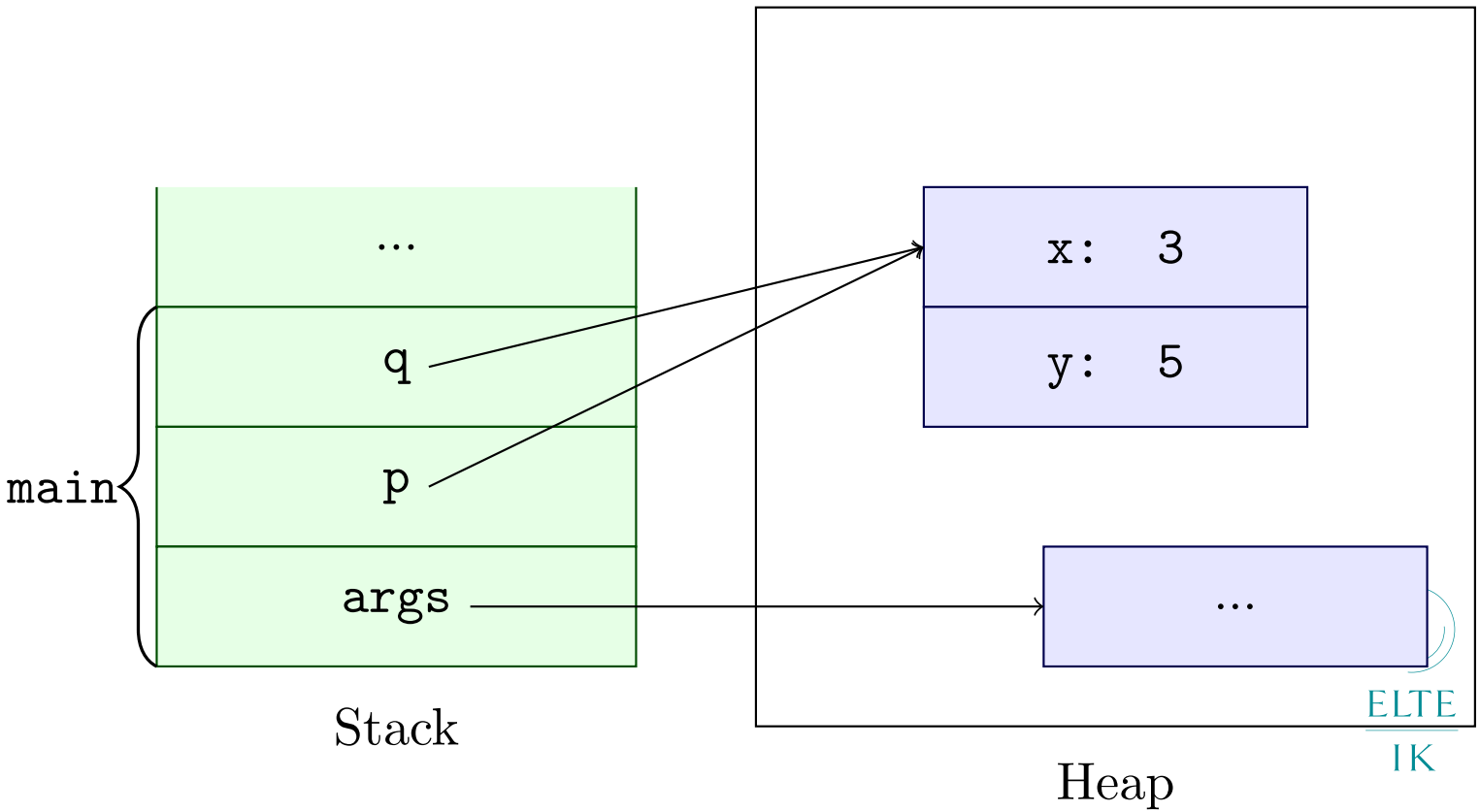
Objektumok élettartama

- 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
```

Üres referencia

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

I call it my billion-dollar mistake.
— Tony Hoare, a `null` megalkotója

Üres referencia

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

I call it my billion-dollar mistake.
— Tony Hoare, a `null` megalkotója

Ezek az Erő sötét oldalához tartoznak. [...] Óvakodjál te tőlük, óvakodjál! Mert nagy árat fizetsz a hatalomért, amivel ők felruháznak. — Yoda

ELTE
IK

Üres referencia

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

I call it my billion-dollar mistake.
— Tony Hoare, a `null` megalkotója

Ezek az Erő sötét oldalához tartoznak. [...] Óvakodjál te tőlük, óvakodjál! Mert nagy árat fizetsz a hatalomért, amivel ők felruháznak. — Yoda

Mezők inicializálása

Automatikusan, nulla-szerű értékre

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

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

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

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

ELTE
IK

Inicializálás üres referenciára

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

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

ELTE
IK

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!

ELTE
IK

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
}
```

1K

Statikus mezők

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

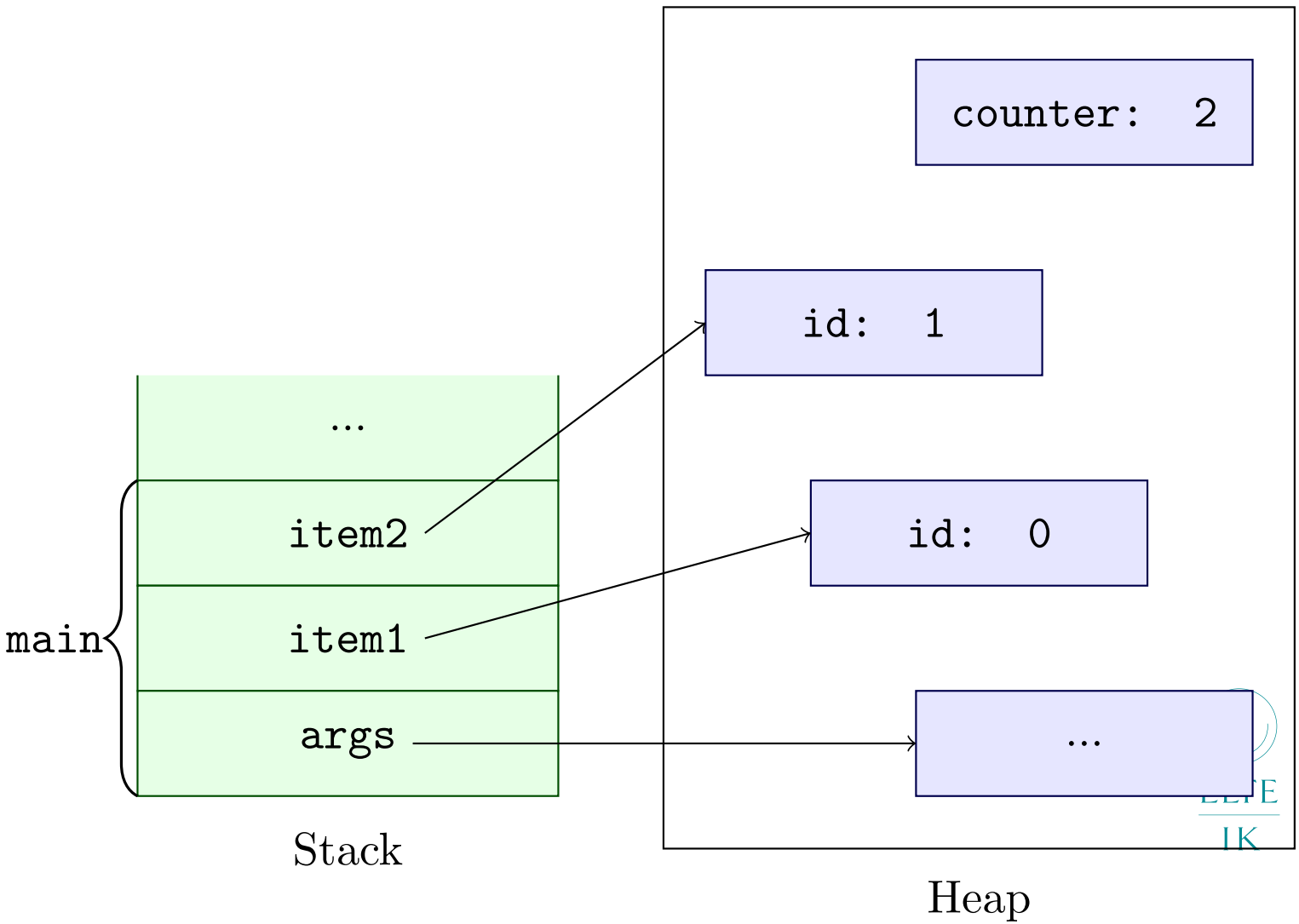
```
class Item {
  static int counter = 0;
}

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

ELTE
IK

Osztályszintű és példányszintű mezők

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

public class Main {
  public static void main(String[] args) {
    Item item1 = new Item(), item2 = new Item();
    System.out.println(item1.id);
    System.out.println(item2.id);

    System.out.println(Item.counter);  // így a helyes
    System.out.println(item1.counter); // csúf, kerülendő
  }                                    // (a Java elfogadja)
}
```

```
Item item1 = new Item(), item2 = new Item();
```



Statikus metódusok

- Hasonló a C globális függvényeihez
- Az osztályon keresztül hívható meg, objektum nélkül is lehet
- Nem kap implicit paramétert (`this`)
- A statikus mezők logikai párja

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

ELTE
IK

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
  }
}

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

Osztályszintű tagok használata

Metódusok

- A `main` kötelezően ilyen
- Tiszta („matematikai jellegű'') számítások
- Segédmetódus

Adattagok: erősen korlátoz, csak indokolt esetben használandó

- Függőség példányok között
- Nehezebb a kód helyességét belátni/tesztelni
- Nehezebb megváltoztatni a kódot
- Csökkenti az enkapszulációt

ELTE
IK

Szemétgyűjtés

Feleslegessé vált objektumok felszabadítása

Helyes

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

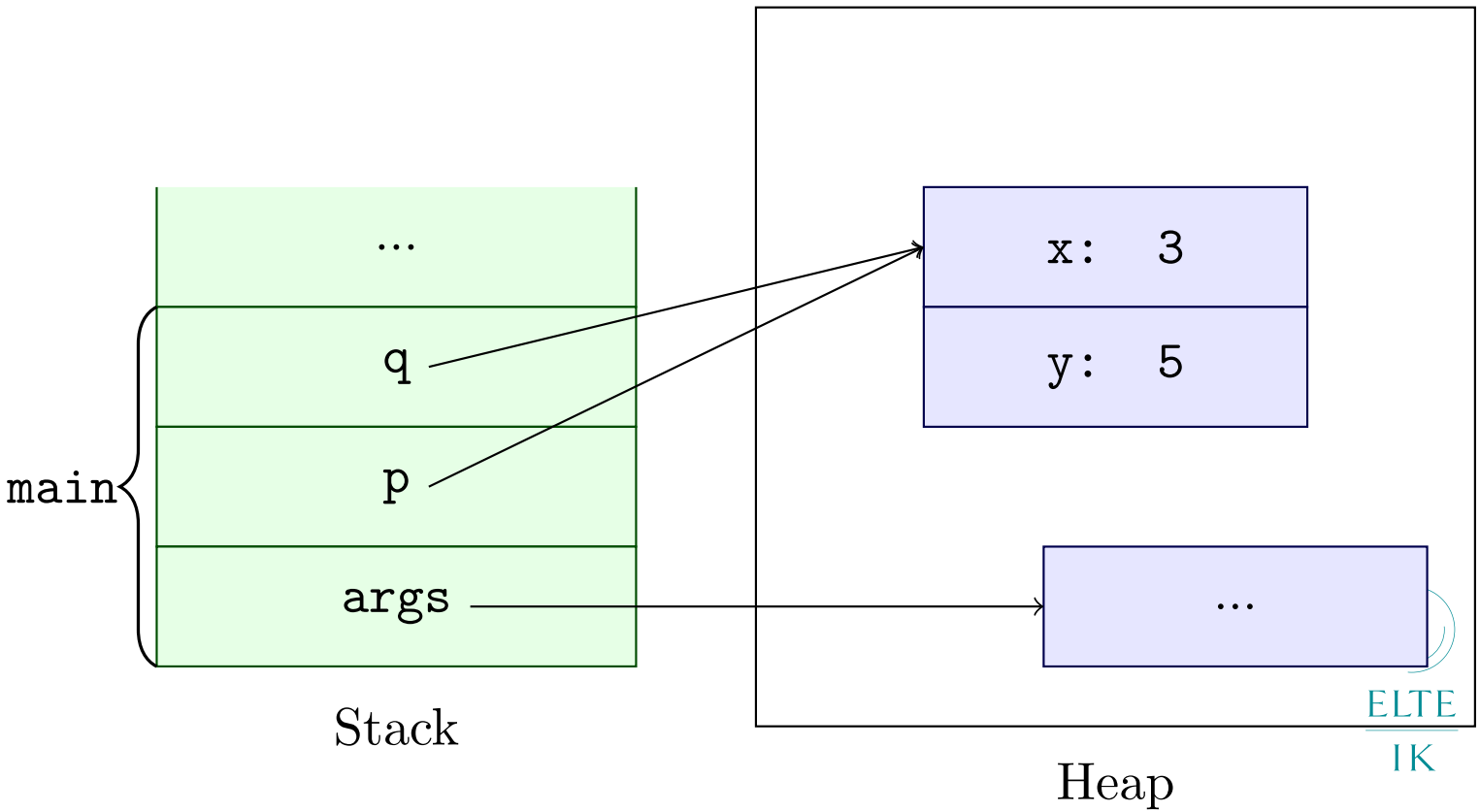
Teljes

Mindent felszabadít, amit nem lehet már elérni

ELTE
IK

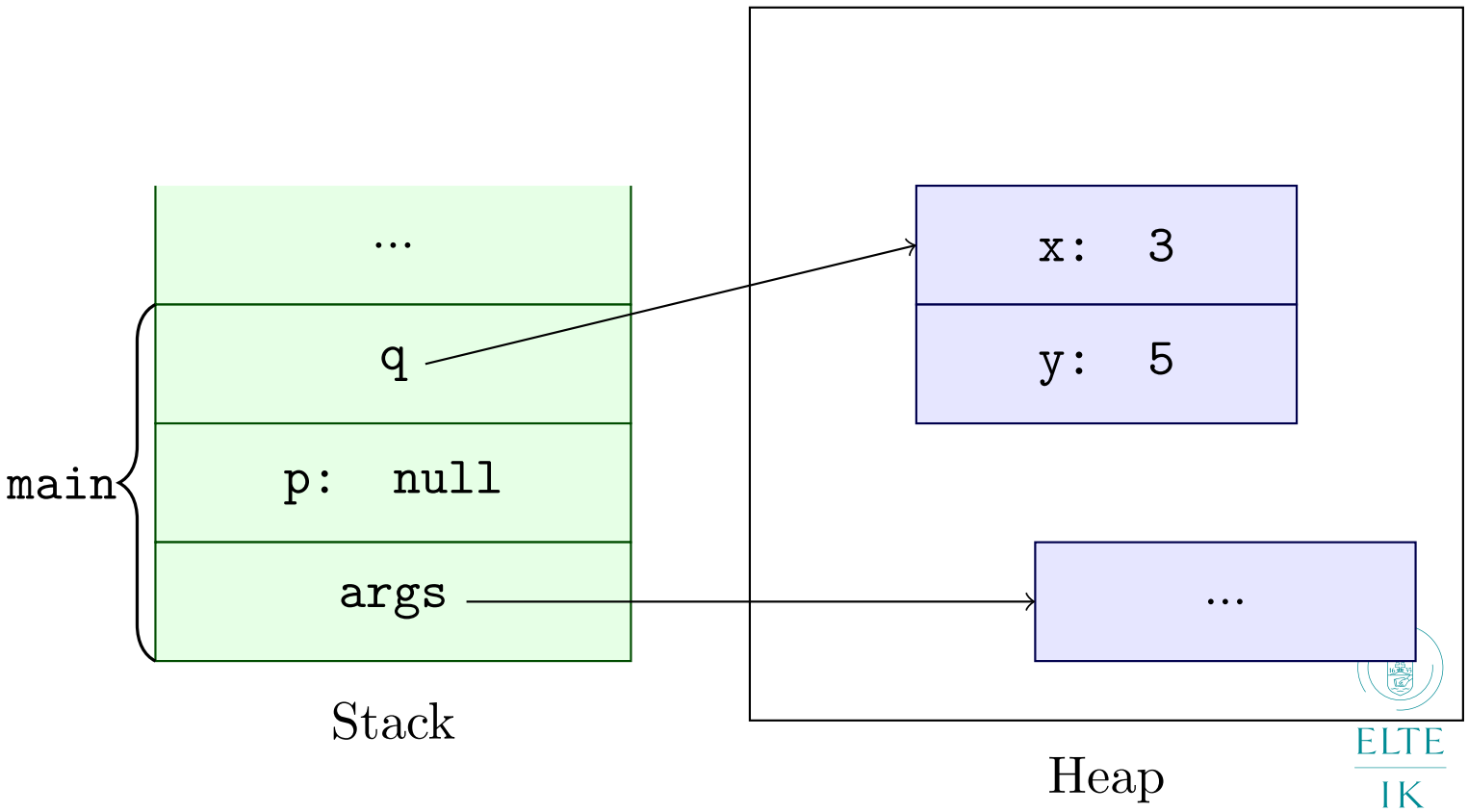
Még nem szabadítható fel

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



Még mindig nem szabadítható fel

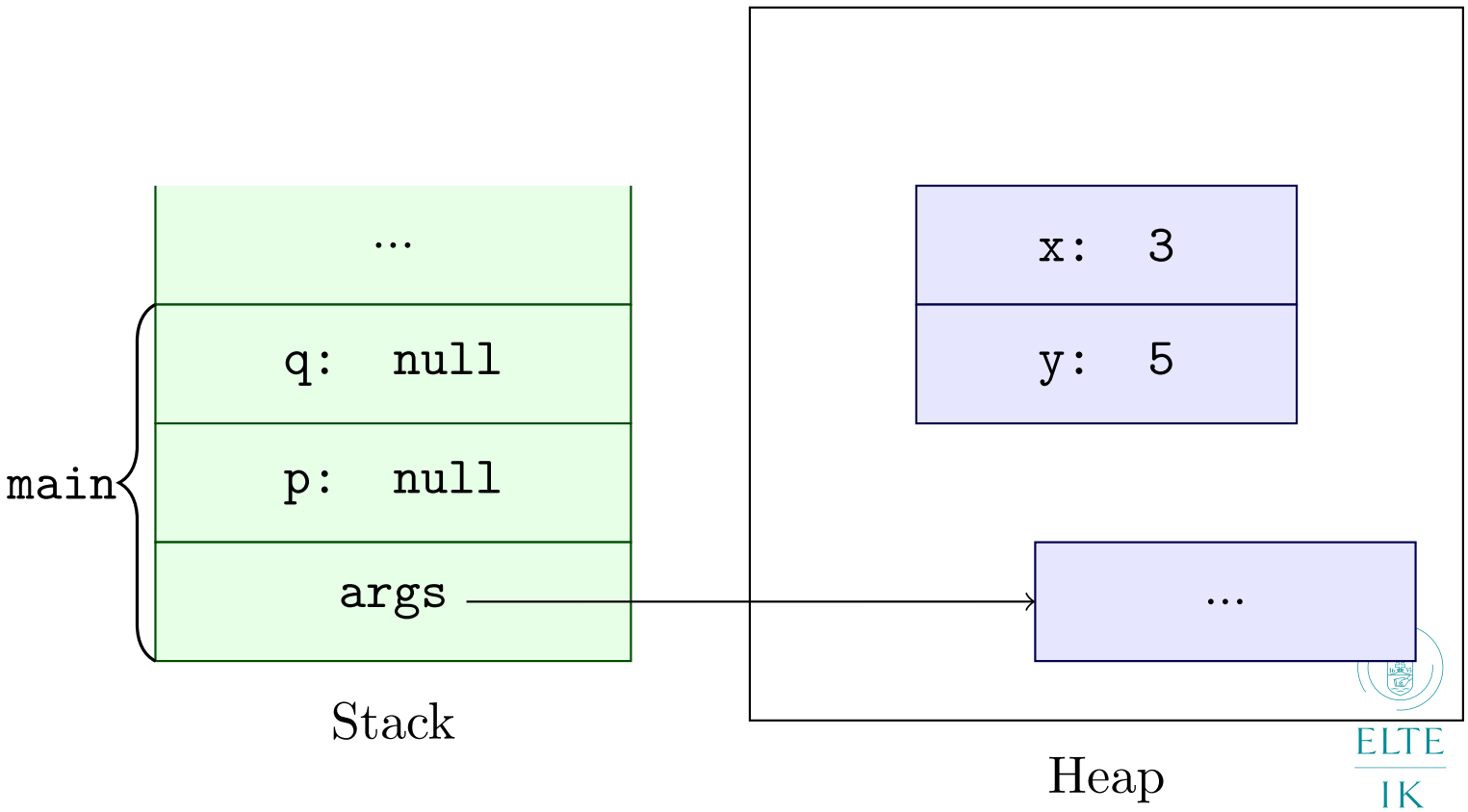
```
p = null;
```



ELTE IK

Már felszabadítható

```
q = null;
```



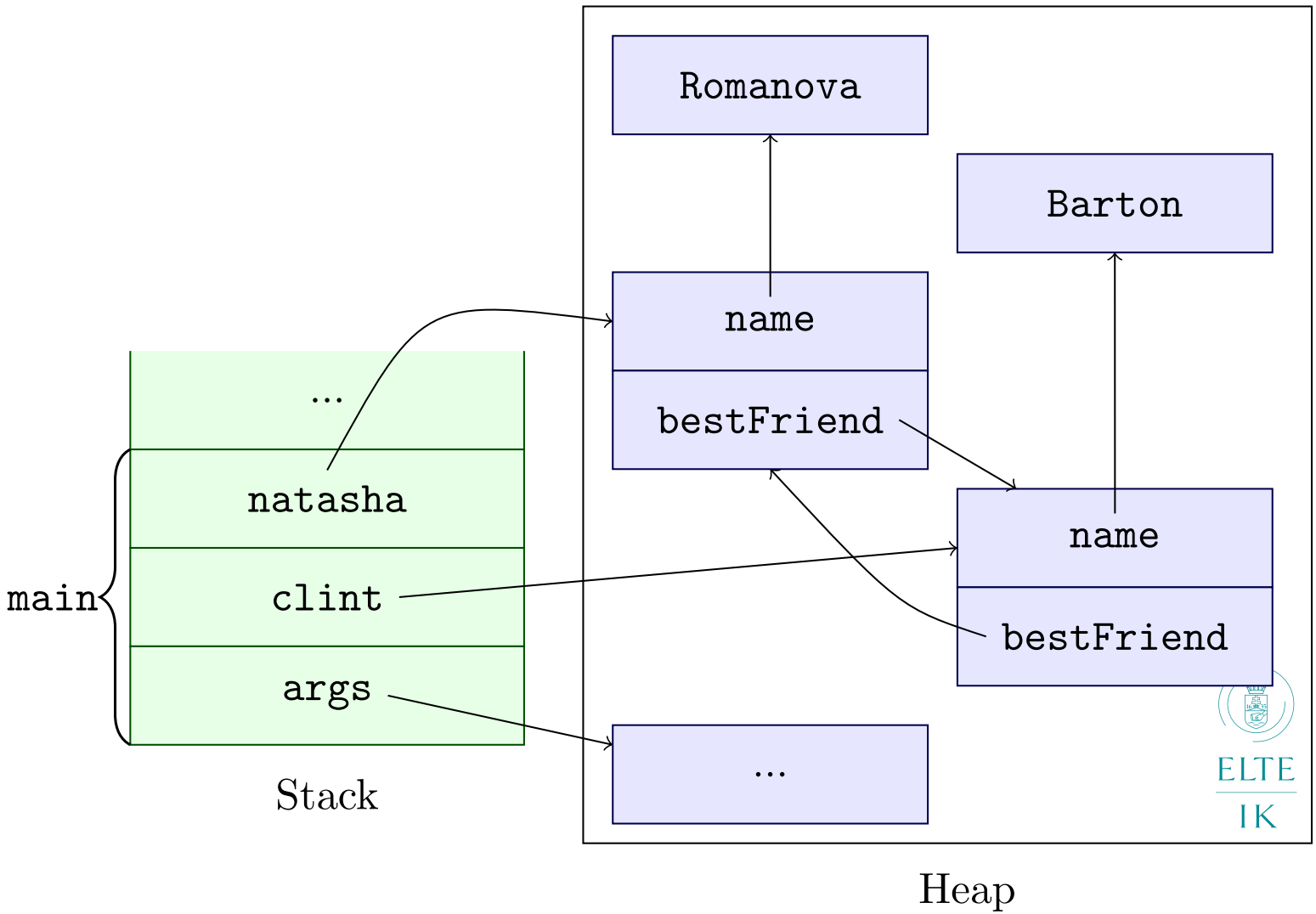
Bonyolultabb példa

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

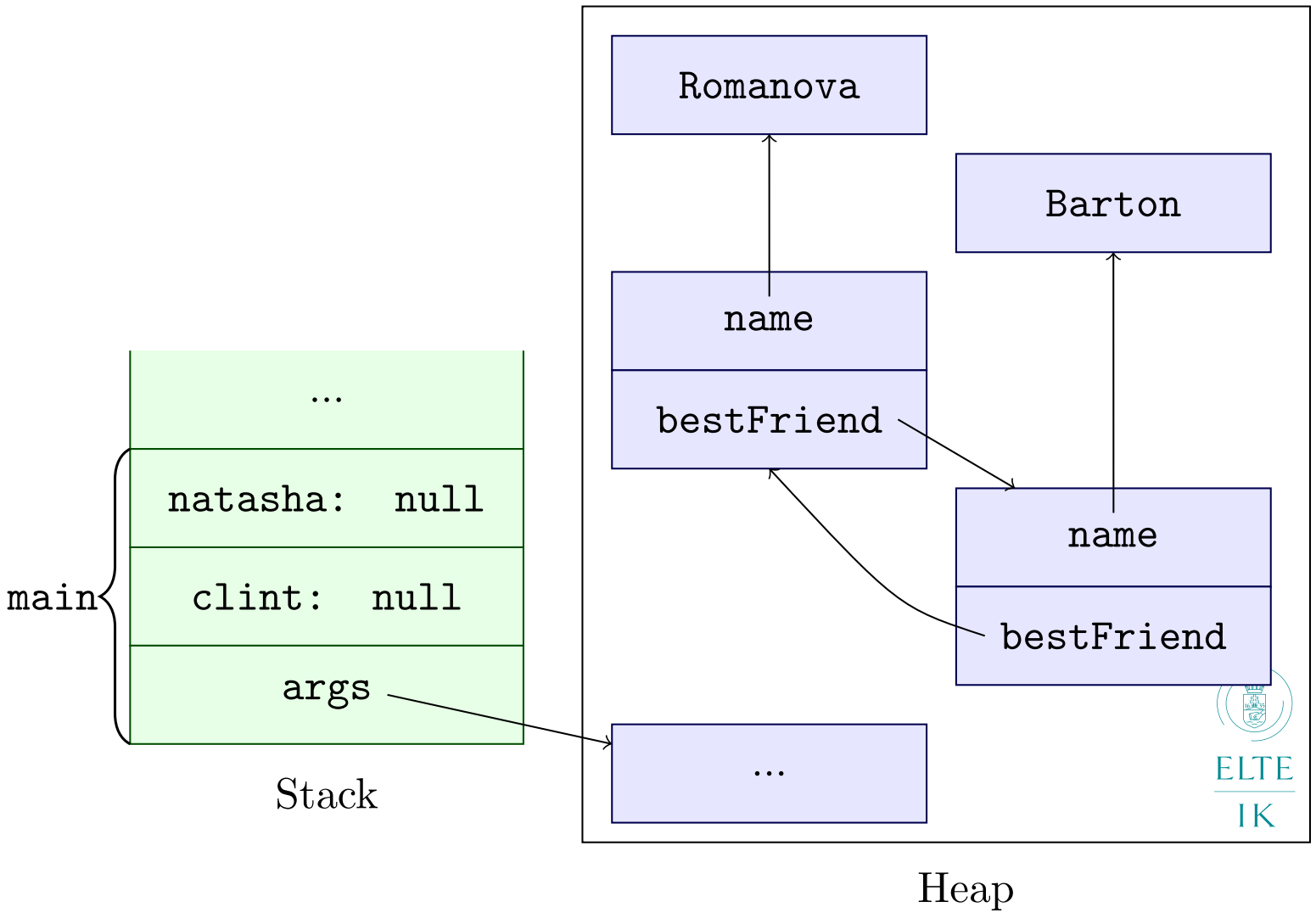
```
Hero clint   = new Hero();
Hero natasha = new Hero();

clint.name          = "Barton";
natasha.name        = "Romanova";
clint.bestFriend    = natasha;
natasha.bestFriend = clint;
```

IK

Hősök a memóriában



```
natasha = clint = null;
```

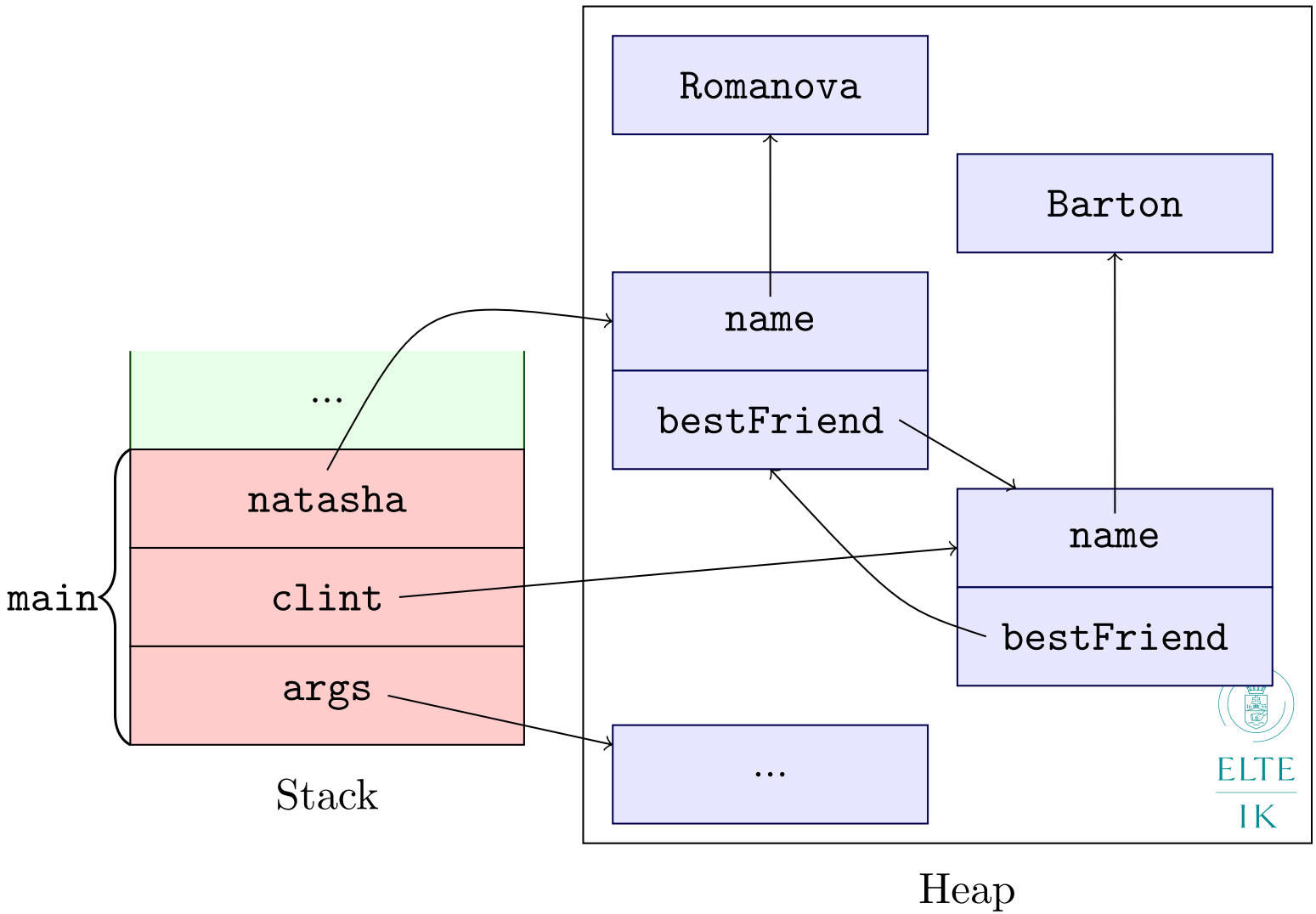


Mark-and-Sweep szemétgyűjtés

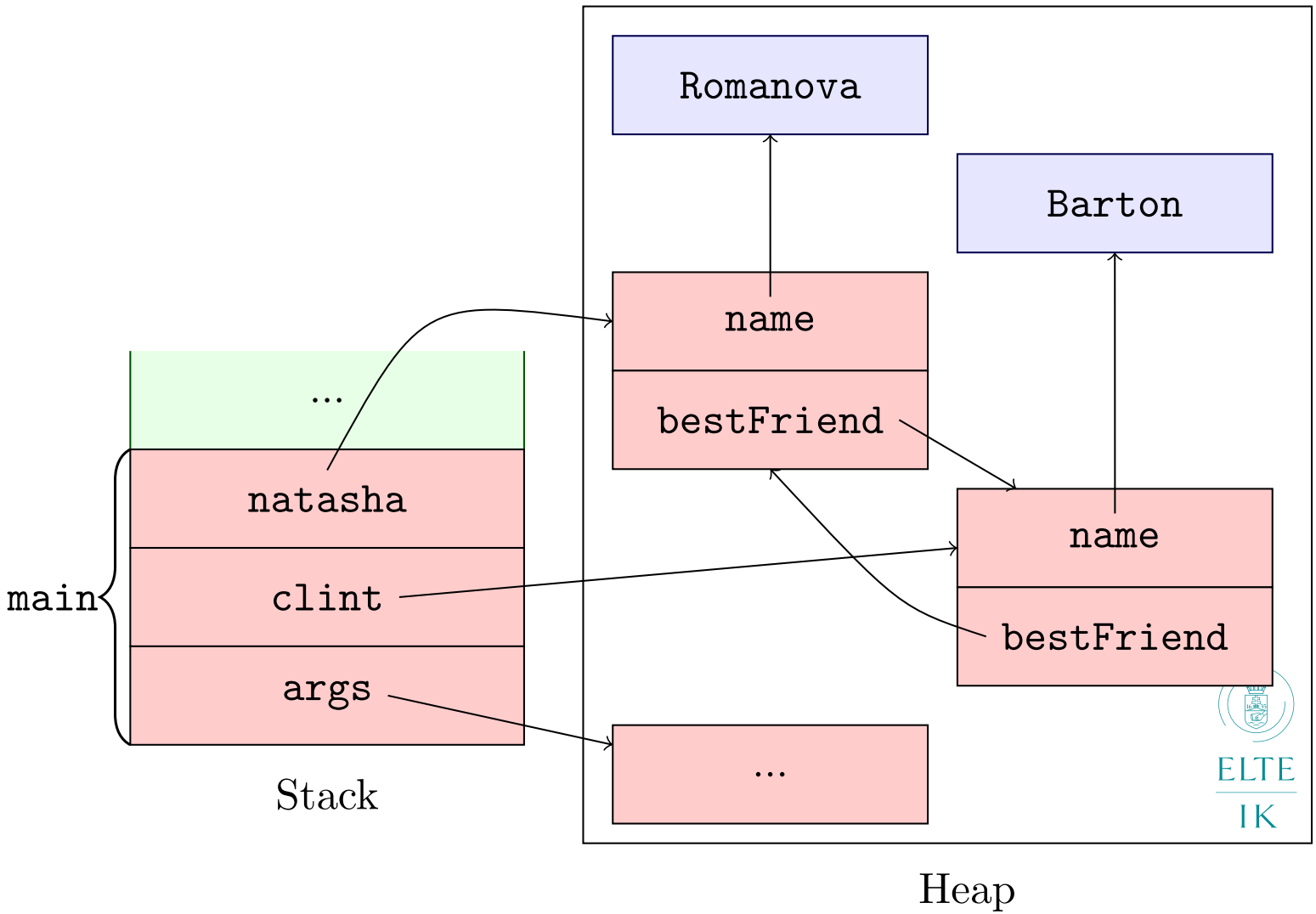
- Mark fázis
  - ◇ Kiindulunk a vermen lévő referenciákból
  - ◇ Megjelöljük a belőlük elérhető objektumokat
    - ▶ Megjelöljük az azokból elérhetőeket
    - ▶ ... amíg tudunk újabbat megjelölni (tranzitív lezárt)
  - ◇ „Stop the world'': a program nem fut eközben
- Sweep fázis
  - ◇ A jelöletlen objektumok felszabadíthatók

ELTE
IK

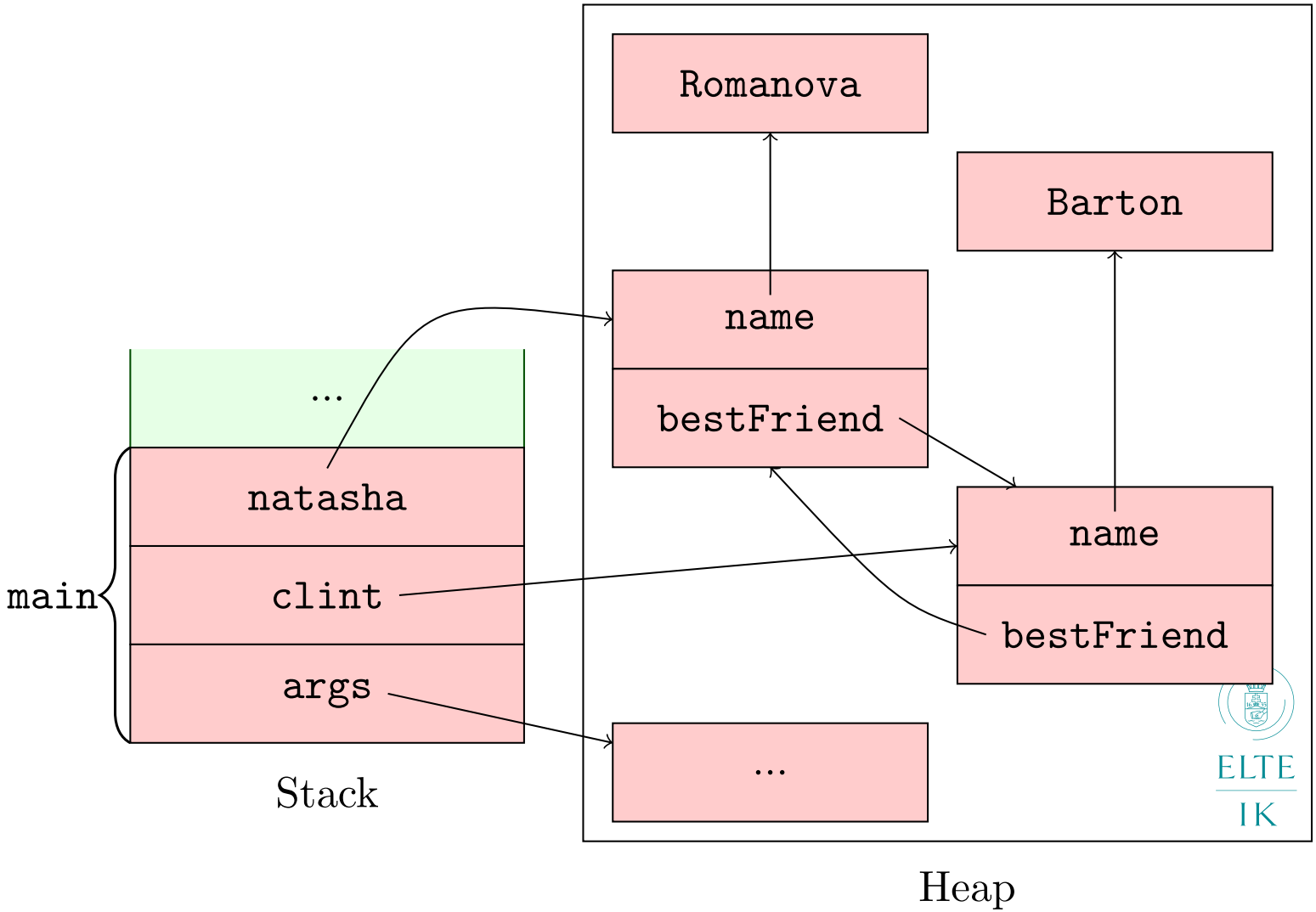
Mark-and-sweep: root set



Mark-and-sweep: propagálás

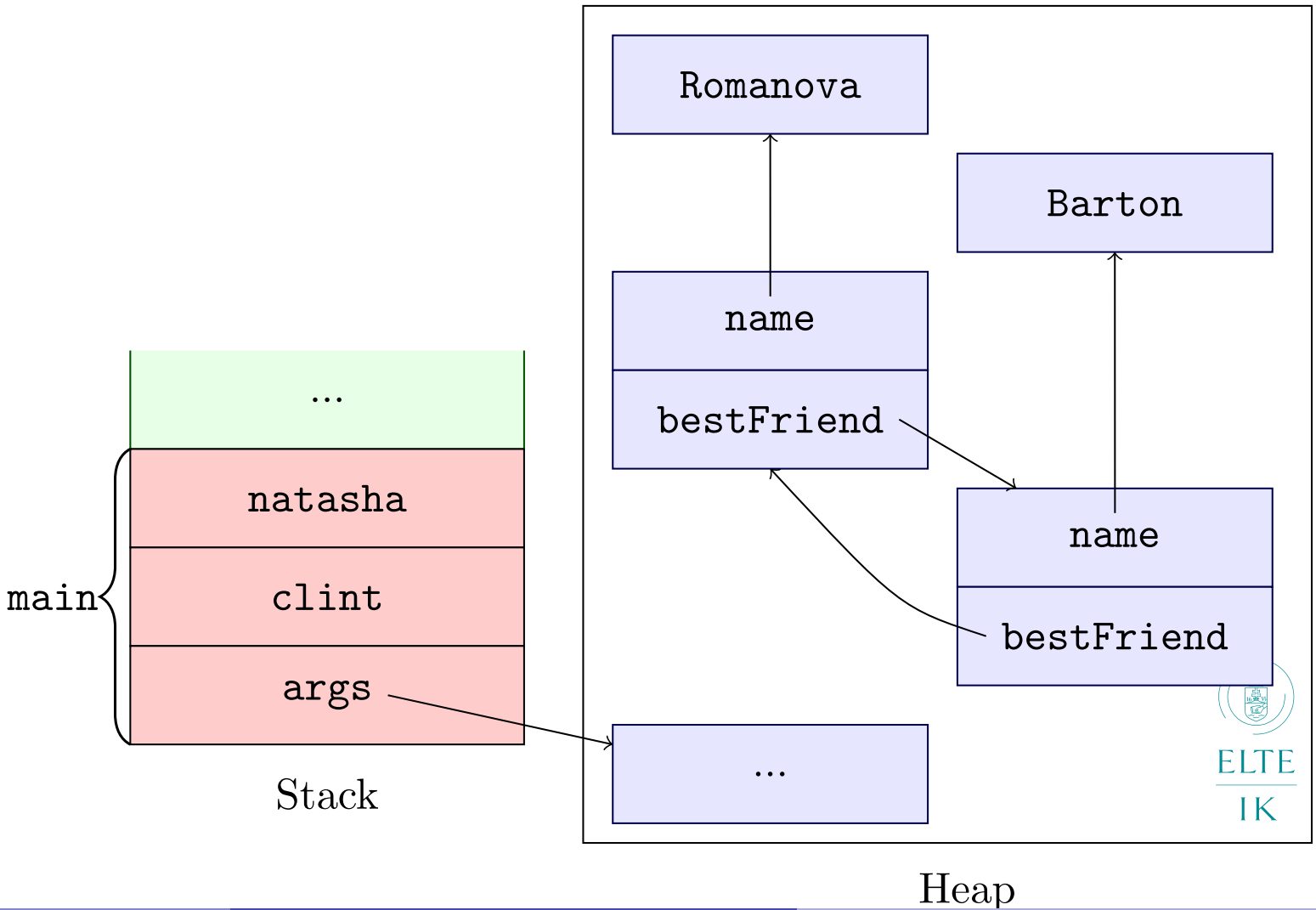


Mark-and-sweep: itt most mindegyik objektum elérhető



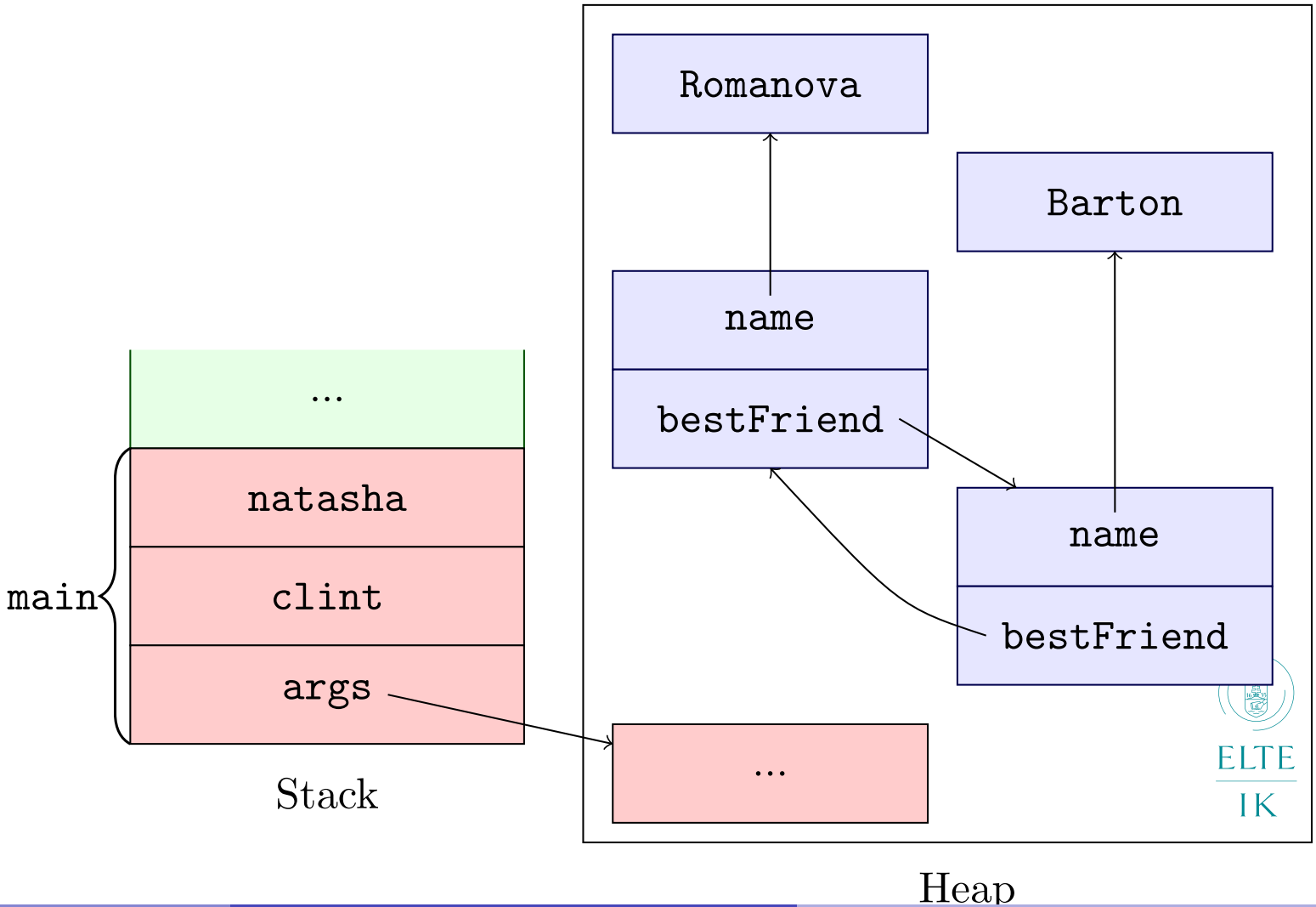
Mark-and-sweep: `natasha = clint = null;` ismét

```
natasha = clint = null;
```



Mark-and-sweep: mark fázis vége

- A sweep fázis felszabadítja az elérhetetlen objektumokat



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);
    }
}
```

ELTE IK

Tömbök létrehozása, feltöltése, rendezése

```
public class Sort {
  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);

    for (int n: numbers) { System.out.println(n); }
  }
}
```

ELTE
IK

Importálás

```
import java.util.Arrays;
public class Sort {
  public static void main(String[] args) {
    int[] numbers = new int[args.length];

    for (int i = 0; i < args.length; ++i) {
      numbers[i] = Integer.parseInt(args[i]);
    }

    Arrays.sort(numbers);

    for (int n: numbers) { System.out.println(n); }
  }
}
```

ELTE
IK

Statikus tagok importja

```
import static java.util.Arrays.sort;
public class Sort {
  public static void main(String[] args) {
    int[] numbers = new int[args.length];

    for (int i = 0; i < args.length; ++i) {
      numbers[i] = Integer.parseInt(args[i]);
    }

    sort(numbers);

    for (int n: numbers) { System.out.println(n); }
  }
}
```

ELTE
IK

Változó paraméterszámú metódus (vararg)

```
private static int[] getInts(String... txts) {
  int[] retval = new int[txts.length];
  for (int i = 0; i < txts.length; ++i) {
    retval[i] = Integer.parseInt(txts[i]);
  }
  return retval;
}

public static void main(String... args) {
  int[] nums = getInts("1", "2", "3");
          // = getInts(new String[]{ "1", "2", "3" });
          // = getInts("1 2 3".split(" "));
          // = getInts(args);
  sort(nums);
  for (int num: nums) { System.out.println(num); }
}
```

ELTE
IK

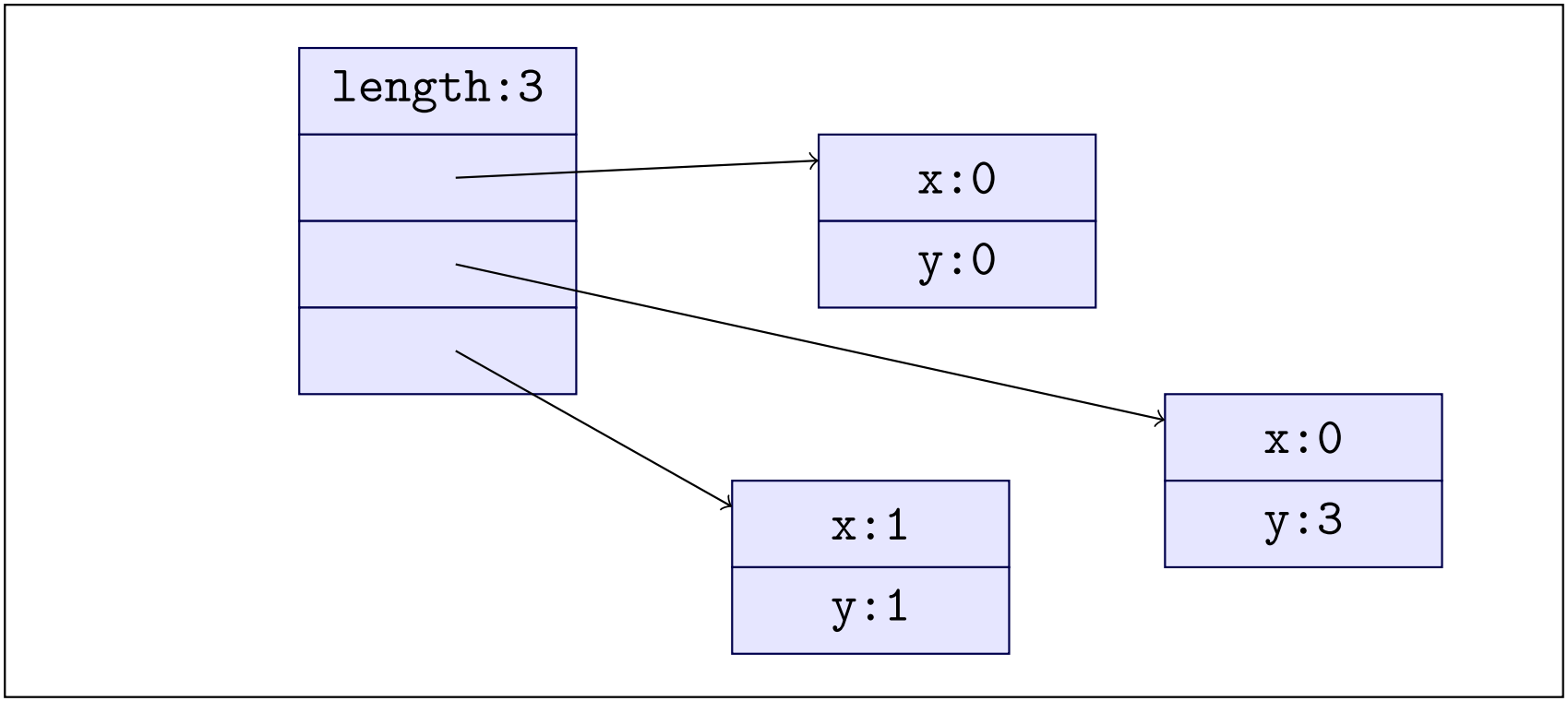
Referenciák tömbje

```
Point[] triangle = { new Point(0,0),
                     new Point(0,3),
                     new Point(1,1) };
```

ELTE
IK

Referenciák tömbje

```
Point[] triangle = { new Point(0,0),
                     new Point(0,3),
                     new Point(1,1) };
```



Heap

Lépésről lépésre

```
static void séta() {
    Láb[] százlábú;
    System.out.println(százlábú.length);
```

Lépésről lépésre

```
static void séta() {
    Láb[] százlábú;
    System.out.println(százlábú.length);

    százlábú = null;
    System.out.println(százlábú.length);
```

ELTE
IK

Lépésről lépésre

```
static void séta() {
    Láb[] százlábú;
    System.out.println(százlábú.length);

    százlábú = null;
    System.out.println(százlábú.length);

    százlábú = new Láb[100];
    System.out.println(százlábú.length);
```

ELTE
IK

Lépésről lépésre

```
static void séta() {
    Láb[] százlábú;
    System.out.println(százlábú.length);

    százlábú = null;
    System.out.println(százlábú.length);

    százlábú = new Láb[100];
    System.out.println(százlábú.length);

    for (int i = 0; i<100; i+=2) {
        százlábú[i]   = new Láb("bal");
        százlábú[i+1] = new Láb("jobb");
    }
}
```

ELTE
IK

Mátrix

```
double[][] id3 = { {1,0,0}, {0,1,0}, {0,0,1} };
```

Mátrix

```
double[][] id3 = { {1,0,0}, {0,1,0}, {0,0,1} };

static double[][] id(int n) {
    double[][] matrix = new double[n][n];
    for (int i=0; i<n; ++i) {
        matrix[i][i] = 1;
    }
    return matrix;
}
```

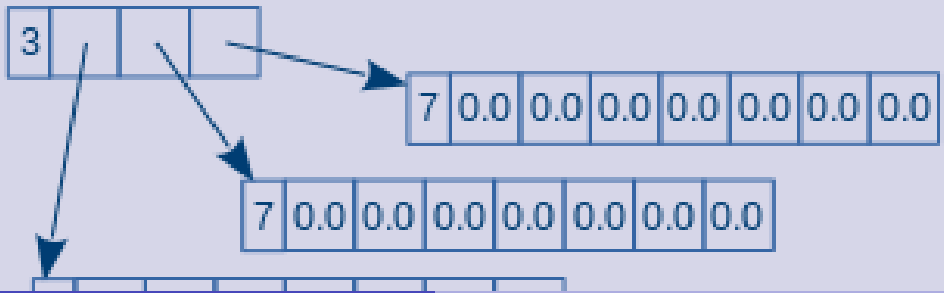
C versus Java

Többdimenziós tömb C-ben

```
double matrix[3][7];
for (int i=0; i<3; ++i)
    for (int j=0; j<7; ++j)
        matrix[i][j] = 0.0;
```

Tömbök tömbje Javában

```
double[][] matrix =
    new double[3][7];
```



Indexelés

Háromdimenziós tömb C-ben

```
T t[L][M][N];
```

$$\text{addr}(t_{i,j,k}) = \text{addr}(t) + ((i \cdot M + j) \cdot N + k) \cdot \text{sizeof}(T)$$

Tömbök tömbjének tömbje Javában

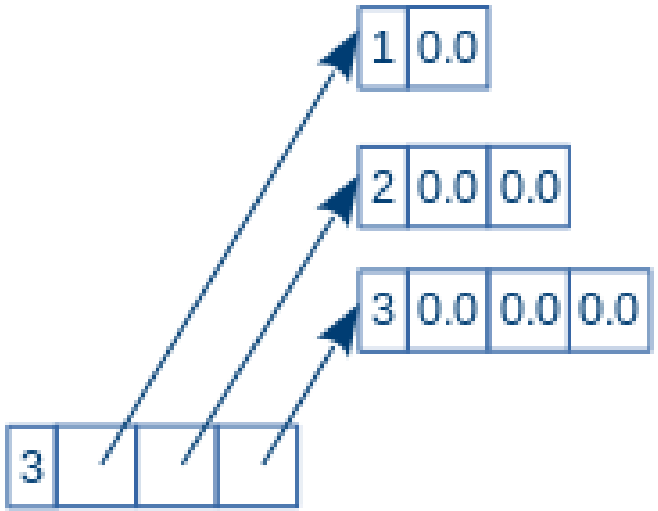
```
T[][][] t = new T[L][M][N];
```

$$\text{addr}(t_{i,j,k}) = \text{val}_8\Big(\text{val}_8(\text{addr}(t) + 4 + i \cdot 8) + 4 + j \cdot 8\Big) + 4 + k \cdot \text{sizeof}(T)$$

ELTE
IK

Alsóháromszög-mátrix

```
static double[][] zeroLowerTriangular(int n) {
    double[][] result = new double[n][];
    for (int i = 0; i<n; ++i) {
        result[i] = new double[i+1];
    }
    return result;
}
```



Parancssori argumentumok

- Javában: `String[] args`
- C-ben: `char *argv[]`
  - ◇ Ennek Java megfelelője: `char[][] argv`

Referencia típusok Javában

- Osztályok (`class`)
- Interfészek (`interface`)
- Felsorolási típusok (`enum`)
- Annotáció típusok (`@interface`)

ELTE
IK

Felsorolási típus

```
enum Day { MON, TUE, WED, THU, FRI, SAT, SUN }
```

- Referencia típus
- Értékek: objektumok, nem `int`ek

ELTE
IK

Felsorolási típus

```
enum Day { MON, TUE, WED, THU, FRI, SAT, SUN }
```

- Referencia típus
- Értékek: objektumok, nem `int`ek

```
Day best = Day.SAT;        // használható "import static" is
best = 3;                  // fordítási hiba
int n = best;              // fordítási hiba
int m = best.ordinal();    // 5
```

- A típusértékeket a felsorolási típus definiálja
- Nem lehet új példányt készíteni belőle
  - ◇ A konstruktor nem hívható meg: ~~`new Day()`~~

Konstruktorok, tagok

```
public enum Coin {
  PENNY(1), NICKEL(5), DIME(10), QUARTER(25);

  private final int centValue;

  Coin(int centValue) { this.centValue = centValue; }

  public int centValue() { return centValue; }

  public int percentageOf(Coin that) {
    return 100 * centValue / that.centValue();
  }
} // Source: Java Community Process (modified)
```

ELTE
IK

Előre definiált tagok

```
public enum Coin {
  PENNY(1), NICKEL(5), DIME(10), QUARTER(25); ...
}

Coin[] allCoins = Coin.values();
System.out.println(allCoins.length);              // 4
System.out.println(allCoins[0] == Coin.PENNY);    // true
System.out.println(allCoins[3] == Coin.QUARTER); // true

String txt = Coin.NICKEL.toString();
System.out.println(txt);                          // NICKEL

Coin coin = Coin.valueOf("DIME");
System.out.println(coin == Coin.DIME);            // true

System.out.println(Coin.PENNY.ordinal());    // 0
System.out.println(Coin.QUARTER.ordinal()); // 3
```

ELTE
IK

switch utasításban és kifejezésben

```
static int workingHours(Day day) {
  switch (day) {  // switch statement
    case SUN: case SAT: return 0;
    case FRI:           return 6;
    default:            return 8;
  }
}
```

ELTE
IK

switch utasításban és kifejezésben

```
static int workingHours(Day day) {
  switch (day) {  // switch statement
    case SUN: case SAT: return 0;
    case FRI:           return 6;
    default:            return 8;
  }
}

static int workingHours(Day day) {
  return switch (day) {  // Java 12+: switch expression
    case SAT, SUN -> 0;
    case FRI      -> 0;
    default       -> 2;
  };
}
```

ELTE
IK