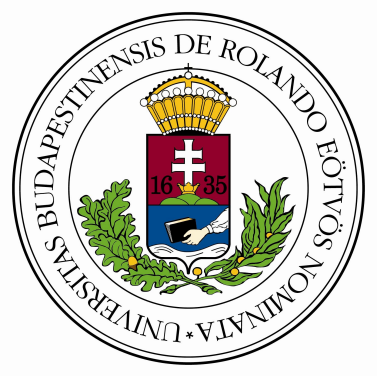
Programozási nyelvek – Java

előadások



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

A tárgy célja

- Fogalomrendszer
- Terminológia magyarul és angolul
- Tudatos nyelvhasználat
  - Objektum-orientált programozás
  - Imperatív programozás
- Részben: programozási készségek
- Linux és parancssori eszközök használata

Szervezés

- Fél félévre blokkosítva
- Párban a Programozási nyelvek (C++) tárggyal
- Gyakorlatvezetők
- Időpontok
- Konzultációk

Számonkérés

- Szerzendő pontok: max. 20
  - Gyakorlatokon: 5x2
  - Vizsgaidőszakban a zárthelyin: 1x10
- Pontozás
  1. – $[0, 10)$
  2. – $[10, 11)$
  3. – $[11, 15)$
  4. – $[15, 18)$
  5. – $[18, 20]$

Információk

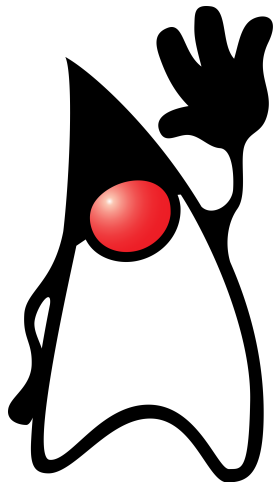
- Neptun
- Honlap
- Canvas
- BE-AD

Outline

A Java nyelv

- C-alapú szintaxis
- Objektumelvű (object-oriented)
  - Osztályalapú (class-based)
- Imperatív
  - Újabban kis FP-beütés
- Fordítás bájtkódra, JVM
- Erősen típusos
- Statikus + dinamikus típusrendszer
- Generikus, konkurens nyelvi eszközök

Java Language Specification



Jellemzői

- Könnyű/olcsó szoftverfejlesztés
- Gazdag infrastruktúra
  - Szabványos és egyéb programkönyvtárak
  - Eszközök
  - Kiterjesztések
  - Dokumentáció
- Platformfüggetlenség (JVM)
  - Write once, run everywhere
  - **Compile** once, run everywhere
- Erőforrásintenzív

JavaZone videó

Történelem

James Gosling és mások, 1991 (SUN Microsystems)

Oak → Green → **Java**

- Java 1.0 (1996)
- Java Community Process (1998)
- Java 1.2, J2SE (1998)
- J2EE (1999)
- J2SE 5.0 (2004)
- JVM GPL (2006)
- Oracle (2009)
- Java SE 8 (2014)
- Java SE 12 (2019)
- Game of Codes, Javazone 2014

Java Virtual Machine

- Alacsonyszintű nyelv: bájtkód
- Sok nyelv fordítható rá (Ada, Closure, Eiffel, Jython, Scala...)
- Továbbfordítható
  - Just In Time compilation
- Dinamikus szerkeszés
- Kódmobilitás

Java Virtual Machine Specification

Egy pillantás a nyelvre

```
class HelloWorld {
    public static void main( String[] args ){
        System.out.println("Hello world!");
    }
}
```

Objektumelvű programozás

Object-oriented programming (OOP)

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

Egységbezárás: 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 );
```

Osztály

Objektumok típusa

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

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

Példányosítás (instantiation)

- Objektum létrehozása osztály alapján
- Javában: mindig a heapen

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

Java programok felépítése

(első blikkre)

- [modul (`module`)]
- csomag (package)
- osztály (`class`)
- tag (member)
  - adattag (mező, field)
  - metódus (method)

Java forrásfájl

- Osztálynévvel
- `.java` kiterjesztés
- Fordítási egység
- Csomagjának megfelelő könyvtárban
- Karakterkódolás

Fordítás, futtatás

- A „tárgykód” a JVM bájtkód (`.class`)
- Nem szerkesztjük statikusan
- Futtatás: bájtkód interpretálása + JIT

Parancssorban

```
$ ls
HelloWorld.java
$ javac HelloWorld.java
$ ls
HelloWorld.class  HelloWorld.java
$ java HelloWorld
Hello world!
$
```

Java programok futása

- Végrehajtási verem (execution stack)
  - Aktivációs rekordok
  - Lokális változók
  - Paraméterátadás
- Dinamikus tárhely (heap)
  - Objektumok tárolása

Outline

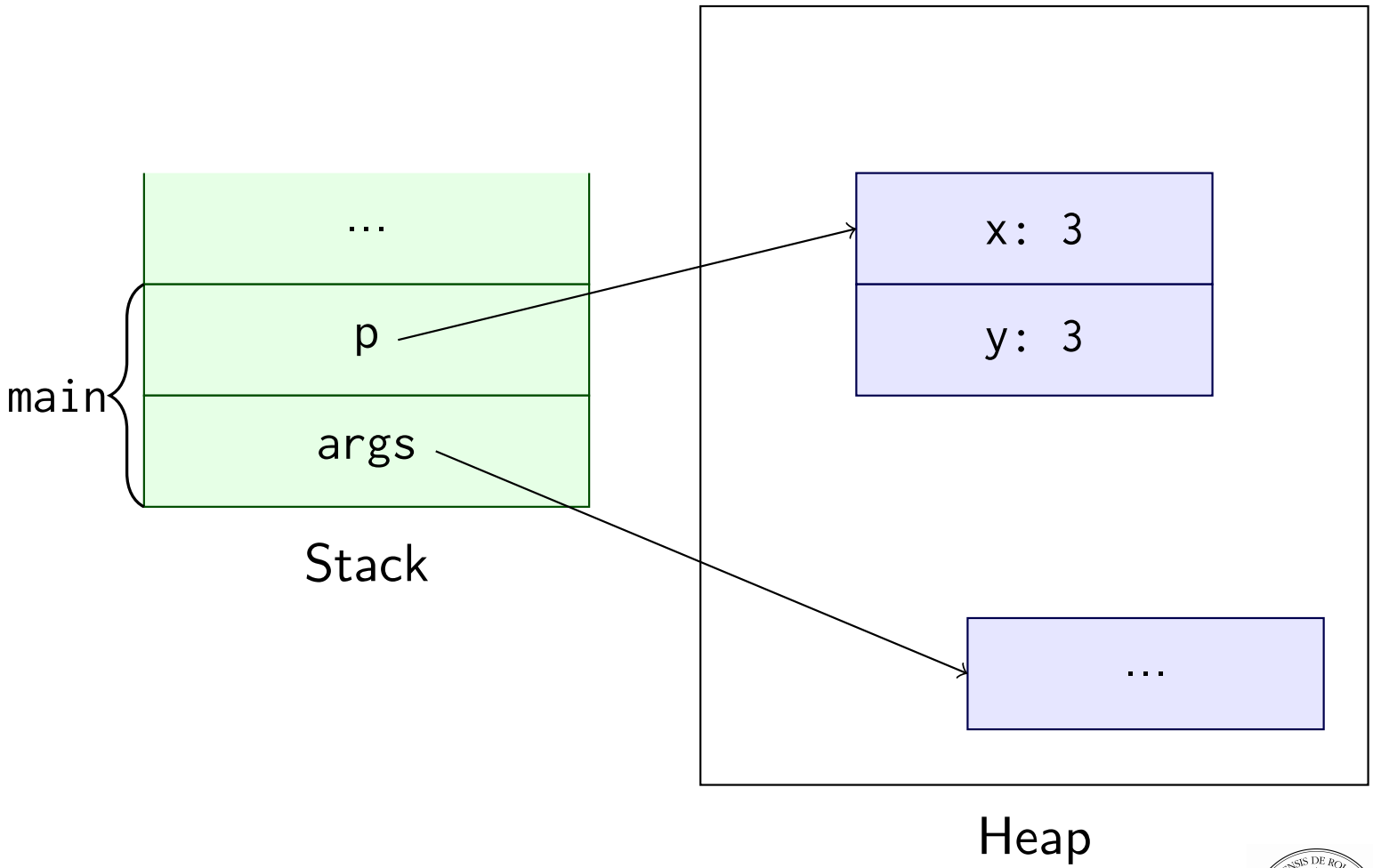
Osztály, objektum, példányosítás

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

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
$
```

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
    }
}
```

Mező alapértelmezett inicializációja

Automatikusan egy nulla-szerű értékre!

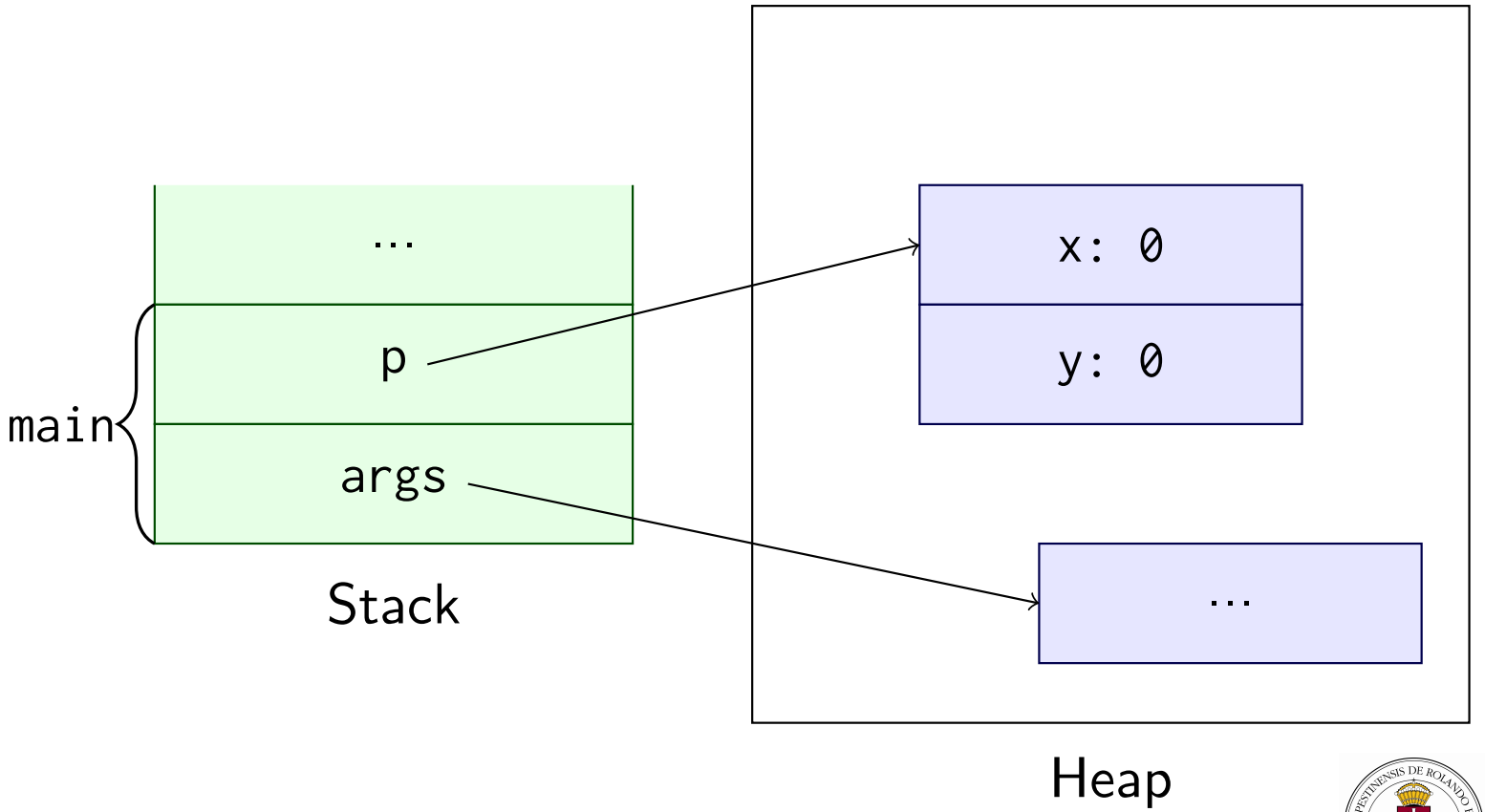
```
class Point {
    int x, y = 3;
}

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

Metódus

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

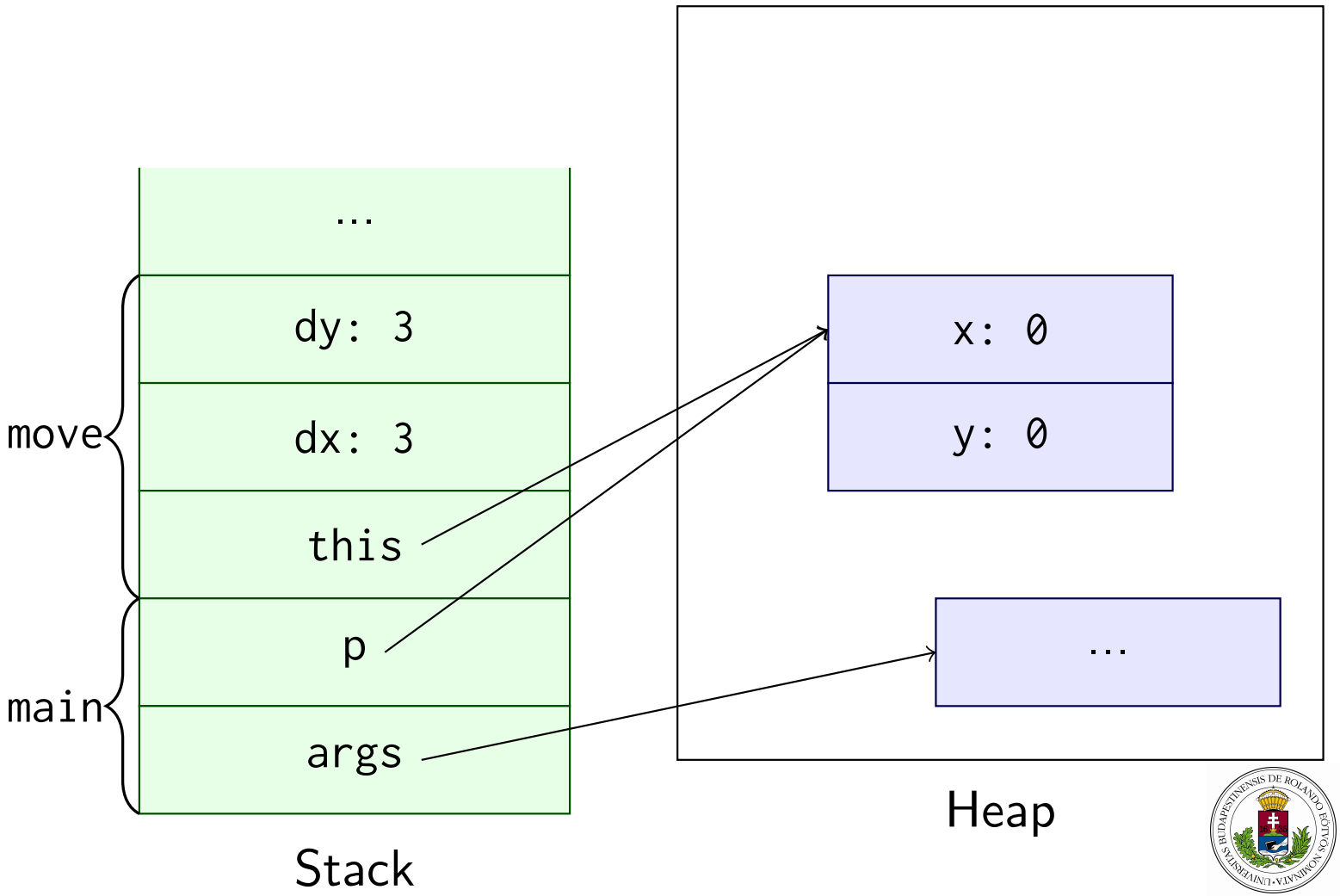
Metódus aktivációs rekordja – 1

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



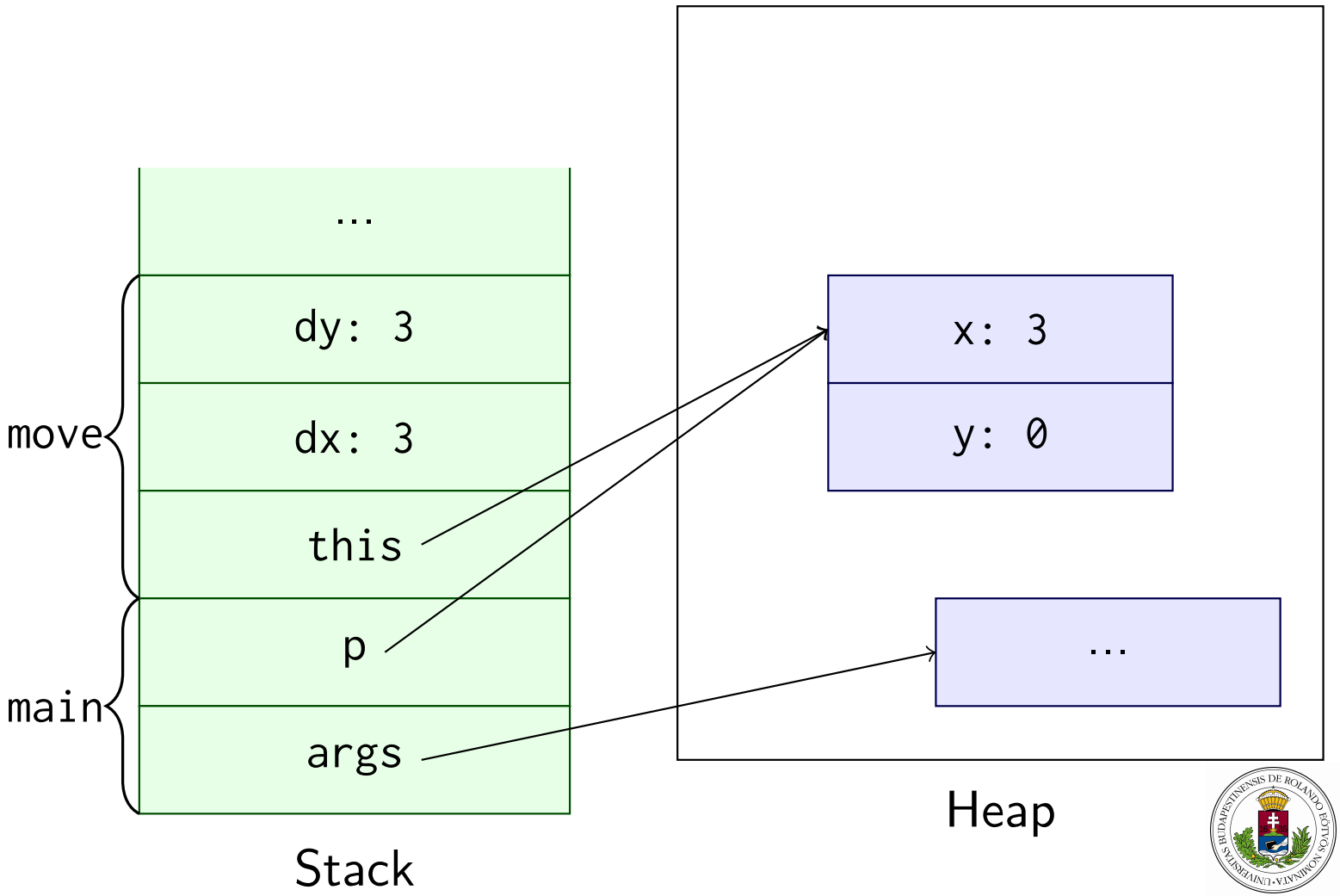
Metódus aktivációs rekordja – 2

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



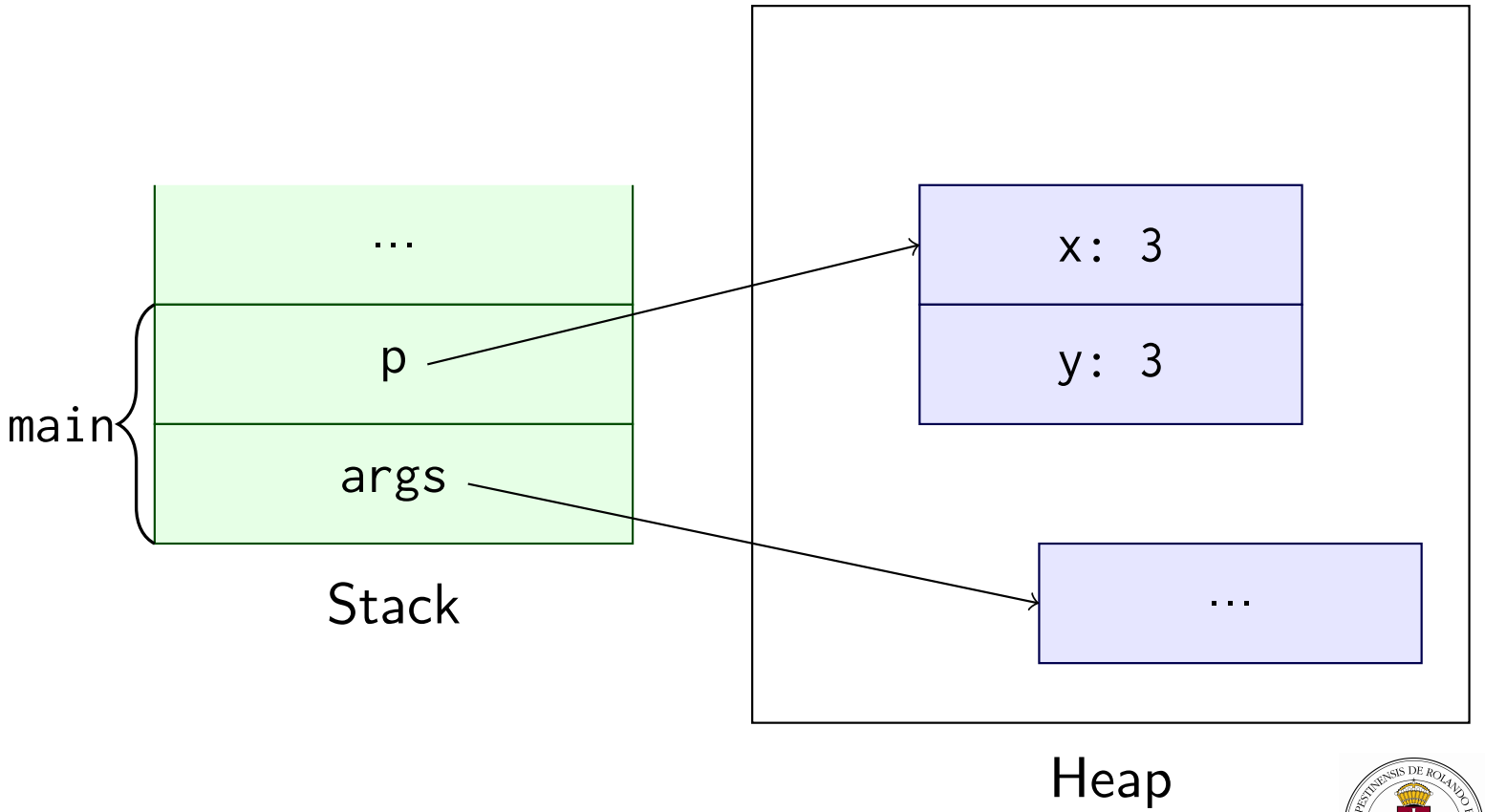
Metódus aktivációs rekordja – 3

```
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);
    }
}
```

Inicializálás konstruktorral

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

Inicializálás konstruktorral – a this elhagyható

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

Nevek újrahasznosítása

```
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
    }
}
```

Paraméter nélküli konstruktor

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

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

Alapértelmezett (default) konstruktor

```
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
    }
}
```

Generálodik egy paraméter nélküli, üres konstruktor

```
Pont(){}
```

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 { ... }
    } // (C) Monty Python
}
```

```
Time morning = new Time(6,10);
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(6,10);
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(Main.java:3)
$
```

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(6,10);
        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(6,10);
        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 aMinutePassed(){ ... }
}
```

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 aMinutePassed(){ ... }
}
```

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(6,10);
        ...
    }
}
```

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(6,10);
        ...
    }
}
```

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

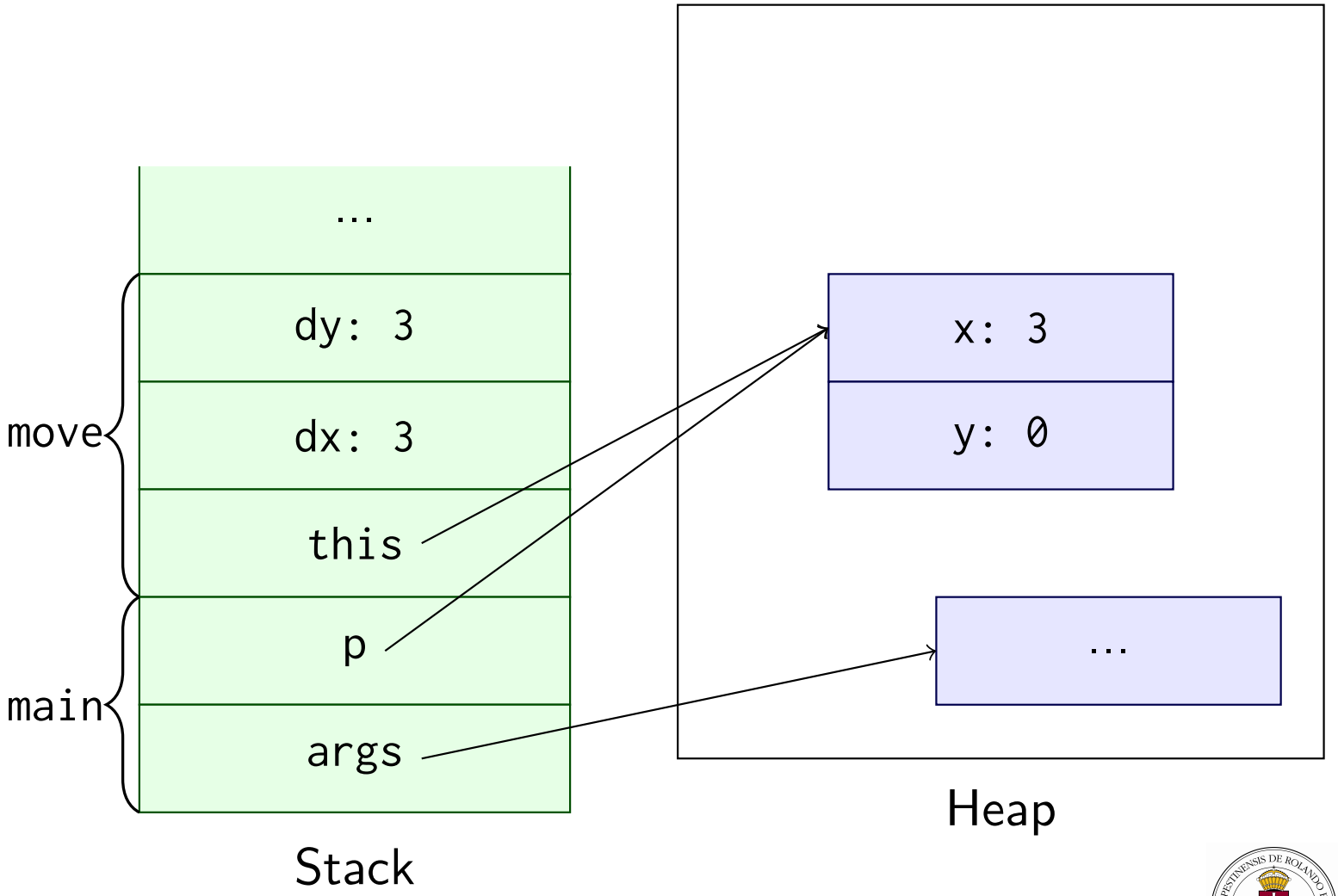
Outline

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
            ...
        }
    }
}
```

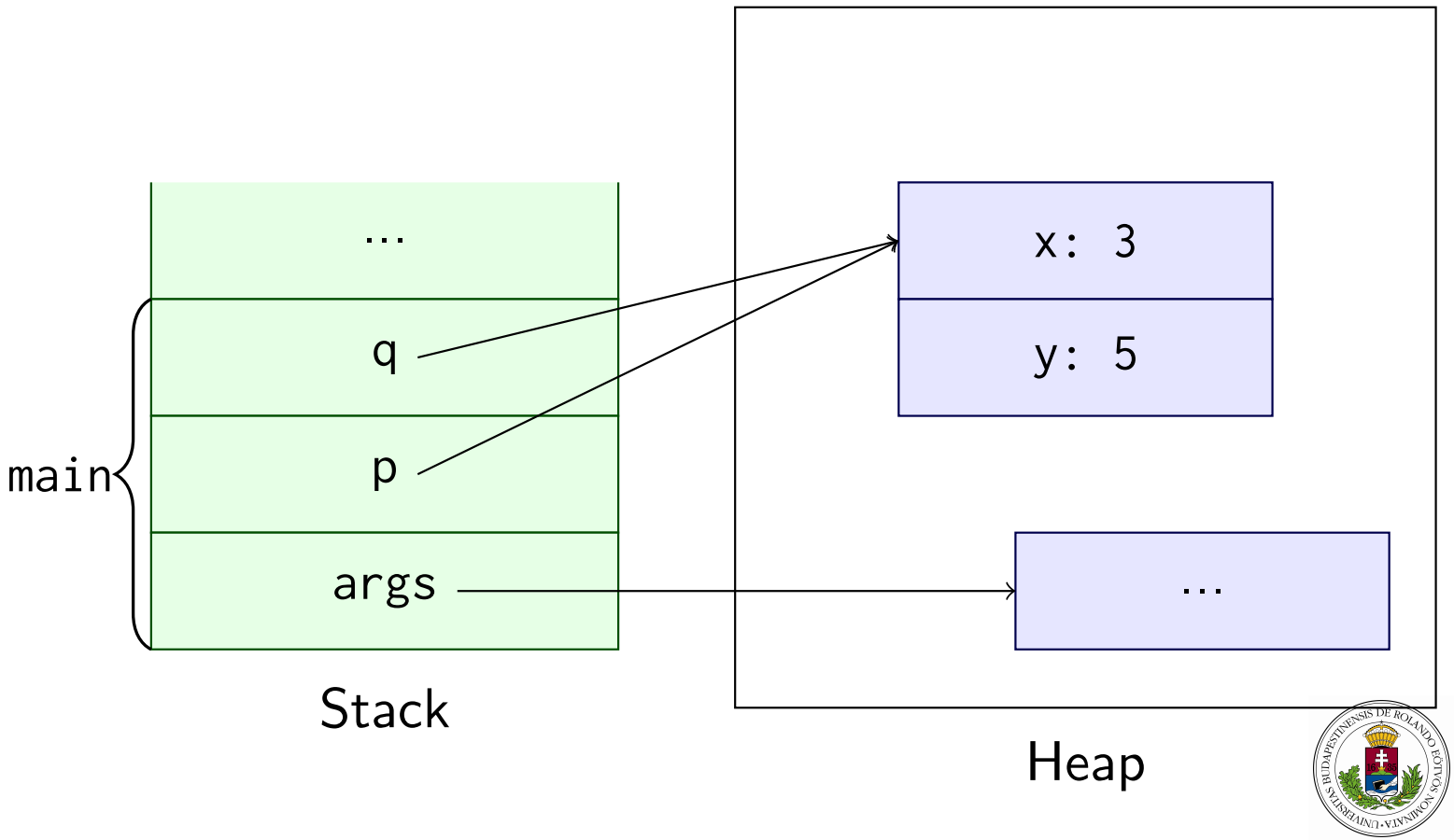
Objektumok élettartama

- 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

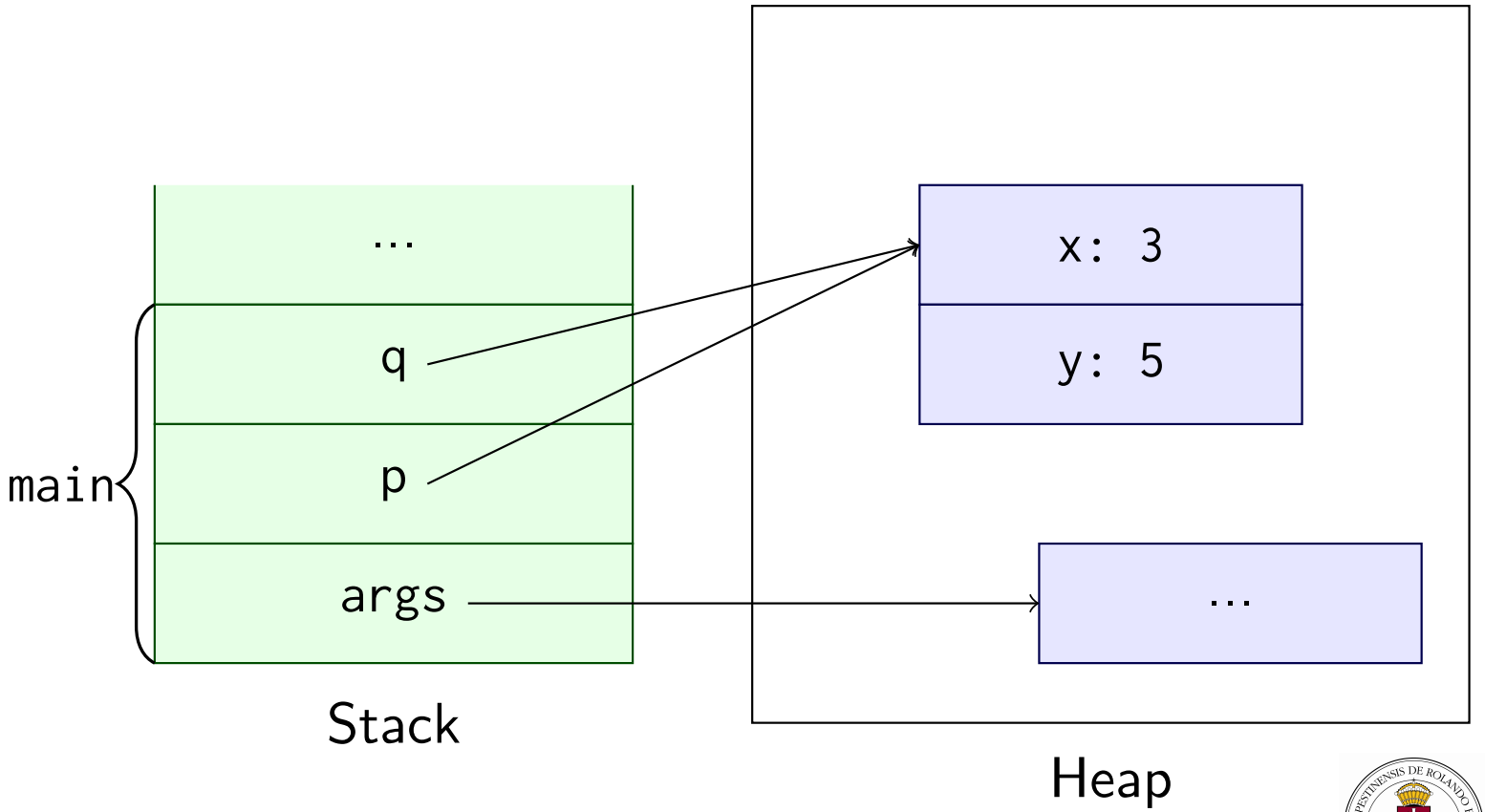
Csak olyat szabadít fel, amit már nem lehet elérni a programból

Teljes

Mindent felszabadít, amit nem lehet már elérni

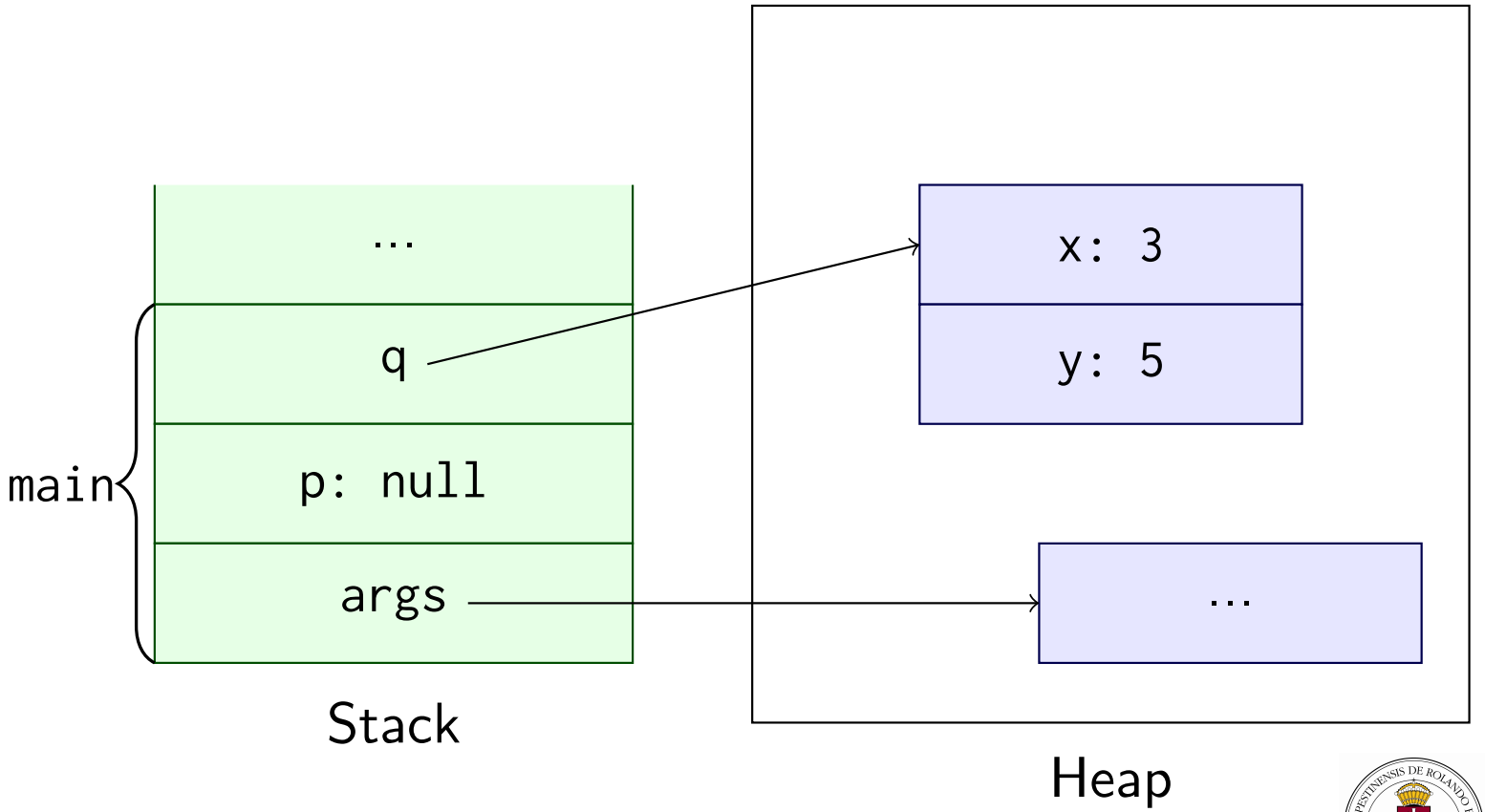
Még nem szabadítható fel

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



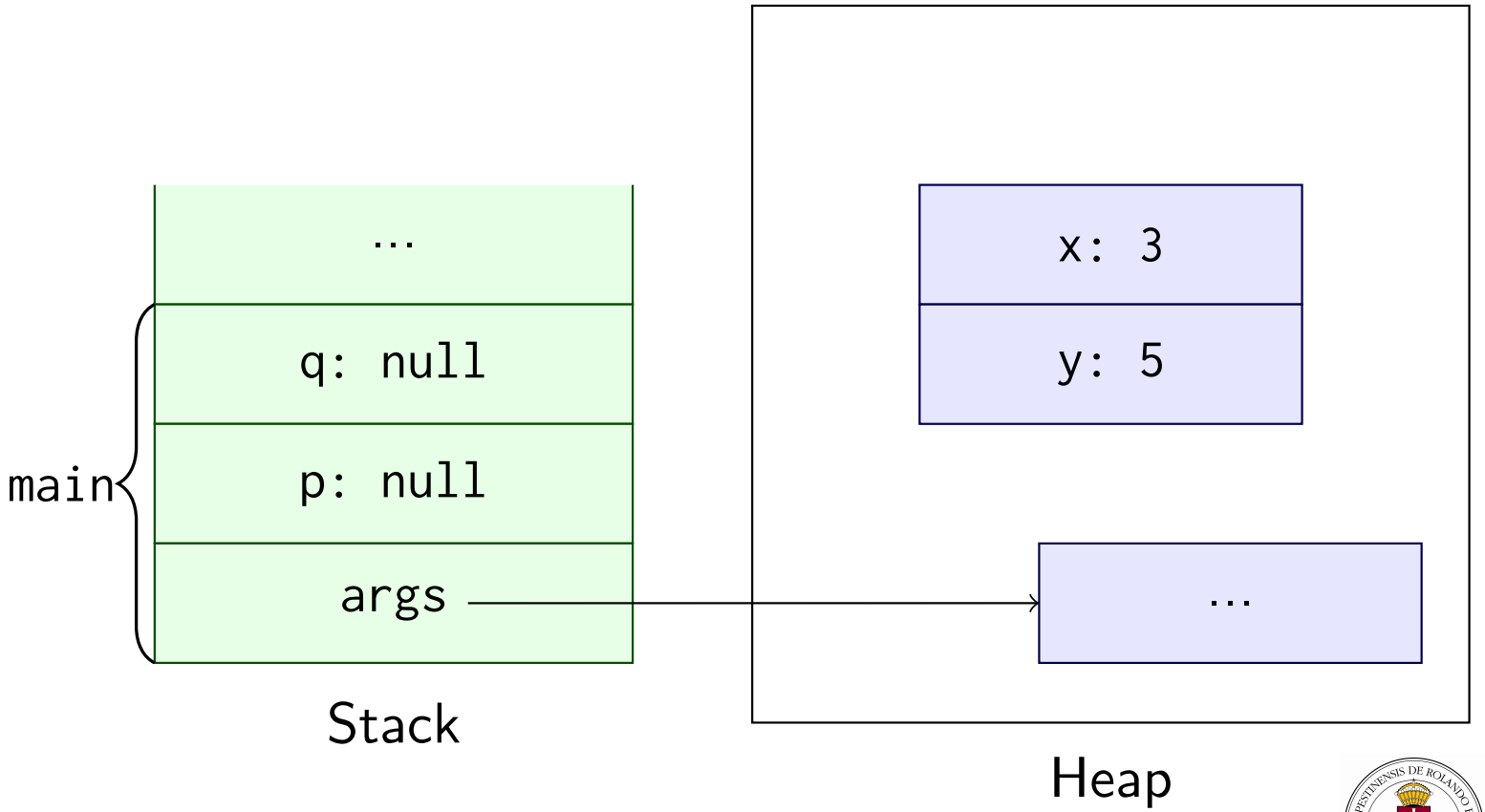
Még mindig nem szabadítható fel

```
p = null;
```



Már felszabadítható

```
q = null;
```

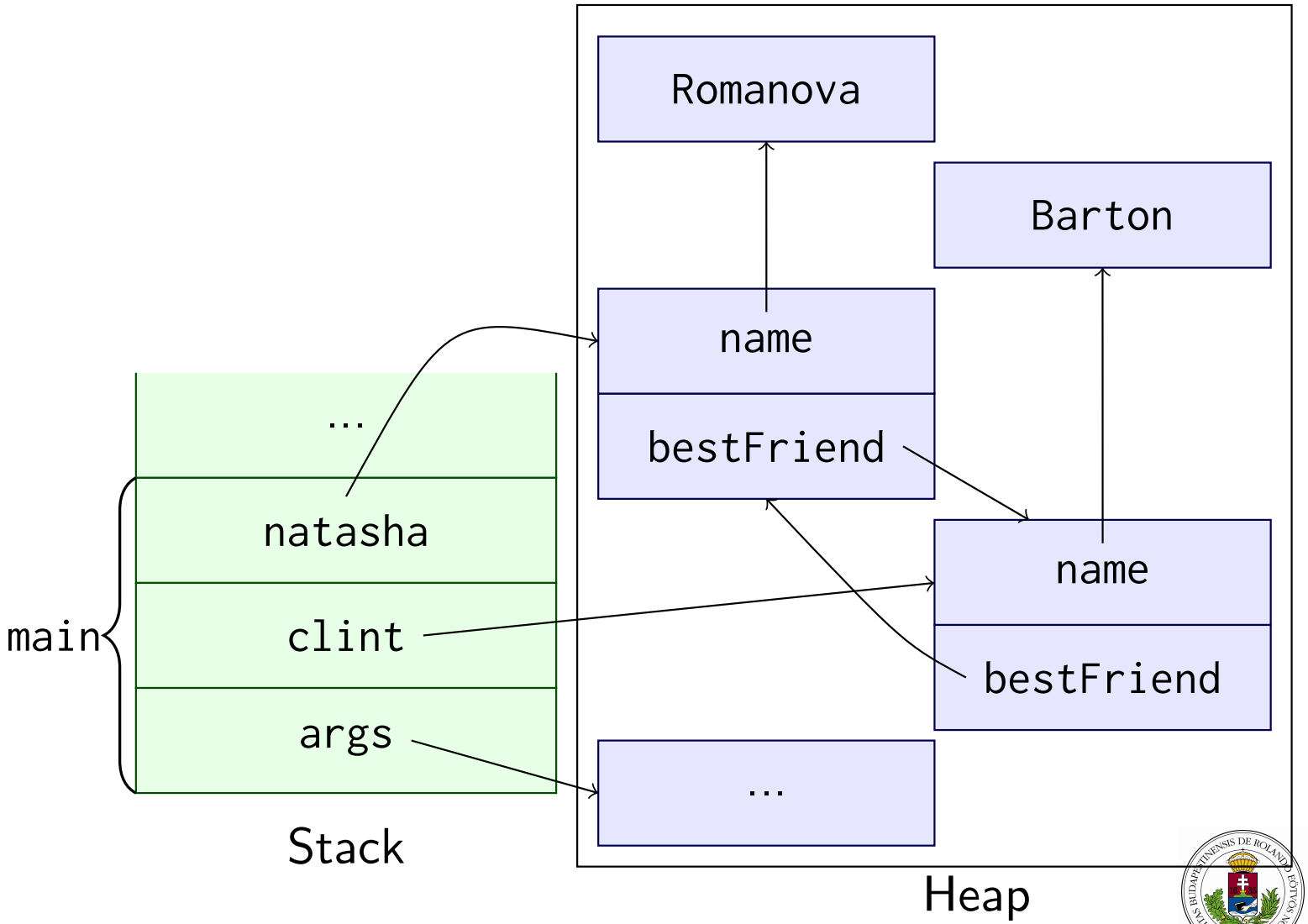


Bonyolultabb példa

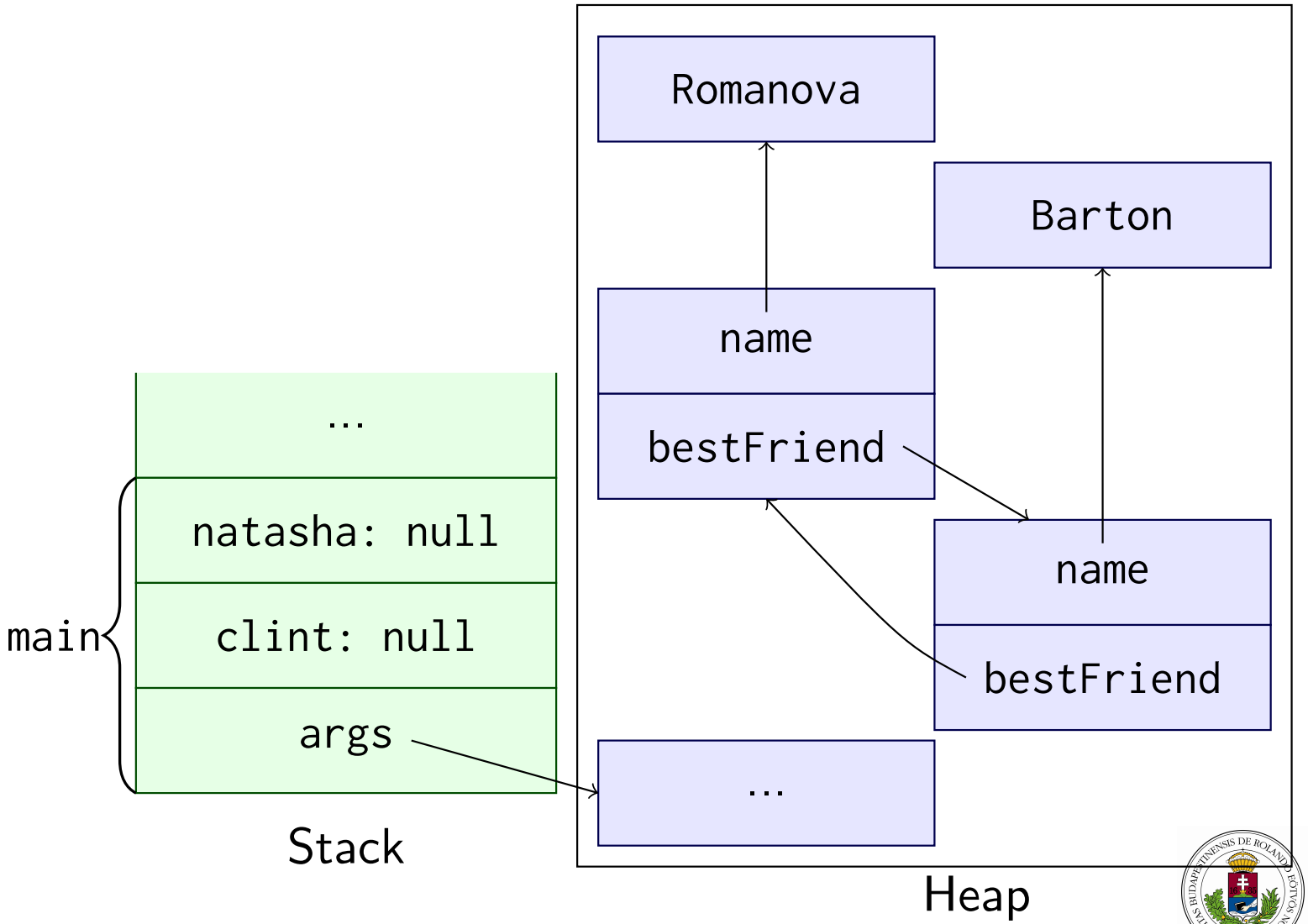
```
class Hero {
    String name;
    Hero bestFriend;
}
```

```
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

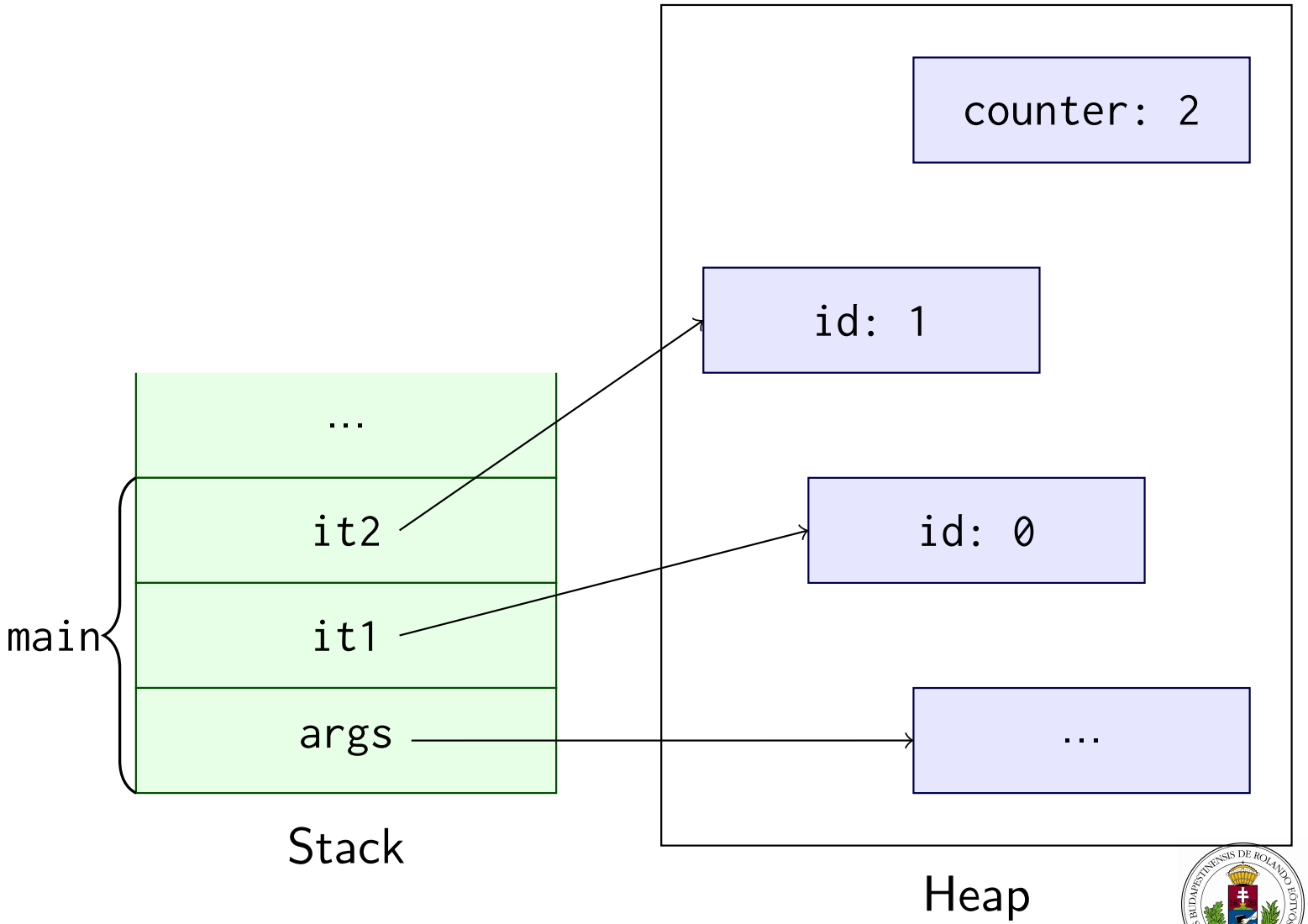
- Hasonló a C globális változóihoz
- Csak egy létezik belőle
- Az osztályon keresztül érhető el
- Mintha *statikus tárhelyen* lenne, nem az objektumokban

```
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, 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 );
    }
}
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);
}
```

Ismétlés

- Objektum-elvű programozás
  - Osztály és objektum
  - Egységbe zárás
  - Információ elrejtése
- Memóriakezelés
  - Referenciák
  - Végrehajtási verem és dinamikus tárhely
  - Példányszintű és osztályszintű tagok
  - Inicializáció
- Csomagok
- Fordítás és futtatás

Outline

Forráskód felépítése

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

Karakterek

Karakterkódolási szabványok (character encodings)

- Bacon's cipher, 1605 (Francis Bacon)
- Baude-code, 1874
- BCDIC, 1928 (Binary Coded Decimal Interchange Code)
- EBCDIC, 1963 (Extended ...)
- ASCII, 1963 (American Standard Code for Information Interchange)
- ISO/IEC 8859 (Latin-1, Latin-2,...)
- Windows 1250 (Cp1250)
- Unicode (UTF-8, UTF-16, UTF-32)

lásd: `iconv` (Unix/Linux)

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; zárójelezés
- kiértékelés
  - precedencia (`A + B * C`)
  - asszociativitás (`A - B - C`)
  - operandusok kiértékelési sorrendje (`A + B`)
  - lustaság (`A & B` és `A && B`)
  - mellékhatás (++x)

Lambda-kifejezések

```
int[] nats = {0, 1, 2, 3, 4, 5, 6};
```

```
int[] nats = new int[1000];
for( int i=0; i<nats.length; ++i ) nats[i] = i;
```

```
int[] nats = new int[1000];
java.util.Arrays.setAll(nats, i->i);

java.util.Arrays.setAll(nats, i->(int)(100*Math.random()));
```

Több paraméterrel

```
public static void main( String[] args ){
    java.util.Arrays.sort(args);
    java.util.Arrays.sort( args, (s,z) -> s.length()-z.length() );
}
```

Lehetőségek

```
i -> i

(int i) -> i+1

(int n, String s) -> {  StringBuilder sb = new StringBuilder();
                        for( int i=1; i<=n; ++i ) sb.append(s);
                        return sb.toString();
                     }
```

Funkcionális programozás

```
int[] nums = new int[1000];
java.util.Arrays.setAll(nums, i->(int)(100*Math.random()));

java.util.Arrays.stream(nums)
                .filter( i -> i%2 == 0 )
                .map( i -> i/2 )
                .limit(10)
                .forEach( i -> System.out.println(i) )
```

Részleges alkalmazás

```
java.util.Arrays.stream(nums)
                .forEach( i -> System.out.println(i) )
```

```
java.util.Arrays.stream(nums)
                .forEach( System.out::println )
```

Utasítások

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

Hagyományos switch-utasítás

```
String name;
switch( dayOf( new java.util.Date() ) ){
    case  0: name =    "Sunday"; break;
    case  1: name =    "Monday"; break;
    case  2: name =   "Tuesday"; break;
    case  3: name = "Wednesday"; break;
    case  4: name =  "Thursday"; break;
    case  5: name =    "Friday"; break;
    case  6: name =  "Saturday"; break;
    default: throw new Exception("illegal value");
}
```

Biztonságosabb switch-utasítás (JDK 12: preview)

```
String name;
switch( dayOf( new java.util.Date() ) ){
    case  0 -> name =    "Sunday";
    case  1 -> name =    "Monday";
    case  2 -> name =   "Tuesday";
    case  3 -> name = "Wednesday";
    case  4 -> name =  "Thursday";
    case  5 -> name =    "Friday";
    case  6 -> name =  "Saturday";
    default -> throw new Exception("illegal value");
}
```

switch-kifejezés (JDK 12: preview)

```
String name = switch( dayOf( new java.util.Date() ) ){
                    case  0 ->    "Sunday";
                    case  1 ->    "Monday";
                    case  2 ->   "Tuesday";
                    case  3 -> "Wednesday";
                    case  4 ->  "Thursday";
                    case  5 ->    "Friday";
                    case  6 ->  "Saturday";
                    default -> throw new Exception("illegal value");
                };
```

Túlcsorgás

```
switch(month){
  case  4:
  case  6:
  case  9:
  case 11: days = 30;
           break;
  case  2: days = 28 + leap;
           break;
  default: days = 31;
}
```

```
days = switch(month){
         case 4, 6,9, 11 -> 30;
         case 2  -> 28 + leap;
         default -> 31;
       };
```

Metódusok

- Végrehajtási verem, aktivációs rekord
- Paraméterátadás [!]
- Osztályszintű és példányszintű
- Hatókör (lokális változók), elfedés
- Láthatósági kategóriák
- Inicializáció: konstruktor
- Túlterhelés [!]
- Példánymetódusok felüldefiniálása [!!]

Típusdefiníciók

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

(egymásba ágyazás)

Fordítási egység

compilation unit

- opcionális package utasítás
- opcionális `import` utasítások
- típusdefiníciók
  - legfeljebb egy publikus

Az import utasítás

- Teljes név helyett rövid név
- Más, mint az #include a C-ben!

Típusnév importálása

```
import java.io.FileReader;
...
FileReader f;
```

Minden típusnév egy csomagból

```
import java.io.*;
...
FileReader in;
FileWriter out;
```

Statikus tagok importálása

```
import static java.util.Arrays.sort;
class Main {
    public static void main( String[] args ){
        sort(args);
        for( String s: args ){
            System.out.println(s);
        }
    }
}
```

Outline

Hiba detektálása és jelzése

```
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 ){
        if( 0 <= hour && hour <= 23 ){
            this.hour = hour;
        } else {
            throw new IllegalArgumentException("Invalid hour!");
        }
    }
    public void setMinute( int minute ){ ... }
    public void aMinutePassed(){ ... }
}
```

Az assert utasítás

```
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; }

    // may throw AssertionError
    public void setHour( int hour ){
        assert 0 <= hour && hour <= 23 ;
        this.hour = hour;
    }

    public void setMinute( int minute ){ ... }
    public void aMinutePassed(){ ... }
}
```

Az assert utasítás

TestTime.java

```
Time time = new Time(6,30);
time.setHour(30);
```

Futtatás

```
$ java TestTime
$ java -enableassertions TestTime
Exception in thread "main" java.lang.AssertionError
    at Time.setHour(Time.java:7)
    at TestTime.main(TestTime.java:5)
$
```

Dokumentációs megjegyzés

```
/** May throw AssertionError. */
public void setHour( int hour ){
    assert 0 <= hour && hour <= 23 ;
    this.hour = hour;
}
```

Dokumentált potenciálisan hibás használat

```
/**
  Blindly sets the hour property to the given value.
  Use it with care: only pass {@code hour} satisfying
  {@code 0 <= hour && hour <= 23}.
*/
public void setHour( int hour ){
    this.hour = hour;
}
```

javadoc Time.java

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Constructor Summary

Constructors

Constructor	Description
`Time()`	

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method	Description
int	`getHour()`	
int	`getMinute()`	
void	`oneMinutePassed()`	
void	`setHour(int hour)`	Blindly sets the hour property to the given value.

javadoc Time.java

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES SEARCH: Search

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

getHour

```
public int getHour()
```

getMinute

```
public int getMinute()
```

setHour

```
public void setHour(int hour)
```

Blindly sets the hour property to the given value. Use it with care: only pass `hour` satisfying `0 <= hour && hour <= 23`.

Szokásos (túl bőbeszédű) dokumentációs megjegyzés

```
/**
* Sets the hour property. Only pass an {@code hour}
* satisfying {@code 0 <= hour && hour <= 23}.
* @param hour The value to be set.
* @throws IllegalArgumentException
*     If the supplied value is not between 0 and 23,
*     inclusively.
*/
public void setHour( int hour ){
    if( 0 <= hour && hour <= 23 ){
        this.hour = hour;
    } else {
        throw new IllegalArgumentException("Invalid hour!");
    }
}
```

javadoc Time.java

setHour

```
public void setHour(int hour)
```

Sets the hour property. Only pass an `hour` satisfying `0 <= hour && hour <= 23`.

Parameters:

`hour` - The value to be set.

Throws:

`java.lang.IllegalArgumentException` - If the supplied value is not between 0 and 23, inclusively.

Szintaxiskiemelés

```java
/**
* Sets the hour property. Only pass an {@code hour}
* satisfying {@code 0 <= hour && hour <= 23}.
* @param hour The value to be set.
* @throws IllegalArgumentException
*     If the supplied value is not between 0 and 23,
*     inclusively.
*/
public void setHour( int hour ){
    if( 0 <= hour && hour <= 23 ){
        this.hour = hour;
    } else {
        throw new IllegalArgumentException("Invalid hour!");
    }
}
```

21,1

Opciók hibák jelzésére

Jó megoldások

- `IllegalArgumentException`: modul határán
- `assert`: modul belsejében
- Dokumentációs megjegyzés

Rossz megoldások

- Csendben elszabotálni a műveletet
- Elsumákolni az ellenőrzéseket

Ellenőrzött kivételek

checked exceptions

```
public Time readTime( String fname ) throws java.io.IOException {
    ...
}
```

- A programszövegben jelölni kell a terjedését
- A fordítóprogram ellenőrzi a konzisztenciát
- Ilyen: `java.sql.SQLException`, `java.security.KeyException`
- Nem ilyen: `NullPointerException`, `ArrayIndexOutOfBoundsException`

Terjedés követése: fordítási hiba

```
import java.io.IOException;
class TestTime {
    public Time readTime( String fname ) throws IOException {
        ... new java.io.FileReader(fname) ...
    }

    public static void main( String[] args ){
        TestTime tt = new TestTime();
        Time wakeUp = tt.readTime("wakeup.txt");
        wakeUp.aMinutePassed();
    }
}
```

Terjedés követése: fordítási hiba javítva

```
import java.io.IOException;
class TestTime {
    public Time readTime( String fname ) throws IOException {
        ... new java.io.FileReader(fname) ...
    }

    public static void main( String[] args ) throws IOException {
        TestTime tt = new TestTime();
        Time wakeUp = tt.readTime("wakeup.txt");
        wakeUp.aMinutePassed();
    }
}
```

Kivételkezelés

```
import java.io.IOException;
class TestTime {
    public Time readTime( String fname ) throws IOException {
        ... new java.io.FileReader(fname) ...
    }

    public static void main( String[] args ){
        TestTime tt = new TestTime();
        try {
            Time wakeUp = tt.readTime("wakeup.txt");
            wakeUp.aMinutePassed();
        } catch( IOException e ){
            System.err.println("Could not read wake-up time.");
        }
    }
}
```

A program tovább futhat a probléma ellenére

```
public class Receptionist {
    ...
    public Time[] readWakeupTimes( String[] fnames ){
        Time[] times = new Time[fnames.length];
        for( int i = 0; i < fnames.length; ++i ){
            try {
                times[i] = readTime(fnames[i]);
            } catch( java.io.IOException e ){
                times[i] = null;   // no-op
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return times; // maybe sort times before returning?
    }
    ...
}
```

A try-catch utasítás

```
<try-catch-statement> ::= try <block-statement>
                          <catch-list>
                          <optional-finally-part>

<catch-list> ::= <catch-part>
               | <catch-part> <catch-list>

<catch-part> ::= catch (<exceptions> <identifier>)
                       <block-statement>

<exceptions> ::= <identifier>
               | <identifier> | <exceptions>

<optional-finally-part> ::= ""
                          | finally <block-statement>
```

Több catch-ág

```
public static Time parse( String str ){
    String errorMessage;
    try {
        String[] parts = str.split(":");
        int hour = Integer.parseInt(parts[0]);
        int minute = Integer.parseInt(parts[1]);
        return new Time(hour,minute);
    } catch( NullPointerException e ){
        errorMessage = "Null parameter is not allowed!";
    } catch( ArrayIndexOutOfBoundsException e ){
        errorMessage = "String must contain \":\"!";
    } catch( NumberFormatException e ){
        errorMessage = "String must contain two numbers!";
    }
    throw new IllegalArgumentException(errorMessage);
}
```

Egy catch-ágban több kivétel

```
public static Time parse( String str ){
    try {
        String[] parts = str.split(":");
        int hour = Integer.parseInt(parts[0]);
        int minute = Integer.parseInt(parts[1]);
        return new Time(hour,minute);
    } catch( NullPointerException
           | ArrayIndexOutOfBoundsException
           | NumberFormatException e ){
        throw new IllegalArgumentException("Can't parse time!");
    }
}
```

A try-finally utasítás

```
public static Time readTime( String fname ) throws IOException {
    BufferedReader in = new BufferedReader(new FileReader(fname));
    Time time;
    try {
        String line = in.readLine();
        time = parse(line);
    } finally {
        in.close();
    }
    return time;
}
```

A finally mindenképp vezérlést kap!

```
public static Time readTime( String fname ) throws IOException {
    BufferedReader in = new BufferedReader(new FileReader(fname));
    try {
        String line = in.readLine();
        return parse(line);
    } finally {
        in.close();
    }
}
```

A try-catch-finally utasítás

```
public static Time readTime( String fname ) throws IOException {
    BufferedReader in = new BufferedReader(new FileReader(fname));
    try {
        String line = in.readLine();
        return parse(line);
    } catch ( IllegalArgumentException e ){
        System.err.println(e);
        System.err.println("Using default value!");
        return new Time(0,0);
    } finally {
        in.close();
    }
}
```

A try-utasítások egymásba ágyazhatók

```
public static Time readTimeOrUseDefault( String fname ){
    try {
        BufferedReader in =
                    new BufferedReader(new FileReader(fname));
        try {
            String line = in.readLine();
            return parse(line);
        } finally {
            in.close();
        }
    } catch( IOException | IllegalArgumentException e ){
        System.err.println(e);
        System.err.println("Using default value!");
        return new Time(0,0);
    }
}
```

A *try-with-resources* utasítás

```
public static Time readTimeOrUseDefault( String fname ){
    try {
        try(
            BufferedReader in =
                    new BufferedReader(new FileReader(fname))
        ){
            String line = in.readLine();
            return parse(line);
        }
    } catch( IOException | IllegalArgumentException e ){
        System.err.println(e);
        System.err.println("Using default value!");
        return new Time(0,0);
    }
}
```

Lényegében ekvivalensek

try-finally

```
BufferedReader in = ... ;
try {
    String line = in.readLine();
    return parse(line);
} finally {
    in.close();
}
```

try-with-resources

```
try(
    BufferedReader in = ...
){
    String line = in.readLine();
    return parse(line);
}
```

Bonyolultabb eset: fájl másolása

```
static void copy( String in, String out ) throws IOException {
    try (
        FileInputStream infile = new FileInputStream(in);
        FileOutputStream outfile = new FileOutputStream(out)
    ){
        int b;
        while( (b = infile.read()) != -1 ){    // idióma!
            outfile.write(b);
        }
    }
}
```

Outline

Racionális számok

```
package numbers;
public class Rational {

    private int numerator, denominator;
    /* class invariant: denominator > 0 */

    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

}
```

Getter-setter

```
package numbers;
public class Rational {

    private int numerator, denominator;

    public Rational( int numerator, int denominator ){ ... }

    public void setDenominator( int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.denominator = denominator;
    }

    public int getDenominator(){ return denominator; }

    ...
}
```

Aritmetika

```
package numbers;
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public void setNumerator( int numerator ){ ... }
    public void setDenominator( int denominator ){ ... }

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }
    ...
}
```

Dokumentációs megjegyzéssel

```
package numbers;
public class Rational {
    ...
    /**
    *  Set {@code this} to {@code this} * {@code that}.
    *  @param that Non-null reference to a rational number,
    *              it will not be changed in the method.
    *  @throws NullPointerException When {@code that} is null.
    */
    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }
    ...
}
```

Főprogram

```
import numbers.Rational.*;
public class Main {
    public static void main( String[] args ){
        Rational p = new Rational(1,3);
        Rational q = new Rational(1,2);
        p.multiplyWith(q);
        println(p);
        println(q);
    }
    private static void println( Rational r ){
        System.out.println( r.getNumerator() + "/" +
                            r.getDenominator() );
    }
}
```

Műveletek sorozása

```
package numbers;
public class Rational {
    ...
    public Rational multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
        return this;
    }
    ...
}
```

```
Rational p = new Rational(1,3);
Rational q = new Rational(1,2);
p.multiplyWith(q).multiplyWith(q).divideBy(q);
println(p);
```

Teljesen másfajta megoldás

```
package numbers;
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    ...

    public Rational times( Rational that ){
        return new Rational( this.numerator * that.numerator,
                             this.denominator * that.denominator );
    }

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }
}
```

Használjuk mindkettőt

```
package numbers;
public class Rational {
    ...
    public void multiplyWith( Rational that ){ ... }
    public Rational times( Rational that ){ ... }
}
```

```
Rational p = new Rational(1,3);
Rational q = new Rational(1,2);
p.multiplyWith(q);
println(p);
Rational r = p.times(q);
println(r);
println(p);
```

Funkcionális stílus

```
package numbers;
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
    public Rational  plus( Rational that ){ ... }
    ...
}
```

Módosíthatatlan mezőkkel

```
package numbers;
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
    public Rational  plus( Rational that ){ ... }
    ...
}
```

Módosíthatatlan változó

```
final int width = 80;
```

- Ha egyszer értéket kapott, nem adhatunk új értéket neki
- A deklarációban értéket kell kapjon
- Hasonló a C-beli const-hoz (de nem pont ugyanaz)

lokális változó, formális paraméter

Globális konstans

```
public static final int WIDTH = 80;
```

- Osztályszintű mező
- Picit olyan, mint a C-ben egy #define
- Konvenció: végig nagy betűvel írjuk a nevét

Módosíthatatlan mező

- Például `WIDTH` globális konstans
- Vagy `Rational` két mezője
- Ha egyszer értéket kapott, nem adhatunk új értéket neki
- Inicializáció során értéket kell kapjon
  - „Üres konstans” (blank final)!

Mutable versus Immutable

Módosítható belső állapot (OOP)

```
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }     ...
    public void setNumerator( int numerator ){ ... }   ...
    public void multiplyWith( Rational that ){ ... }
```

Módosíthatatlan belső állapot (FP)

```
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
```

Nyilvános módosíthatatlan belső állapot

```
public class Rational {
    public final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public Rational times( Rational that ){ ... }
    ...
}
```

Érzékeny a reprezentációváltoztatásra!

Más elnevezési konvenció

```
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int numerator(){ return numerator; }
    public int denominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
}
```

```
System.out.println( p.numerator() + "/" + p.denominator() );
```

Reprezentációváltás

```
public class Rational {
    private final int[] data;
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        data = new int[]{ numerator, denominator };
    }
    public int numerator(){ return data[0]; }
    public int denominator(){ return data[1]; }
    public Rational times( Rational that ){ ... }
}
```

final referencia

```
final int[] data = new int[2];
data[0] = 3;
data[0] = 4;
data = new int[3]; // fordítási hiba
```

Karaktersorozatok ábrázolása

- java.lang.String: módosíthatatlan (immutable)

```
String fourtytwo = "42";
String twentyfour = fourtytwo.reverse();
String twentyfourhundredfourtytwo = twentyfour + fourtytwo;
```

- java.lang.StringBuilder és java.lang.StringBuffer: módosítható

```
StringBuilder sb = new StringBuilder("");
for( char c = 'a'; c <= 'z'; ++c ){
   sb.append(c).append(',');
}
sb.deleteCharAt(sb.length()-1);    // remove last comma
String letters = sb.toString();
```

- char[]: módosítható

Hatékonyságbeli kérdés

```
StringBuilder sb = new StringBuilder("");    // temporary
for( char c = 'a'; c <= 'z'; ++c ){
   sb.append(c).append(',');
}
sb.deleteCharAt(sb.length()-1);
String letters = sb.toString();
```

```
String letters = "";
for( char c = 'a'; c <= 'z'; ++c ){
   letters += (c + ',');
}
letters = letters.substring(0,letters.length()-1);
```

Procedurális stílus (függvény)

```
public class Rational {
    private final int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int numerator(){ return numerator; }
    public int denominator(){ return denominator; }

    public static Rational times( Rational left, Rational right ){
        return new Rational( left.numerator * right.numerator,
                             left.denominator * right.denominator );
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
Rational r = Rational.times(p,q);
```

Procedurális stílus (eljárás)

```
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public void setNumerator( int numerator ){ ... }
    ...
    public static void multiplyLeftWithRight( Rational left,
                                              Rational right ){
        left.numerator *= right.numerator;
        left.denominator *= right.denominator;
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
Rational.multiplyLeftWithRight(p,q);
```

Paraméterátadás Javában

Érték szerinti (call-by-name)

- primitív típusú paraméterre

```
public void setNumerator( int numerator ){
    this.numerator = numerator;
}
```

Megosztás szerinti (call-by-sharing)

- referencia típusú paraméterre
- a referenciát érték szerint adjuk át

```
public static void multiplyLeftWithRight( Rational left,
                                          Rational right ){
    left.numerator *= right.numerator;
    left.denominator *= right.denominator;
}
```

Érték szerinti (call-by-name)

```
public void setNumerator( int numerator ){
    this.numerator = numerator;
    numerator = 0;
}
```

```
Rational p = new Rational(1,3);
int two = 2;
p.setNumerator(two);
println(p);
System.out.println(two);
```

Megosztás szerinti (call-by-sharing)

```
public static void multiplyLeftWithRight( Rational left,
                                          Rational right ){
    left.numerator *= right.numerator;
    left.denominator *= right.denominator;
    left = new Rational(9,7);
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
Rational.multiplyLeftWithRight(p,q);
println(p);
```

Ha a paraméterek nem diszjunktak...

```
package numbers;
public class Rational {
    ...

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }

    public void divideBy( Rational that ){
        if( that.numerator == 0 )
            throw new ArithmeticException("Division by zero!");
        this.numerator *= that.denominator;
        this.denominator *= that.numerator;
    }
}
```

Belső állapot kiszivárgása

```
public class Rational {
    private int[] data;
    ...
    public int getNumerator(){ return data[0]; }
    public int getDenominator(){ return data[1]; }
    public void set( int[] data ){
        if( data == null || data.length != 2 || data[1] <= 0 )
            throw new IllegalArgumentException();
        this.data = data;
    }
}
```

```
Rational p = new Rational(1,2);
int[] cheat = {3,4};
p.set(cheat);
cheat[1] = 0;       // p.getDenominator() == 0   :-(
```

Belső állapot kiszivárgása ügyetlen konstruálás miatt

```
public class Rational {
    private final int[] data;
    public int getNumerator(){ return data[0]; }
    public int getDenominator(){ return data[1]; }
    public Rational( int[] data ){
        if( data == null || data.length != 2 || data[1] <= 0 )
            throw new IllegalArgumentException();
        this.data = data;
    }
}
```

```
int[] cheat = {3,4};
Rational p = new Rational(cheat);
cheat[1] = 0;       // p.getDenominator() == 0   :-(
```

Belső állapot kiszivárgása getteren keresztül

```
public class Rational {
    private final int[] data;
    ...
    public int getNumerator(){ return data[0]; }
    public int getDenominator(){ return data[1]; }
    public int[] get(){ return data; }
}
```

```
Rational p = new Rational(1,2);
int[] cheat = p.get();
cheat[1] = 0;       // p.getDenominator() == 0   :-(
```

Defenzív másolás

```
public class Rational {
    private final int[] data;
    public Rational( int[] data ){
        if( data == null || data.length != 2 || data[1] <= 0 )
            throw new IllegalArgumentException();
        this.data = new int[]{ data[0], data[1] };
    }
    public void set( int[] data ){ /* hasonlóan */ }
    public int[] get(){
        return new int[]{ data[0], data[1] };
    }
}
```

Módosíthatatlan objektumokat nem kell másolni

```
public class Person {
    private String name;
    private int age;
    public Person( String name, int age ){
        if( name == null || name.trim().isEmpty() || age < 0 )
            throw new IllegalArgumentException();
        this.data = name;
        this.age = age;
    }
    public String getName(){ return name; }
    public int getAge(){ return age; }
    public void setName( String name ){ ... this.name = name; }
    public void setAge( int age ){ ... this.age = age; }
}
```

Tömbelemek között is lehet aliasing

```
Rational rats[2]; // fordítási hiba

Rational rats[] = new Rational[2]; // = {null,null};

Rational[] rats = new Rational[2]; // gyakoribb
rats[0] = new Rational(1,2);
rats[1] = rats[0];
rats[1].setDenominator(3);
System.out.println(rats[0].getDenominator());
```

- módosítható versus módosíthatatlan

Ugyanaz az objektum többször is lehet a tömbben

```
/**
    ...
    PRE: rats != null
    ...
*/
public static void increaseAllByOne( Rational[] rats ){
    for( Rational p: rats ){
        p.setNumerator( p.getNumerator() + p.getDenominator() );
    }
}
```

Dokumentálva

```
/**
    ...
    PRE: rats != null and (i!=j => rats[i] != rats[j])
    ...
*/
public static void increaseAllByOne( Rational[] rats ){
    for( Rational p: rats ){
        p.setNumerator( p.getNumerator() + p.getDenominator() );
    }
}
```

Tömbök tömbje

- Javában nincs többdimenziós tömb (sor- vagy oszlopfolytonos)
- Tömbök tömbje (referenciák tömbje)

```
int[][] matrix = {{1,0,0},{0,1,0},{0,0,1}};

int[][] matrix = new int[3][3];
for( int i=0; i<matrix.length; ++i ) matrix[i][i] = 1;

int[][] matrix = new int[5][];
for( int i=0; i<matrix.length; ++i ) matrix[i] = new int[i];
```

Ismét aliasing – bug-gyanús

```
Rational[][] matrix = { {new Rational(1,2), new Rational(1,2)},
                        {new Rational(1,2), new Rational(1,2)},
                        {new Rational(1,2), new Rational(1,2)} };
```

```
Rational half = new Rational(1,2);
Rational[] halves = {half, half};
Rational[][] matrix = {halves, halves, halves};
```

Outline

Több metódus ugyanazzal a névvel

```
public class Rational {
    ...

    public void multiplyWith( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
    }

    public void multiplyWith( int that ){
        this.numerator *= that.numerator;
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(1,2);
p.multiplyWith(q);
p.multiplyWith(2);
```

Több konstruktor ugyanabban az osztályban

```
public class Rational {
    ...

    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public Rational( int value ){
        numerator = value;
        denominator = 1;
    }
}
```

```
Rational p = new Rational(1,3), q = new Rational(3);
```

Túlterhelés

- Több metódus ugyanazzal a névvel, több konstruktor
- Formális paraméterek eltérnek
  - Paraméterek száma
  - Paraméterek deklarált típusa
- Híváskor a fordító eldönti, melyiket kell hívni
  - Az aktuális paraméterek száma,
  - illetve deklarált típusa alapján
- Fordítási hiba, ha:
  - Egyik sem felel meg a hívásnak
  - Ha több is egyformán megfelel

Konstruktorok egymást hívhatják

```
public class Rational {
    ...
    public Rational( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public Rational( int value ){
        this(value,1);
    }

    public Rational(){
        this(0);
    }
}
```

Alapértelmezett érték

```
public class Rational {
    ...
    public void set( int numerator, int denominator ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public void set( int value ){
        set(value,1);
    }

    public void set(){
        set(0);
    }
}
```

Outline

Egy korábbi példa

```
public class Receptionist {
    ...
    public Time[] readWakeupTimes( String[] fnames ){
        Time[] times = new Time[fnames.length];
        for( int i = 0; i < fnames.length; ++i ){
            try {
                times[i] = readTime(fnames[i]);
            } catch( java.io.IOException e ){
                times[i] = null;   // no-op
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return times; // maybe sort times before returning?
    }
}
```

A null értékek kiszűrése

```
public class Receptionist {
    ...
    public Time[] readWakeupTimes( String[] fnames ){
        Time[] times = new Time[fnames.length];
        int j = 0;
        for( int i = 0; i < fnames.length; ++i ){
            try {
                times[j] = readTime(fnames[i]);
                ++j;
            } catch( java.io.IOException e ){
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return java.util.Arrays.copyOf(times,j); // possibly sort
    }
}
```

Tömbök előnyei és hátrányai

- Elemek hatékony elérése (indexelés)
- Szintaktikus támogatás a nyelvben (indexelés, tömbliterál)
- Fix hossz: létrehozáskor
  - Bővítéshez új tömb létrehozása + másolás
  - Törléshez új tömb létrehozása + másolás

Alternatíva: java.util.ArrayList

kényelmes szabványos könyvtár, hasonló belső működés

```
String[] names = { "Tim",
                   "Jerry" };
```

```
ArrayList<String> names =
            new ArrayList<>();
names.add("Tim");
names.add("Jerry");
```

```
names[0] = "Tom";
String mouse = names[1];
```

```
names.set(0, "Tom");
String mouse = names.get(1);
```

```
String trio = new String[3];
trio[0] = names[0];
trio[1] = names[1];
trio[2] = "Spike";
names = trio;
```

```
names.add("Spike");
```

Az előző példa átalakítva

```
public class Receptionist {
    ...
    public ArrayList<Time> readWakeupTimes( String[] fnames ){
        ArrayList<Time> times = new ArrayList<Time>();
        for( int i = 0; i < fnames.length; ++i ){
            try {
                times.add( readTime(fnames[i]) );
            } catch( java.io.IOException e ){
                System.err.println("Could not read " + fnames[i]);
            }
        }
        return times; // possibly sort before returning
    }
}
```

Paraméterezett típus

```
ArrayList<Time> times
```

```
Time[] times
Time times[]
```

Paraméterezés típussal

```
length :: [a] -> Int
length (x:xs) = 1 + length xs
length [] = 0

          length [1..10] + length ["alma", "a", "fa", "alatt"]


reverse :: [a] -> [a]
reverse (x:xs) = reverse xs ++ [x]
reverse [] = []
```

Generikus osztály

Nem pont így, de hasonlóan...!

```
package java.util;
public class ArrayList<T> {
    public ArrayList(){ ... }
    public T get( int index ){ ... }
    public void set( int index, T item ){ ... }
    public void add( T item ){ ... }
    public T remove( int index ){ ... }
    ...
}
```

Használatkor típusparaméter megadása

```
import java.util.ArrayList;

...

ArrayList<Time> times;
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> names = new ArrayList<>();
```

Generikus metódus

```
import java.util.*;
class Main {
    public static <T> void reverse( T[] array ){
        int lo = 0, hi = array.length-1;
        while( lo < hi ){
            T tmp = array[hi];
            array[hi] = array[lo];
            array[lo] = tmp;
            ++lo; --hi;
        }
    }
    public static void main( String[] args ){
        reverse(args);
        System.out.println( Arrays.toString(args) );
    }
}
```

Parametrikus polimorfizmus

- Több típusra is működik ugyanaz a kód
  - Java: típus (osztály), metódus
- Típussal paraméterezhető kód
  - Java: referenciatípusokkal

Típusparaméter

Helytelen

```
ArrayList<int> numbers
```

Helyes

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add( Integer.valueOf(7) );
Integer seven = numbers.get(0);

numbers.add(42);
int fourtytwo = numbers.get(1);
```

Outline

Típuskonverziók primitív típusok között

Automatikus típuskonverzió (tranzitív)

- `byte` < `short` < `int` < `long`
- `long` < `float`
- `float` < `double`
- `char` < `int`
- `byte b = 42;` és `short s = 42;` és `char c = 42;`

Explicit típuskényszerítés (type cast)

```
int i = 42;
short s = (short)i;
```

Puzzle 3: Long Division (Bloch & Gafter: Java Puzzlers)

```
public class LongDivision {
    public static void main(String[] args) {
        final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;
        final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

Csomagoló osztályok

Implicit importált (`java.lang`), immutable osztályok

- `java.lang.Boolean` – `boolean`
- `java.lang.Character` – `char`
- `java.lang.Byte` – `byte`
- `java.lang.Short` – `short`
- `java.lang.Integer` – `int`
- `java.lang.Long` – `long`
- `java.lang.Float` – `float`
- `java.lang.Double` – `double`

java.lang.Integer interfésze (részlet)

```
static int MAX_VALUE    // 2^31-1
static int MIN_VALUE    // -2^31

static int compare( int x, int y )    // 3-way comparison
static int max( int x, int y )
static int min( int x, int y )
static int parseInt( String str [, int radix] )
static String toString( int i [, int radix] )
static Integer valueOf( int i )

int compareTo( Integer that )        // 3-way comparison
int intValue()
```

Auto-(un)boxing

- Automatikus kétirányú konverzió
- Primitív típus és a csomagoló osztálya között

```
Integer ref = 42;
int pri = ref;

Integer sum = ref + pri;
```

```
Integer ref = Integer.valueOf(42);
int pri = ref.intValue();

Integer sum = Integer.valueOf (
                ref.intValue()
                + pri           );
```

Auto-(un)boxing + generikusok

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(7);
int seven = numbers.get(0);
```

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add( Integer.valueOf(7) );
int seven = numbers.get(0).intValue();
```

Számolás egész számokkal

```
int n = 10;
int fact = 1;
while( n > 1 ){
    fact *= n;
    --n;
}
```

Rosszul használt auto-(un)boxing

```
Integer n = 10;
Integer fact = 1;
while( n > 1 ){
    fact *= n;
    --n;
}
```

Jelentés

```
Integer n = Integer.valueOf(10);
Integer fact = Integer.valueOf(1);
while( n.intValue() > 1 ){
    fact = Integer.valueOf(fact.intValue() * n.intValue());
    n = Integer.valueOf(n.intValue() - 1);
}
```

Outline

Absztrakció: egységbe zárás és információ elrejtése

```
public class Rational {
    private final int numerator, denominator;
    private static int gcd( int a, int b ){ ... }
    private void simplify(){ ... }
    public Rational( int numerator, int denominator ){ ... }
    public Rational( int value ){ super(value,1); }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
    public Rational times( int that ){ ... }
    public Rational plus( Rational that ){ ... }
    ...
}
```

Egy osztály interfésze

```
public Rational( int numerator, int denominator )
public Rational( int value )
public int getNumerator()
public int getDenominator()
public Rational times( Rational that )
public Rational times( int that )
public Rational plus( Rational that )
...
```

Az interface-definíció

```
public interface Rational {
    public int getNumerator();
    public int getDenominator();
    public Rational times( Rational that );
    public Rational times( int that );
    public Rational plus( Rational that );
    ...
}
```

interface: automatikusan publikusak a tagok

```
public interface Rational {
    int getNumerator();
    int getDenominator();
    Rational times( Rational that );
    Rational times( int that );
    Rational plus( Rational that );
    ...
}
```

Az interface-definíció tartalma

Példánymetódusok deklarációja: specifikáció és ;

```
int getNumerator();
```

Az `interface`-definíció tartalma, de tényleg

- Példánymetódusok deklarációja
  - Esetleg `default` implementáció
- Konstansok definíciója: `public static final`
- Statikus metódus
- Beágyazott (tag-) típus

Interface megvalósítása

Rational.java

```
public interface Rational {
    int getNumerator();
    int getDenominator();
    Rational times( Rational that );
}
```

Fraction.java

```
public class Fraction implements Rational {
    private final int numerator, denominator;
    public Fraction( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
}
```

Több megvalósítás

Simplified.java

```
public class Simplified implements Rational {
    ...
    public int getNumerator(){ ... }
    public int getDenominator(){ ... }
    Rational times( Rational that ){ ... }
}
```

Fraction.java

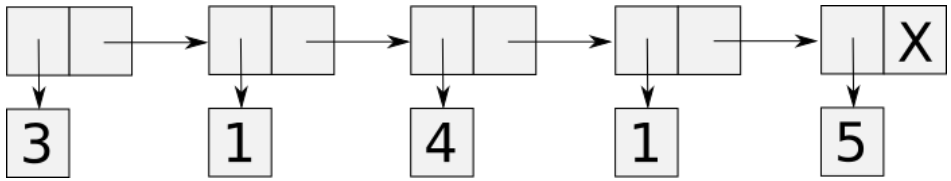
```
public class Fraction implements Rational {
    private final int numerator, denominator;
    public Fraction( int numerator, int denominator ){ ... }
    public int getNumerator(){ return numerator; }
    public int getDenominator(){ return denominator; }
    public Rational times( Rational that ){ ... }
}
```

Sorozat típusok ismét

- `int[]`
- `java.util.ArrayList<Integer>`
- `java.util.LinkedList<Integer>`

Láncolt ábrázolás

```
public class LinkedList<T> {
    private T head;
    private LinkedList<T> tail;
    public LinkedList(){ ... }
    public T get( int index ){ ... }
    public void set( int index, T item ){ ... }
    public void add( T item ){ ... }
    ...
}
```



Generikus interface

java/util/List.java

```
package java.util;
public interface List<T> {
    T get( int index );
    void set( int index, T item );
    void add( T item );
    ...
}
```

java/util/ArrayList.java

```
package java.util;
public class ArrayList<T> implements List<T>{
    public ArrayList(){ ... }
    public T get( int index ){ ... }
    ...
}
```

Altípusosság

```
class Fraction implements Rational { ... }
class ArrayList<T> implements List<T> { ... }
```

- `Fraction <: Rational`
- `Simplified <: Rational`
- Minden `T`-re: `ArrayList<T> <: List<T>`
- Minden `T`-re: `LinkedList<T> <: List<T>`

Liskov-féle helyettesítési elv

LSP: Liskov's Substitution Principle

Egy A típus altípusa a B (bázis-)típusnak, ha az A egyedeit használhatjuk a B egyedei helyett, anélkül, hogy ebből baj lenne.

Az interface egy típus

```
List<String> names;

static List<String> noDups( List<String> names ){
    ...
}
```

Nem példányosítható

```
List<String> names = new List<String>();    // fordítási hiba
```

Az osztály is egy típus, és példányosítható

```
ArrayList<String> names = new ArrayList<String>();
ArrayList<String> nicks = new ArrayList<>();
```

Típusozás interface-szel, példányosítás osztállyal

```
List<String> names = new ArrayList<>();
```

Jó stílus...

Statikus és dinamikus típus

Változó (vagy paraméter) „deklarált”, illetve „tényleges” típusa

```
List<String> names = new ArrayList<>();


static List<String> noDups( List<String> names ){
    ... names ...
}


List<String> shortList = noDups(names);
```

Speciális jelentésű interface-ek

```
class DataStructure<T> implements java.lang.Iterable<T>
// működik rá az iteráló ciklus

class Resource implements java.lang.AutoCloseable
// működik rá a try-with-resources

class Rational implements java.lang.Cloneable
// működik rá a (sekély) másolás

class Data implements java.io.Serializable
// működik rá az objektumszerializáció
```

OOP paradigma

- Absztrakció
  - Egységbe zárás
  - Információ elrejtés
- Öröklődés
- Altípusos polimorfizmus
- Dinamikus kötés

Outline

Öröklődés (inheritance)

```
class A extends B { ... }
```

- Egy típust egy másik típusból származtatunk
  - Csak a különbségeket kell megadni: A Δ B
  - Újrafelhasználás
- Itt: az A a gyermekosztálya a B szülőosztálynak
- Tranzitivitás: leszármazott osztály – ősosztály
  - alosztály: subclass, derived class
  - bázisosztály: super class, base class
- Körkörösség kizárva!

Öröklődéssel definiált osztály

- A szülőosztály tagjai átöröklődnek
- Újabb tagokkal bővíthető (Java: `extends`)
- Megörökölt példánymetódusok újradefiniálhatók
  - ... és újradeklarálhatók

```
public class Date {
    public final int year, month, day;
    public Date( int year, int month, int day ){ ... }
}
```

```
public class Time extends Date {
    public final int hour, minute;
    public Time( int y, int m, int d, int hour, int minute ){ ... }
}
```

A konstruktor(ok) megírandó(k)!

```
class Date {
    private int year, month, day;
    Date( int year, int month, int day ){
        this.year  = year; ...
    }
    ...
}

class Time extends Date {
    private int hour, minute;
    Time( int y, int m, int d, int hour, int minute ){
        super(y,n,d);              // szülőosztály konstruktorát
        this.hour = hour; ...
    }
    ...
}
```

Öröklődéssel definiált interface

Adatszerkezetek bejárásához

```
package java.util;
public interface Iterator<E> {
    boolean hasNext();
    E next();
}
```

Új műveletekkel való kibővítés

```
package java.util;
public interface ListIterator<E> extends Iterator<E> {
    boolean hasPrevious();
    E previous();
    ...
}
```

Típusok közötti származtatás

- Interface `extends` interface
- Osztály `implements` interface
- Osztály `extends` osztály

Többszörös öröklődés

(Multiple inheritance)

- Egy típust több más típusból származtatunk
- Javában: több interface-ből
- Problémákat vet fel

Példák

OK

```
package java.util;
public class Scanner implements Closeable, Iterator<Sring> { ... }
```

OK

```
interface PoliceCar extends Car, Emergency { ... }
```

Hibás

```
    class PoliceCar extends Car, Emergency { ... }
```

Különbség class és interface között

- Osztályt lehet példányosítani
  - `abstract class`?
- Osztályból csak egyszeresen örökölhetünk
  - `final class`?
- Osztályban lehetnek példánymezők
  - interface-ben: `public static final`

abstract class

- Részlegesen implementált osztály
  - Tartalmazhat `abstract` metódust
- Nem példányosítható
- Származtatással konkretizálhatjuk

```
package java.util;
public abstract class AbstractList<E> implements List<E> {
    ...
    public abstract E get( int index );  // csak deklarálva
    public Iterator<E> iterator(){ ... } // implementálva
    ...
}
```

Konkretizálás

```
public abstract class AbstractCollection<E> implements Collection<E>
    ...
    public abstract int size();
}
```

```
public abstract class AbstractList<E> extends AbstractCollection<E>
                                              implements List<E> {
    ...
    public abstract E get( int index );
}
```

```
public class ArrayList<E> extends AbstractList<E> {
    ...
    public int size(){ ... }              // implementálva
    public E get( int index ){ ... }      // implementálva
}
```

Öröklődésre tervezés

- Könnyű legyen származtatni belőle
- Ne lehessen elrontani a típusinvariánst

protected láthatóság

```
package java.util;
public abstract class AbstractList<E> implements List<E> {
    ...
    protected int modCount;
    protected AbstractList(){ ... }
    protected void removeRange(int fromIndex, int toIndex){ ... }
    ...
}
```

- Ugyanabban a csomagban
- Más csomagban csak a leszármazottak

`private` $\subseteq$ félnyilvános (package-private) $\subseteq$ `protected` $\subseteq$ `public`

A private tagok nem hivatkozhatók a leszármazottban!

```
class Counter {
    private int counter = 0;
    public int count(){ return ++counter; }
}

class SophisticatedCounter extends Counter {
    public int count( int increment ){
        return counter += increment;   // fordítási hiba
    }
}
```

Javítva

```
class Counter {
    private int counter = 0;
    public int count(){ return ++counter; }
}

class SophisticatedCounter extends Counter {
    public int count( int increment ){
        if( increment < 1 ) throw new IllegalArgumentException();
        while( increment > 1 ){
            count();
            --increment;
        }
        return count();
    }
}
```

protected

```
package my.basic.types;
public class Counter {
    protected int counter = 0;
    public int count(){ return ++counter; }
}
```

```
package my.advanced.types;
class SophisticatedCounter extends my.basic.types.Counter {
    public int count( int increment ){
        return counter += increment;
    }
}
```

final class

```
package java.lang;
public final class String implements ... { ... }
```

- Nem lehet belőle leszármaztatni
- Módosíthatatlan (immutable) esetben nagyon hasznos

Mi lehet egy interface-ben?

- Absztrakt (példány)metódus
- [Statikus metódus]
- [Példánymetódus `default` implementációval]
- [`private` példánymetódus]
- Konstansdefiníció

Réges-régen, egy messzi-messzi galaxisban...

```
interface Color {
    int BLACK = 0;    // public static final
    int WHITE = 1;
    int GREEN = 2;
    ...
}
```

Felsorolási típus

```
enum Color { BLACK, WHITE, GREEN }
```

- Nem példányosítható
- Nem származtatható le belőle
- Használható `switch`-utasításban
- ...

Az öröklődés két aspektusa

- Kódöröklés
- Altípusképzés

Outline

Altípus fogalma

A <: B

LSP: Liskov's Substitution Principle

Egy *A* típus altípusa a *B* (bázis-)típusnak, ha az *A* egyedeit használhatjuk a *B* egyedei helyett, anélkül, hogy ebből baj lenne.

Öröklődés $\Rightarrow$ altípusosság

class A implements I

$A\ \Delta_{ci}\ I \Rightarrow A <: I$

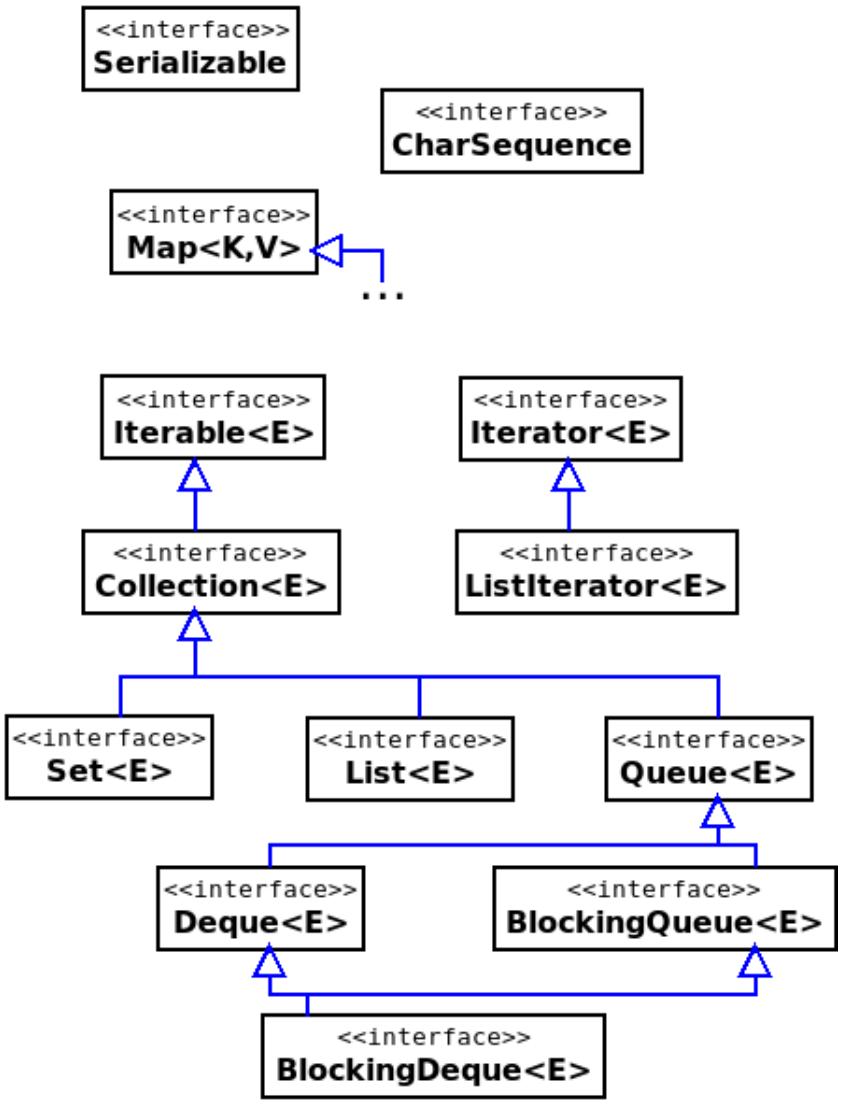
class A extends B

$A\ \Delta_{c}\ B \Rightarrow A <: B$

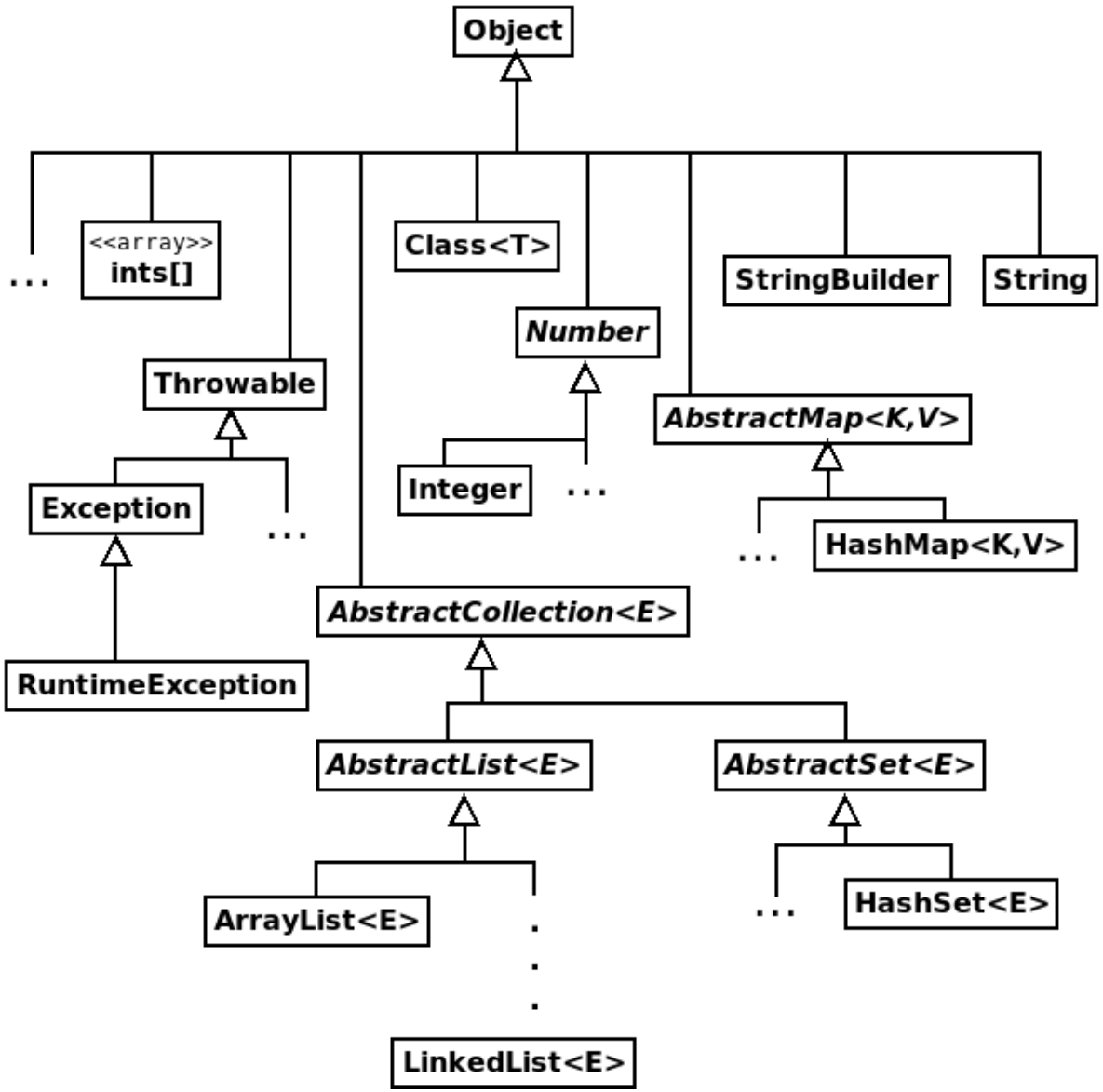
interface I extends J

$I\ \Delta_{i}\ J \Rightarrow I <: J$

Interface-ek hierarchiája a Javában (részlet)



Osztályok hierarchiája a Javában (részlet)



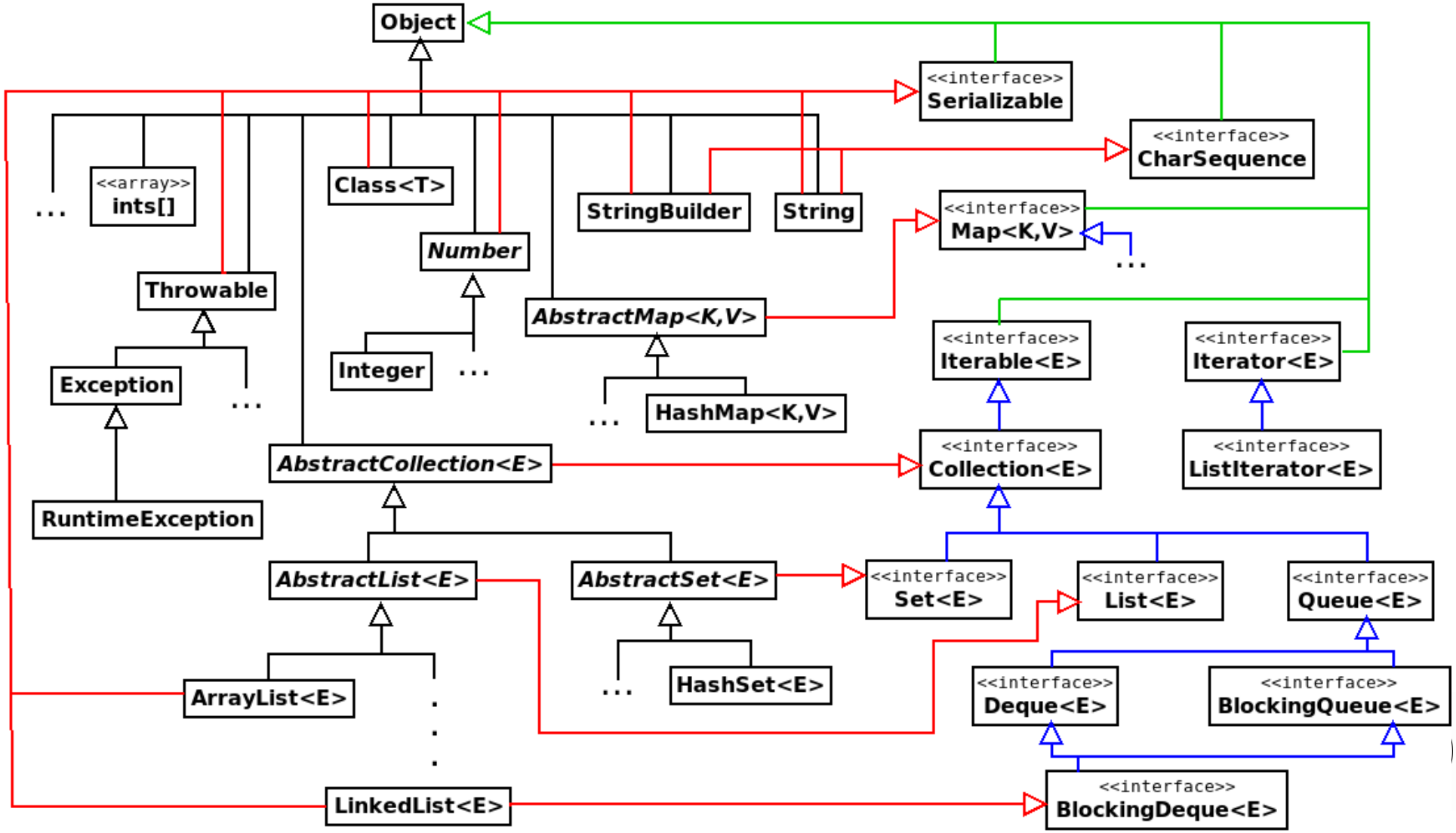
java.lang.Object

Minden osztály belőle származik, kivéve önmagát!

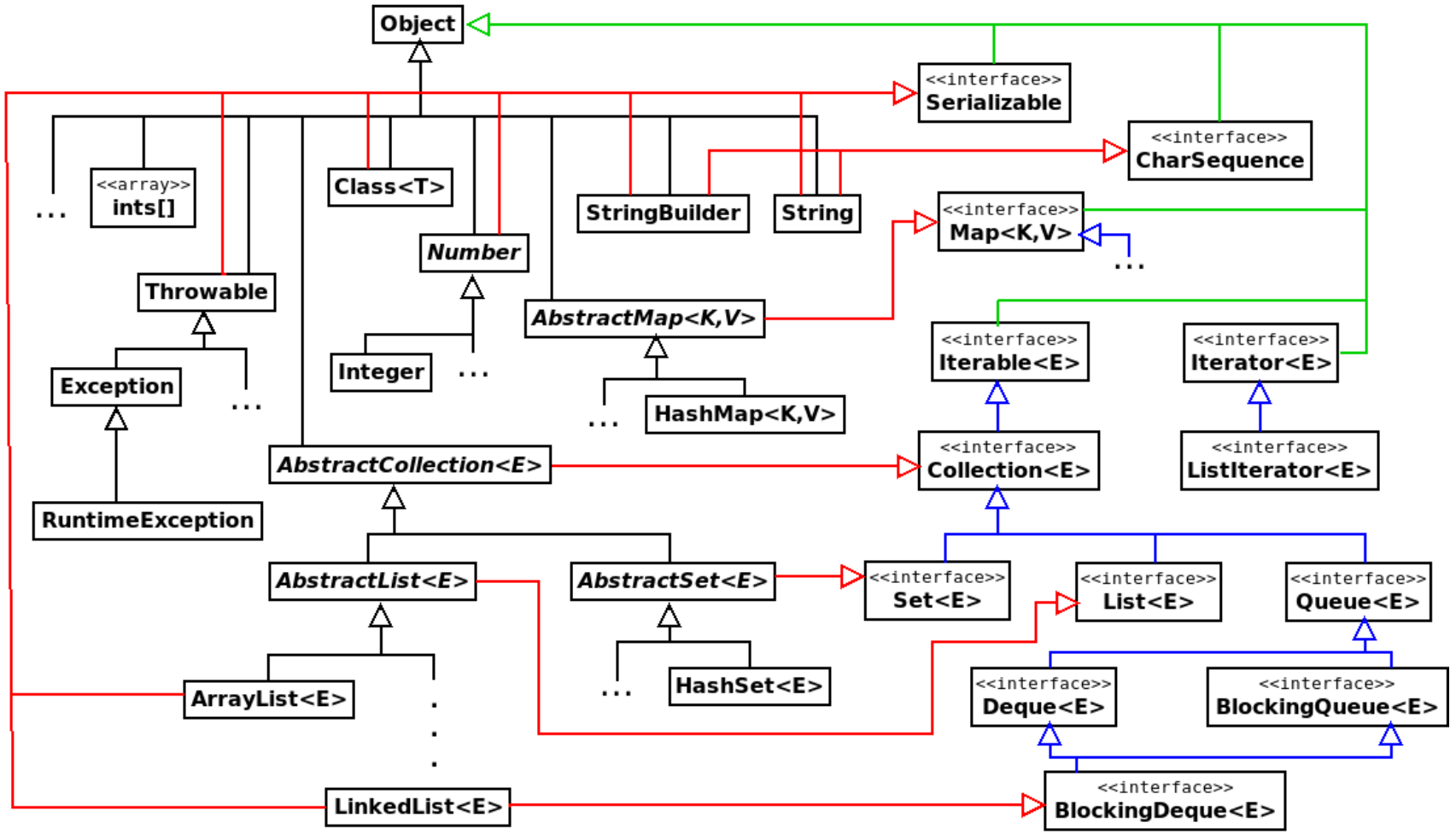
```
package java.lang;
public class Object {
    public Object(){ ... }
    public String toString(){ ... }
    public int hashCode(){ ... }
    public boolean equals( Object that ){ ... }
    ...
}
```

Referenciatípusok hierarchiája a Javában (részlet)

körmentes irányított gráf (DAG: directed acyclic graph)



Típusok hierarchiája a Javában (részlet)



boolean
char
byte
short
int
long
float
double

Minden a java.lang.Object-ből származik

... kivéve a primitív típusokat!

```
class A {}
```

```
class A extends java.lang.Object {
    A(){            // generated default constructor
        super();    // calls no-arg constructor of parent
    }
}
```

Konstruktorok egy osztályban

- Egy vagy több explicit konstruktor
- Alapértelmezett konstruktor

Konstruktor törzse

Első utasítás

- Explicit `this`-hívás
- Explicit `super`-hívás
- Implicit (generálódó) `super()`-hívás (no-arg!)

Többi utasítás

Nem lehet `this`- vagy `super`-hívás!

Érdekes hiba

Ártatlannak tűnik

```
class Base {
    Base( int n ){}
}

class Sub extends Base {}
```

Jelentése

```
class Base extends Object {
    Base( int n ){
        super();
    }
}

class Sub extends Base {
    Sub(){ super(); }
}
```

Altípus reláció

$$<: \quad = \quad (\Delta_c \cup \Delta_i \cup \Delta_{ci} \cup \Delta_o)^*$$

- Δ_o jelentése: minden a `java.lang.Object`-ből származik
- ϱ^* jelentése: ϱ reláció reflexív, tranzitív lezártja
  - Ha `A` ϱ `B`, akkor `A` ϱ^* `B`
  - Reflexív lezárt: `A` ϱ^* `A`
  - Tranzitív lezárt: ha `A` ϱ^* `B` és `B` ϱ^* `C`, akkor `A` ϱ^* `C`

Ez egy parciális rendezés (RAT)!

A dinamikus típus a statikus típus altípusa

Ha `A` $<:$ `B`, akkor

- `B v = new A();` helyes
- `void m( B p )...` esetén `m(new A())` helyes

Specializálás

- Az altípus „mindent tud”, amit a bázistípus
- Az altípus speciálisabb lehet
- Ez az *is-egy* reláció
  - `Car` *is-a* `Vehicle`
  - `Boat` *is-a* `Vehicle`
- Emberi gondolkodás, OO modellezés

Többszörös altípusképzés

- Egy fogalom több általános fogalom alá tartozhat
  - `PoliceCar` *is-a* `Car` **és** *is-a* `EmergencyVehicle`
  - `FireBoat` *is-a* `Boat` **és** *is-a* `EmergencyVehicle`
- Összetett fogalmi modellezés Javában: `interface`

Altípusos polimorfizmus

Ha egy kódbázist megírtunk, újrahasznosíthatjuk speciális típusokra!

- Általánosabb típusok helyett használhatunk altípusokat
- Több típusra is működik a kódbázis: polimorfizmus

Újrafelhasználhatóság!

Kivételosztályok hierarchiája

- java.lang.Throwable
  - java.lang.Exception
    - java.sql.SQLException
    - java.io.IOException
      - java.io.FileNotFoundException
    - ...
    - saját kivételek általában ide kerülnek
    - java.lang.RuntimeException
      - java.lang.NullPointerException
      - java.lang.ArrayIndexOutOfBoundsException
      - java.lang.IllegalArgumentException
      - ...
  - java.lang.Error
    - java.lang.VirtualMachineError
    - ...

Nem ellenőrzött kivételek

- `java.lang.RuntimeException` és leszármazottjai
- `java.lang.Error` és leszármazottjai

Egyes alkalmazási területen akár ezek is kezelendők!

Kivételkezelő ágak

```
try {
    ...
} catch( FileNotFoundException e ){
    ...
} catch( EOFException e ){
    ...
} // nem kezeltük a java.net.SocketException-t
```

Speciálisabb-általánosabb kivételkezelő ágak

```
try {
    ...
} catch( FileNotFoundException e ){
    ...
} catch( EOFException e ){
    ...
} catch( IOException e ){  // minden egyéb IOException
    ...
}
```

Fordítási hiba: elérhetetlen kód

```
try {
    ...
} catch( FileNotFoundException e ){
    ...
} catch( IOException e ){  // minden egyéb IOException
    ...
} catch( EOFException e ){ // rossz sorrend!
    ...
}
```

Outline

Példánymetódusok felüldefiniálhatók

redefine/override an instance method

```
package java.lang;
public class Object {
    public String toString(){ ... }
    ...
}
```

```
public class Date {
    ...
    @Override public String toString(){
        return year + "." + month + "." + day + ".";
    }
}
```

És újra felüldefiniálhatók

```
public class Date {
    ...
    @Override public String toString(){
        return year + "." + month + "." + day + ".";
    }
}
```

```
public class Time extends Date {
    ...
    @Override public String toString(){
        return year + "." + month + "." + day + ". " +
               hour + ":" + minute;
    }
}
```

Meghívható a szülőben adott implementáció

```
public class Date {
    ...
    @Override public String toString(){
        return year + "." + month + "." + day + ".";
    }
}
```

```
public class Time extends Date {
    ...
    @Override public String toString(){
        return super.toString() + " " +
                hour + ":" + minute;
    }
}
```

Túlterhelés és felüldefiniálás

```
package java.lang;
public final class Integer extends Number {
    ...
    @Override public String toString(){ ... }
    public static String toString( int i ){ ... }
    public static String toString( int i, int radix ){ ... }
    ...
}
```

Különbségtétel

Túlterhelés

- Ugyanazzal a névvel, különböző paraméterezéssel
- Megörökölt és bevezetett műveletek között
- Fordító választ az aktuális paraméterlista szerint

Felüldefiniálás

- Bázisosztályban adott műveletre
- Ugyanazzal a névvel és paraméterezéssel
  - Ugyanaz a metódus
  - Egy példánymetódusnak lehet több implementációja
- Futás közben választódik ki a „legspeciálisabb” implementáció

Dinamikus kötés

dynamic/late binding

```
Time t = new Time(2003,11,22,17,25);
Date d = t;       // altípusosság (up-cast)
Object o = d;

System.out.println( t.toString() );  // 2003.11.22. 17:25
System.out.println( d.toString() );
System.out.println( o.toString() );
```

Példánymetódus hívásához használt kitüntetett paraméter dinamikus típusához legjobban illeszkedő implementáció hajtódik végre.

A statikus és a dinamikus típus szerepe

Statikus típus

Mit szabad csinálni a változóval?

- Statikus típusellenőrzés

```
Object o = new Date(1970,1,1);
o.setYear(2000);  // fordítási hiba
```

Dinamikus típus

- Melyik implementációját egy felüldefiniált műveletnek?

```
Object o = new Date(1970,1,1);
System.out.println(o);  // toString() impl. kiválasztása
```

- Dinamikus típusellenőrzés

Típuskényszerítés (down-cast)

- A „`(Date)o`” kifejezés statikus típusa `Date`
- Ha `o` dinamikus típusa `Date`, akkor működik
- Ha nem, `ClassCastException` lép fel (futási hiba)

Dinamikus típusellenőrzés

- Futás közben, dinamikus típus alapján
- Pontosabb, mint a statikus típus
  - Altípus lehet
- Rugalmasság
- Biztonság: csak ha explicit kérjük (type cast)

```
Object o = new Time(2003,11,22,17,25);
...
if( o instanceof Date ){
    ((Date)o).setYear(2000);
}
```

Statikus és dinamikus típus: összefoglalás

Változók, paraméterek, kifejezések esetén

Statikus

- Deklarált
- Osztály/interface
- Állandó
- Fordítási időben ismert
- Általánosabb
- Statikus típusellenőrzéshez
- Biztonságot ad

Dinamikus

- Tényleges
- Osztály
- Változhat futás közben
- Futási időben derül ki
- Speciálisabb
- Dinamikus típusellenőrzéshez
- Rugalmasságot ad

Dinamikus típus ábrázolása futás közben

- `java.lang.Class` osztály objektumai
- Futás közben lekérhető

```
Object o = new Time(2003,11,22,17,25);
Class c = o.getClass();   // Time.class
Class cc = c.getClass();  // Class.class
```

Dinamikus kötés megörökölt metódusban is!

```
public class Date {
    ...
    @Override public String toString(){ ... }
    public void print(){
        System.out.println( toString() );
    }
}
```

```
public class Time extends Date {
    ...
    @Override public String toString(){ ... }
}
```

```
Object o = new Time(2003,11,22,17,25);
((Date)o).print();
```

Dinamikus kötés: csak példánymetódusra

- *Felüldefiniálni* csak példánymetódust lehet
  - ha nem `final`
- *Megvalósítani* `abstract`-ot, pl. `interface`-ből

Kell a kitüntetett paraméter (dinamikus típusa)

Mező és osztályszintű metódus nem definiálható felül

Elfedési szabályok...

Példa: equals-metódus

- *Tartalmi* egyenlőségvizsgálat referenciatípusokra

```
Date d1 = new Date(1970,1,1);
Date d2 = new Date(1970,1,1);
System.out.println( d1 == d2 );
System.out.println( d1.equals(d2) );
```

- Egy equals metódus sok (rész)implementációval
  - Együttesen adnak egy összetett implementációt
- Szerződése betartandó
  - Determinisztikus
  - Ekvivalencia-reláció (RST)
  - Igaz ez: `a == null || a.equals(null) == false`
  - Konzisztens a hashCode metódussal

Szabályos felüldefiniálás

```
package java.lang;
public class Object {
    ...
    public boolean equals( Object that ){ return this == that; }
    public int hashCode(){ ... }
}
```

```
public class Date {
    ...
    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Date d = (Date)that;
            return year == d.year && month == d.month && ... ;
        } else return false;
    }
    public int hashCode(){ return 512*year + 32*month + day; }
}
```

Jellemző hiba

```
package java.lang;
public class Object {
    ...
    public boolean equals( Object that ){ return this == that; }
    public int hashCode(){ ... }
}
```

Fordítási hiba a @Override-nak köszönhetően

```
public class Date {
    ...
    @Override public boolean equals( Date that ){
        return that != null && year == that.year && && ... ;
    }
    public int hashCode(){ return 512*year + 32*month + day; }
}
```

Nagyon valószínű, hogy bug, és egyben rossz gyakorlat

```
package java.lang;
public class Object {
    ...
    public boolean equals( Object that ){ return this == that; }
    public int hashCode(){ ... }
}
```

Túlterhelés, nincs dinamikus kötés

```
public class Date {
    ...
    public boolean equals( Date that ){
        return that != null && year == that.year && && ... ;
    }
    public int hashCode(){ return 512*year + 32*month + day; }
}
```