Programozási nyelvek Java

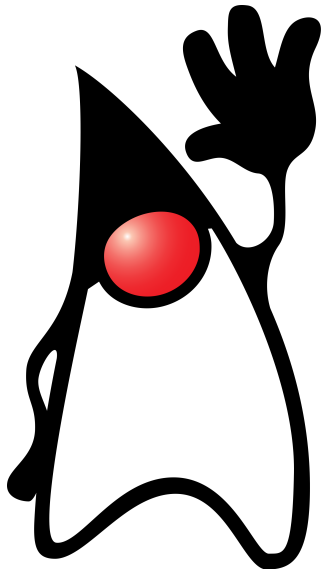
Alapok

Kozsik Tamás

A Java nyelv

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

Java Language Specification



ELTE
IK

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

ELTE IK

Történelem

James Gosling és mások, 1991 (Sun Microsystems)

Java version history

- 1991: Oak → Green → **Java**
- 1996: Java 1.0 (SE, Standard Edition)
- 1999: Enterprise Edition (J2EE, Jakarta EE)
- 2010: a Java az Oracle-höz kerül
- SE LTS kiadások: Java 11 (2018), Java 17 (2021)

ELTE
IK

Java Virtual Machine

- Alacsony szintű nyelv: bájtkód
- Sok nyelv fordítható rá (Ada, Closure, Eiffel, Jython, Kotlin, Scala...)
- Továbbfordítható
  - Just In Time compilation
- Dinamikus szerkesztés
- Kódmobilitás

Java Virtual Machine Specification

C és Java hasonlósága

```
// legnagyobb közös osztó
int lnko(int a, int b) {
  while (b != 0) {
    int c = a % b;
    a = b;
    b = c;
  }
  return a;
}
```

ELTE
IK

C és Java különbsége

```
double sum(double array[]) {
  double s = 0.0;
  for (int i = 0; i < array.length; ++i) {
    s += array[i];
  }
  return s;
}
```

C és Java különbsége - hangsúlyosabban

```
double sum(double[] array) {
  double s = 0.0;
  for (double item: array) {
    s += item;
  }
  return s;
}
```

Java programok felépítése

(első blikkre)

- [modul (`module`)]
- csomag (`package`)
- osztály (`class`)
  - ◇ adattag (mező, field)
  - ◇ metódus (method) vagy kicsit pontatlanul *függvény*
    - ▶ utasítás (statement)
      - kifejezés (expression)
        - ◇ literál

Tag (member): adattagok és metódusok összefoglaló neve.

Literál: érték megjelenése a forráskódban, pl. `123` vagy `"abc"`.

ELTE
IK

Java forrásfájl

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

Hello World!

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

ELTE
IK

Parancssorban

```
$ ls
HelloWorld.java

$ javac HelloWorld.java

$ ls
HelloWorld.class  HelloWorld.java

$ java HelloWorld
Hello world!
```

Fordítás, futtatás

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

Java programok futása

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

ELTE
IK

Csomag

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

ELTE
IK

A package utasítás

```
package geometry;

public class Point {     // geometry.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;

public 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`

ELTE
IK

Compilation and execution

- Munkakönyvtár (working directory)
- Hierarchikus csomagszerkezet → 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
$
```

Fordítás: Java és C

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

Rekurzív fordítás

```
$ ls geometry/basics
Main.java  Point.java
$ javac geometry/basics/Main.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

ELTE
IK

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(){ ... }
}
```

ELTE
IK

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);
// fordítási hiba: nincs Time a névtelenben    ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
  }
}
```

Egy csomagon belül

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

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

public class Time { ... }
```

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

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

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

Egy forrásfájlban több típusdefiníció

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

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

public class Time {
    ...
}

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

IK

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.eloadas.*;`
- Nem tranzitív
- A `java.lang` csomag típusait nem kell
- Névütközés: teljes név kell
  - `java.util.List`
  - `java.awt.List`

Fordítási egység szerkezete

- opcionális `package` utasítás
- 0, 1 vagy több `import` utasítás
- 1 vagy több típusdefiníció

javac kapcsolók

```
-d <directory>
    Specify where to place generated class files

--source-path <path>, -sourcepath <path>
    Specify where to find input source files

--class-path <path>, -classpath <path>, -cp <path>
    Specify where to find user class files...
```

Classpath

```
javac -classpath .:/usr/lib/java:/opt/java/myfiles.jar \\
      geometry/basics/Point.java

java -classpath .:/usr/lib/java:/opt/java/myfiles.jar \\
      geometry.basics.Main
```

- Ha kell a `colors.RGB` osztály:
  - `./colors/RGB.class`
  - `/usr/lib/java/colors/RGB.class`
  - `/opt/java/myfiles.jar`-ban `colors/RGB.class`
- Windows alatt: `.;C:\Users\kto\mylib;D:\myfiles.jar`
- `CLASSPATH` környezeti változó

ELTE
IK

jar fájlok

- Java Archive
- ZIP-tömörítésű fájl
- `jar` parancs az SDK-ban

Egységteszt (Unit test)

- A program legkisebb, önálló részeinek kipróbálása
  - ◇ Egység lehet: metódus, **osztály**, komponens/modul
  - ◇ Nem egységteszt, ha külső függőségei vannak
    - ▶ Ilyen pl.: fájlrendszer, adatbázis, hálózat használata
- Kis, gyorsan lefutó, független tesztek
  - ◇ Futási időben működik
  - ◇ Fekete dobozos: az egység belső szerkezete nem ismert
    - ▶ Csak az osztály publikus interfészét (metódusait) használja
- Funkcionális helyességet tesztel: a lefutás az elvárt eredményt adja-e
  - ◇ Nem cél: hatékonyság tesztelése

ELTE
IK

Egységteszt: helyesség

- Nem *bizonyítja*, csak *alátámasztja* a helyességet
- Regressziók felfedése: hamar kiderül, ha hibás a kód
- Egyúttal dokumentálja, mi az elvárt működés
  - ◇ Együtt fejlődik a kóddal: ezt a fordítóprogram „érti'' és ellenőrzi
  - ◇ A szöveges dokumentáció elavulhat
- Lefedettség (code coverage)
- Sok hibát megelőz még fejlesztés alatt
  - ◇ Nagyobb munkaigény kezdetben
  - ◇ Olcsóbb lehet az utólagos hibajavításnál
    - ▶ Az éles rendszer jobban működik

ELTE
IK

Egységteszt: módszerek

- Tesztvezérelt fejlesztés (test driven development, TDD)
  1. Új teszteset hozzáadása, ami még "piros" (sikertelen)
  2. Kód írása/fejlesztése: minden teszteset legyen "zöld" (sikeres)
  3. A kód minőségének javítása (refaktorálás): minden "zöld"
- Egyéb tesztelési megközelítések
  - ◇ Naplózás, kiírások használata
  - ◇ Hibakeresés (debugging)
  - ◇ Összetettebb: integrációs ∼, teljesítmény∼, stressz/terhelési ∼, automatizált ∼, véletlenített/tulajdonság alapú ∼, mock ∼, folyamatos ∼ (CI/CD), ...
  - ◇ Felhasználói élmény: elfogadási ∼, biztonsági ∼, használati ∼, lokalizációs ∼, ...
  - ◇ Formális helyességbizonyítás

ELTE
IK

Egységtesztelő: így használandó

- Egy tesztelő metódus egyetlen vizsgálatot tartalmaz
- A lehető legegyszerűbb szerkezet: ciklus, elágazás, véletlen, ... nélkül
- Saját kódot teszteljünk, ne könyvtárakat
- Lebegőpontos típusok tesztelése: az eredménynek lehet pontatlansága
  - ◇ Extra paraméter: tűréshatár (delta)
- Számítás adatainak struktúrája: egyszerűtől bonyolultig
  - ◇ `null`
  - ◇ üres szöveg, `0`
  - ◇ konstruktorhívás, majd getter
  - ◇ kis, pozitív értékek
  - ◇ egy-két lépéssel összeállított adatok
  - ◇ negatív/szokatlan/extrém értékek
    - ▶ pl. `Integer.MAX_VALUE` vagy `Double.MIN_VALUE`
    - ▶ kivételek
  - ◇ hosszabb “történet”, több hívással

ELTE
IK

Egységtesztelés: FIRST

- Fast: μs-ms
- Isolated: egymástól és külvilágtól elkülönülő
- Repeatable: megismételhető
  - ◇ Nincsenek mellékhatások
  - ◇ Nincs nemdeterminisztikus futás
- Self-verifying: önellenőrző
  - ◇ Minden teszt elbukhat
  - ◇ Minden bukásnak pontosan egy oka lehet
- Timely: a kóddal együtt bővülnek/fejlődnek a tesztek
- vagy Thorough: lásd előző fólia

ELTE
IK

JUnit

- Java nyelvű megvalósítások közül a legnépszerűbb
- A jelenleg legújabb kiadás: JUnit 5, 1.9.2 verzió
- Innen letölthető a `jar` fájl
  - ◇ A letöltött fájl átnevezhető rövidebb névre, pl. `junit5.jar`
- Tesztelendő osztály: system under test (SUT)
  - ◇ Tegyük fel, hogy a `time.Time` osztályt teszteljük
  - ◇ A SUT kódja a `time/Time.java` fájlban van
  - ◇ A tesztelő kód a `time/TimeTest.java` fájlba kerül
- Fordítás: `javac -cp junit5.jar time/TimeTest.java`
- Futtatás: `java -jar junit5.jar -cp . -c time.TimeTest`

ELTE
IK

JUnit teszteset: Arrange-Act-Assert

```java
package time;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class DemoTest {
  @Test
  void testHour00_00() {
    // Step 1: Arrange
    Time sut = new Time(0, 0);
    // Step 2: Act
    int hour = sut.getHour();
    // Step 3: Assert
    assertEquals(0, hour);
  }
}
```

ELTE
IK

JUnit teszteset: Arrange-Act-Assert röviden

```
package time;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class DemoTest {
  @Test
  void testHour00_00() {
    assertEquals(0, new Time(0, 0).getHour());
  }
}
```

ELTE
IK

JUnit teszteset kimenete

- Fontos: az elvárt érték az első paraméter
  - ◇ Ez mindig egy konstans legyen, ne számított érték

```
@Test
void wrongResultTest() { assertEquals(5, 2+2); }
```

```
org.opentest4j.AssertionFailedError: expected:<5> but was:<4>
  ... (sok, érdektelen információ)
  at testing.DemoTest.wrongResultTest(DemoTest.java:9)
  ... (még több sor)
```

JUnit teszteset kimenete

- Fontos: az elvárt érték az első paraméter
  - ◇ Ez mindig egy konstans legyen, ne számított érték

```
@Test
void wrongResultTest() { assertEquals(5, 2+2); }
```

```
org.opentest4j.AssertionFailedError: expected:<5> but was:<4>
  ... (sok, érdektelen információ)
  at testing.DemoTest.wrongResultTest(DemoTest.java:9)
  ... (még több sor)
```

```
@Test
void wrongOrderTest() { assertEquals(2+2, 5); }
```

```
org.opentest4j.AssertionFailedError: expected:<4> but was:<5>
  at testing.DemoTest.wrongOrderTest(DemoTest.java:9)
```

JUnit: ritkábban használatos eszközök

```
fail();

assertEquals("y", "x", "expected to be y");

assertEquals("y", "x", () -> "Also expected to be y");
```

```
... AssertionFailedError
    at time.JUnitDemoTest.testFail(JUnitDemoTest.java:19)
...: expected to be y ==> expected: <y> but was: <x>
    at time.JUnitDemoTest.testMessageV1(JUnitDemoTest.java:24)
...: Also expected to be y ==> expected: <y> but was: <x>
    at time.JUnitDemoTest.testMessageV2(JUnitDemoTest.java:29)
```

JUnit: ritkábban használatos eszközök

```
@Test
public void testTrue() {
  assertTrue(2 + 2 == 4);
}

@Test
public void testFalse() {
  assertFalse("it's true" == "it's " + true);
}
```

- Az `assertEquals` jobb: precízebb a hibaüzenet
- Figyelem: a == nem helyes egyenlőségvizsgálat a `String` típuson!
  - ◇ Az ellenpárja, != szintén rossz

```
...: expected: <false> but was: <true>
    at time.JUnitDemoTest.testFalse(JUnitDemoTest.java:14)
```

JUnit: paraméterezett teszt: azonos működés több adaton

```
@CsvSource("this is some text,4")
@ParameterizedTest
public void testSplit(String text, int partCount) {
    assertEquals(partCount, text.split(" ").length);
}

@DisplayName("Computing the Fibonacci numbers")
@ParameterizedTest(name = "fib({0}) = {1}")
@CsvSource({"13,6", "21,7"})
public void testFib(int expected, int num) {
  assertEquals(expected, Fibonacci.fib(num));
}
```

```
'-- Computing the Fibonacci numbers [OK]
  +-- fib(6) = 13 [OK]
  '-- fib(7) = 21 [OK]
```

JUnit: paraméterezett tesztek szövegblokkokkal

Forrás: JUnit 5 dokumentációja

```
@ParameterizedTest(name = "[{index}] {arguments}")
@CsvSource(useHeadersInDisplayName = true, textBlock = """
  FRUIT,          RANK
  apple,          1
  strawberry,     700_000
  'lemon, lime', 0xF1
  """)
public void testWithCsvSource(String fruit, int rank) {
  // ...
}
```

Kimenet:

```
[1] FRUIT = apple, RANK = 1
[2] FRUIT = strawberry, RANK = 700_000
[3] FRUIT = lemon, lime, RANK = 0xF1
```

ELTE
IK

JUnit: kivételek

```
@Test
public void testInvalidTime() {
  InvalidTimeException exception =
    assertThrows(InvalidTimeException.class, () -> {
      new Time(123, 456);
    };
    assertEquals("/ by zero", exception.getMessage());
  }
}
```

- `() -> { ... }`: a kivételt potenciálisan kiváltó kódrészlet ide kerül
- A `.class` tekinthető speciális adattagnak
- Itt megengedett két `assertX` írása is egy tesztelő metódusba
  - ◇ Sokszor nincs üzenet, akkor változó sem szükséges

JUnit: tömbök

- Tömbök tesztelése: külön `assertArrayEquals` művelettel
  - ◇ `assertEquals` nem jó
  - ◇ Más adatszerkezetek jól működnek

```
@Test
public void testFibArray() {
  int[] fibs = Fibonacci.fibsUpTo(6);
  assertArrayEquals(new int[] { 1, 1, 2, 3, 5, 8 }, fibs);
}
```

ELTE
IK

JUnit: életciklus

```
public class TimeTest {
  private Time time;

  @BeforeEach
  public void beforeEach() {
    time = new Time(12, 34);
  }

  @Test void test1() { assertEquals(12, time.getHour()); }
  @Test void test2() { assertEquals(34, time.getMin());  }
  @Test void test3() { assertEquals(35, time.inc().getHour());
}
```

- `@BeforeEach`: tesztesetek ismétlődő adatainak közös beállítása
  - ◇ A tesztesetek nem zavarják egymást, mert mindig újrainicializál
- `@AfterEach`: pl. átmeneti fájlok törlésére
- `@BeforeAll`, `@AfterAll`: ritkán használatos

ELTE IK

CheckThat

- A szokásos JUnit tesztek a kód funkcionalitását vizsgálják
- Ez az eszköz a kód szerkezetét ellenőrzi
- Használata intuitív
- A megvalósító kód túlmutat a félév anyagán, nem kell megérteni

CheckThat példa

```
package time;

import static check.CheckThat.Condition.*;
import check.CheckThat;

import org.junit.jupiter.api.Test;

public class StructureTest01_Time {
  @Test
  public void test1() {
    CheckThat...
  }

  ...
}
```

CheckThat példa

```
CheckThat.theClass("time.Time")
   .thatIs(NOT_ABSTRACT, PUBLIC)
   .hasConstructorWithParams("int", "int")
     .thatIs(PUBLIC);
}

CheckThat.theClass("time.Time")
   .hasFieldOfType("hour", "int")
     .thatIs(PRIVATE, NOT_STATIC, MODIFIABLE)
     .has(GETTER, SETTER);

CheckThat.theClass("time.Time")
   .hasMethodWithParams("getEarlier", "Time")
     .thatIs(PUBLIC, NOT_STATIC)
     .thatReturns("Time");
```

ELTE
IK

CheckThat hibaüzenetek

```
org.opentest4j.MultipleFailuresError: Multiple Failures (1 fai
    ...: Nincsen megfelelő GETTER metódus
                                ehhez az adattaghoz: Time.hour
```

További üzenetek:

```
...: A Time.hour visszatérése nem megfelelő
...: A Time.hour láthatósága nem megfelelő
```

- Egy változóval angolra is állítható

CheckThat használata

```
package time;
import org.junit.platform.suite.api.*;

@Suite
@SelectClasses({
  StructureTest01_Time.class,
  StructureTest02_WorldTimes.class
  ,TimeTest.class
  ,WorldTimesTest.class    // (*)
})
public class TestSuite {}
```

- Fordítás: `javac -cp junit5.jar time/TimeTestSuite.java`
- Futtatás: `java -jar junit5.jar -cp . -c time.TimeTestSuite`
- A tesztelő kódhoz nem kell hozzányúlni
  - ◇ Ha még csak a `Time` osztály van készen, `(*)` kikommentezendő

CheckThat használata, elkülönülő tesztelő kód

```
root
+ project
  + src
    + time
      + Time.java
  + test
    + time
      + StructureTest01_Time.java
      + StructureTest02_WorldTimes.java
      + TestSuite.java
      + TimeTest.java
```

```
root
+ tester
  + junit5.jar
  + check
    + CheckThat.java
```

- Továbbra is ugyanabban a csomagban van a SUT és a tesztelő
- Fordítás: `javac -cp ../tester/junit5.jar;../tester test/time/*.java src/time/*.java`
- Futtatás: `java -jar ../tester/junit5.jar -cp ../tester:test:src -c time.TestSuite`

ELTE
IK

CheckThat használata, elkülönülő tesztelő kód

```
'-- JUnit Platform Suite [OK]
  '-- TestSuite [OK]
    '-- JUnit Jupiter [OK]
      '-- StructureTest01_Time [OK]
        +-- test1() [OK]
        +-- test2() [OK]
        '-- test3() [OK]
```

ELTE
IK

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) + ".";
```

Osztály, objektum, példányosítás

Point.java

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

Osztály, objektum, példányosítás

Point.java

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

Main.java

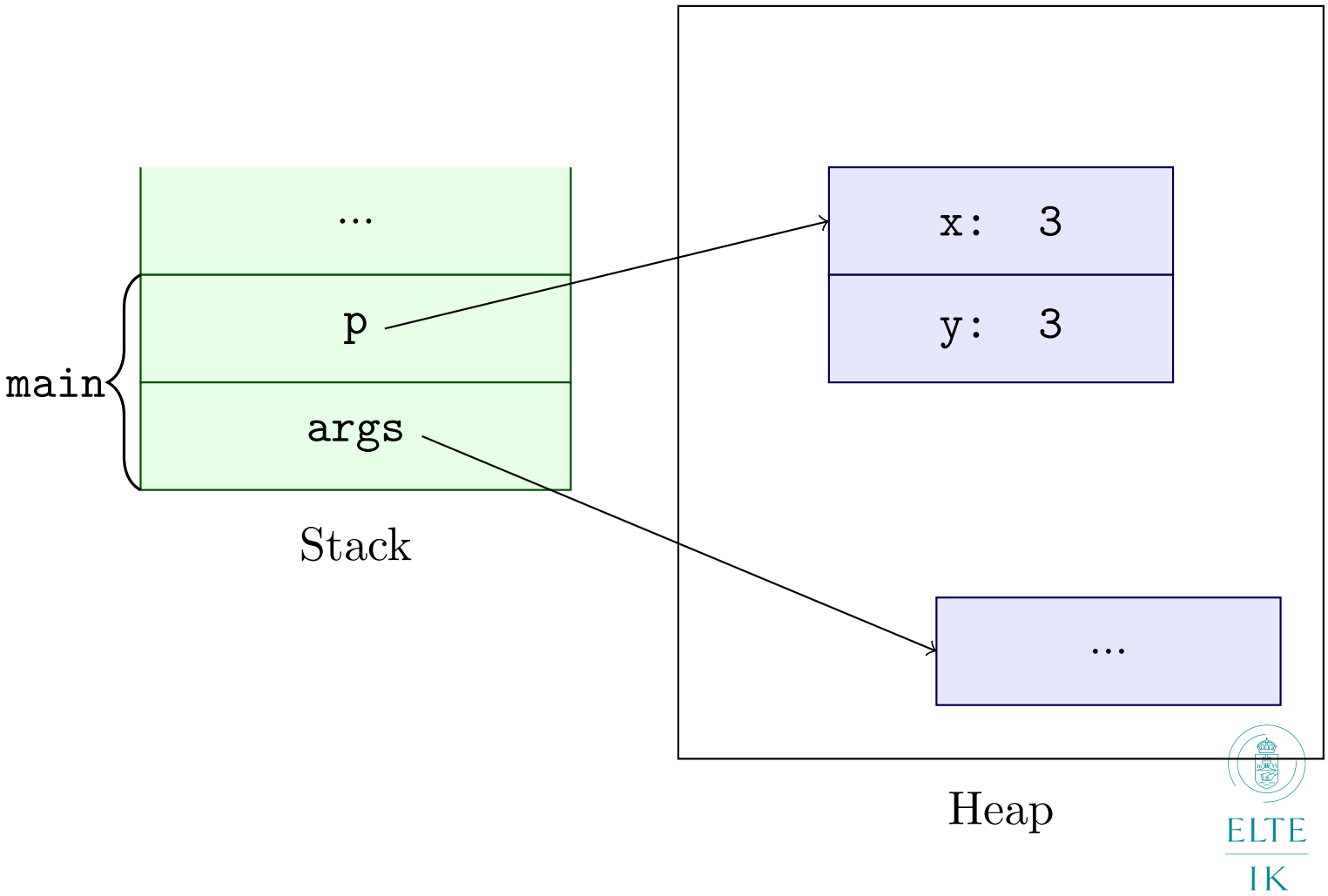
```
class Main {
    public static void main( String[] args ){   // főprogram
        Point p = new Point();     // példányosítás (heap)
        p.x = 3;                   // objektum állapotának
        p.y = 3;                      // módosítása
    }
}
```

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



ELTE IK

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

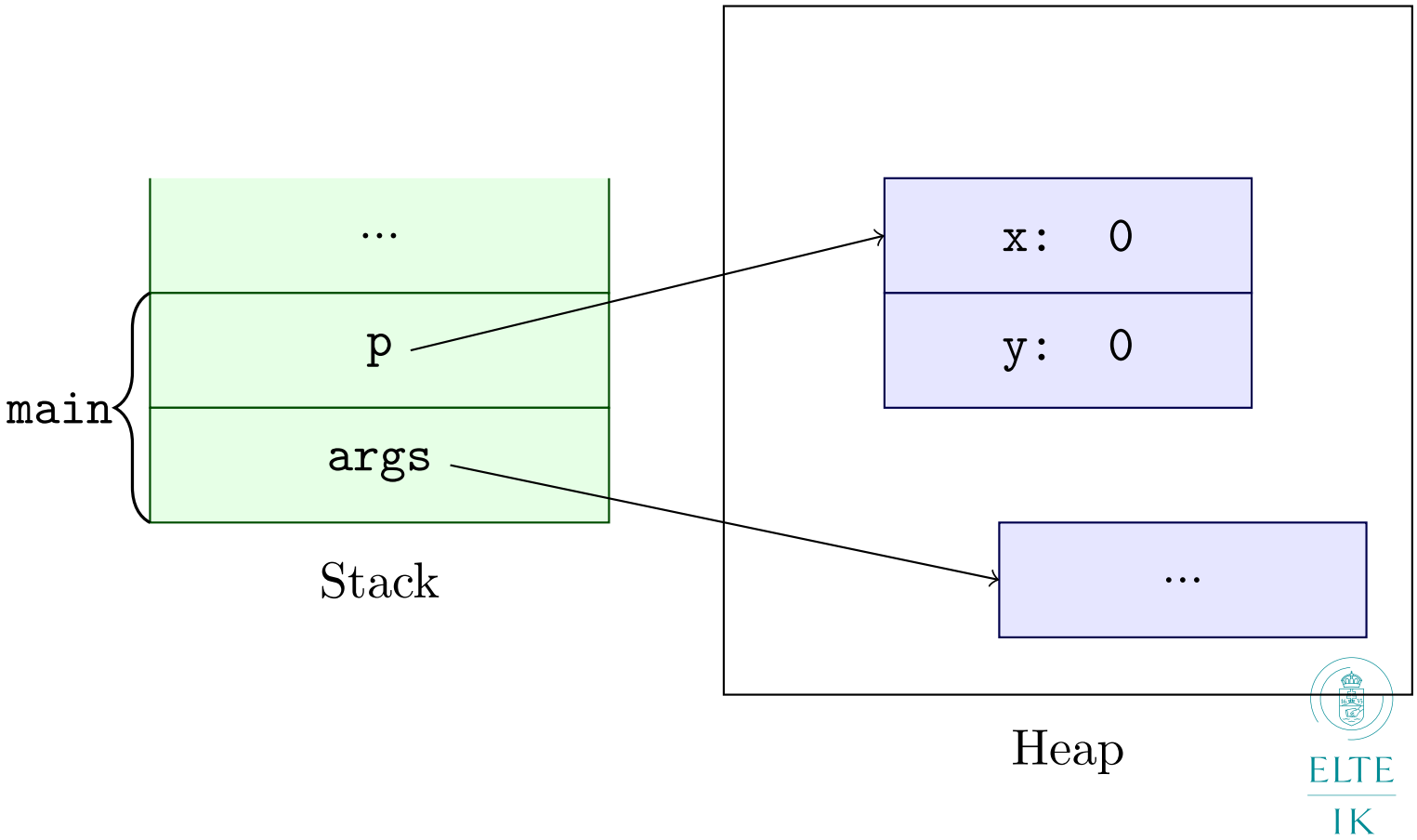
Metódus

```
class Point {
    int x, y;   // 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 ->
    }
}
```

ELTE
IK

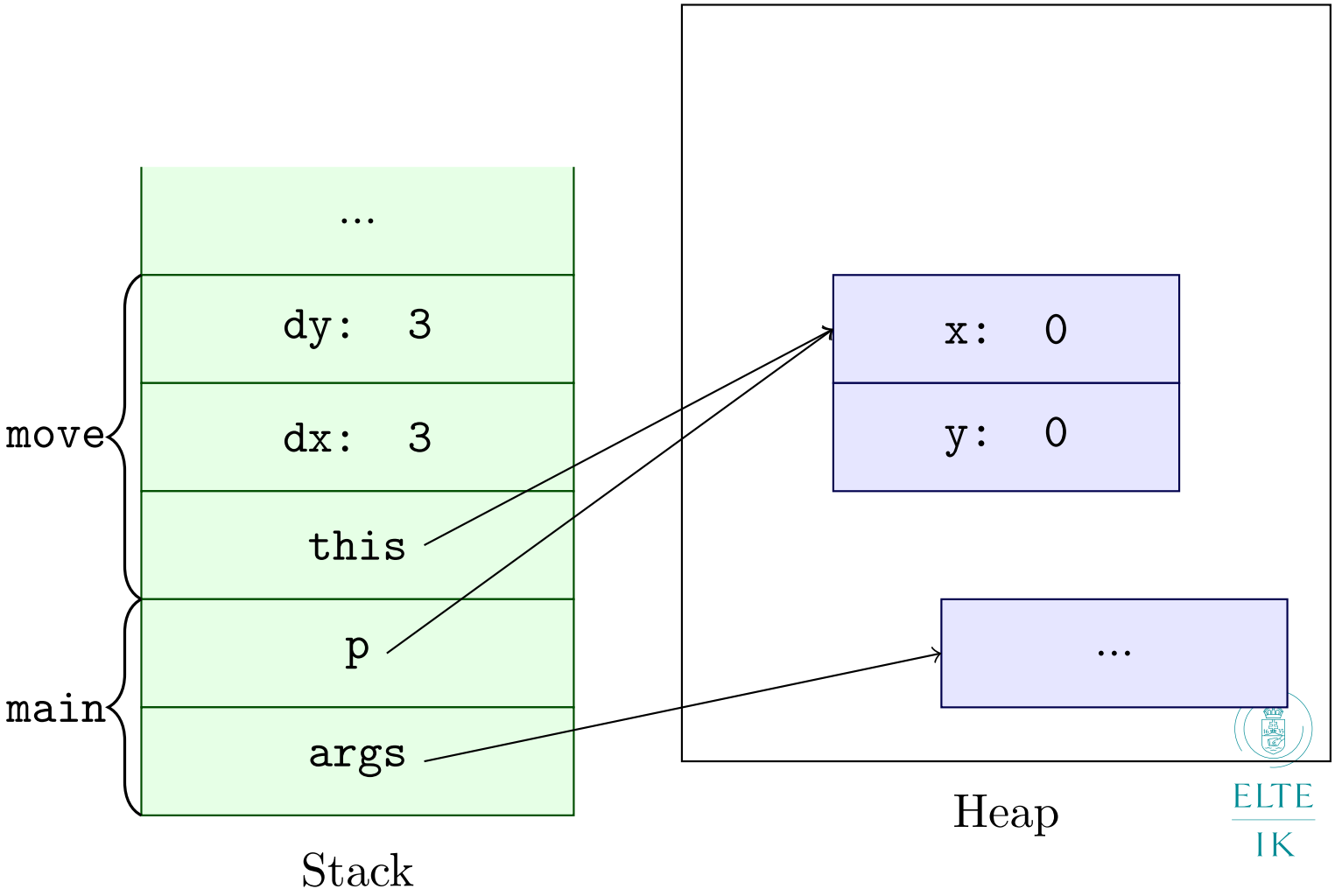
Metódus aktivációs rekordja – 1

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



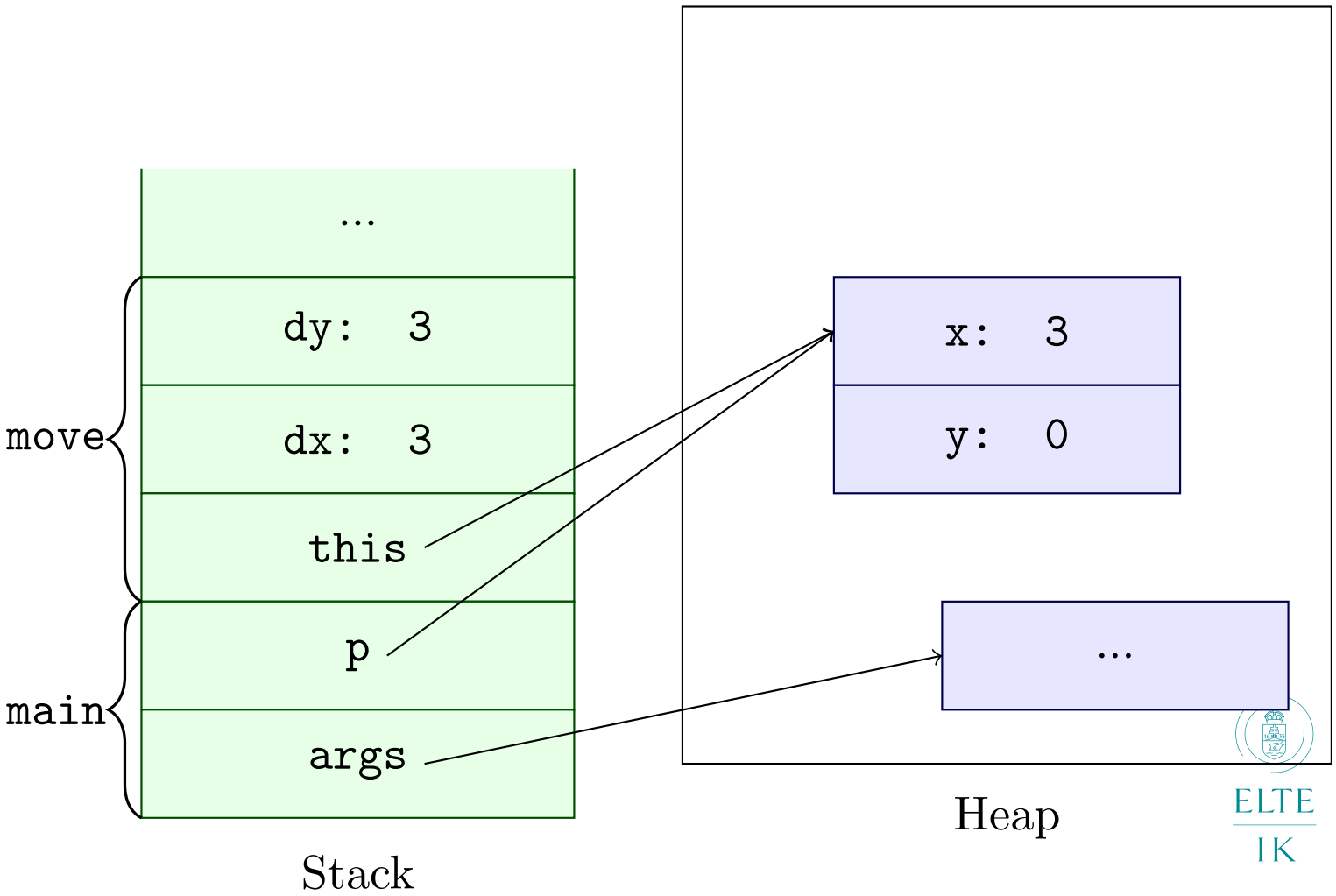
Metódus aktivációs rekordja – 2

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



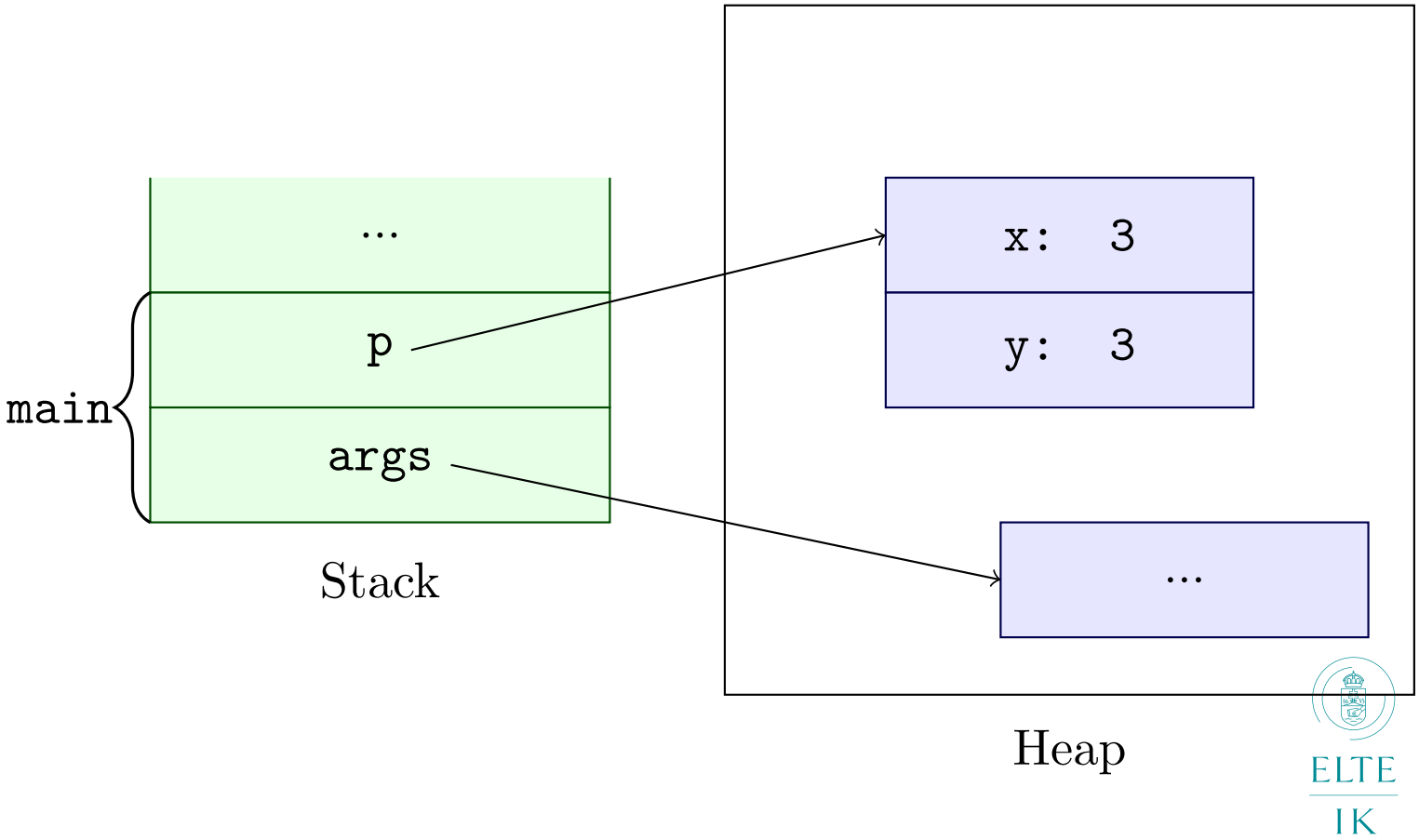
Metódus aktivációs rekordja – 3

```
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);
    }
}
```

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
    }
}
```

ELTE
IK

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
    }
}
```

ELTE
IK

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
    }
}
```

ELTE
IK

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
    }
}
```

ELTE
IK

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
    }
}
```

Generálódik egy paraméter nélküli, üres konstruktor

```
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;
```

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 { ... }
    }
}
```

IK

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() { ... }
}
```

IK

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 meghatározott célja van)

ELTE
IK

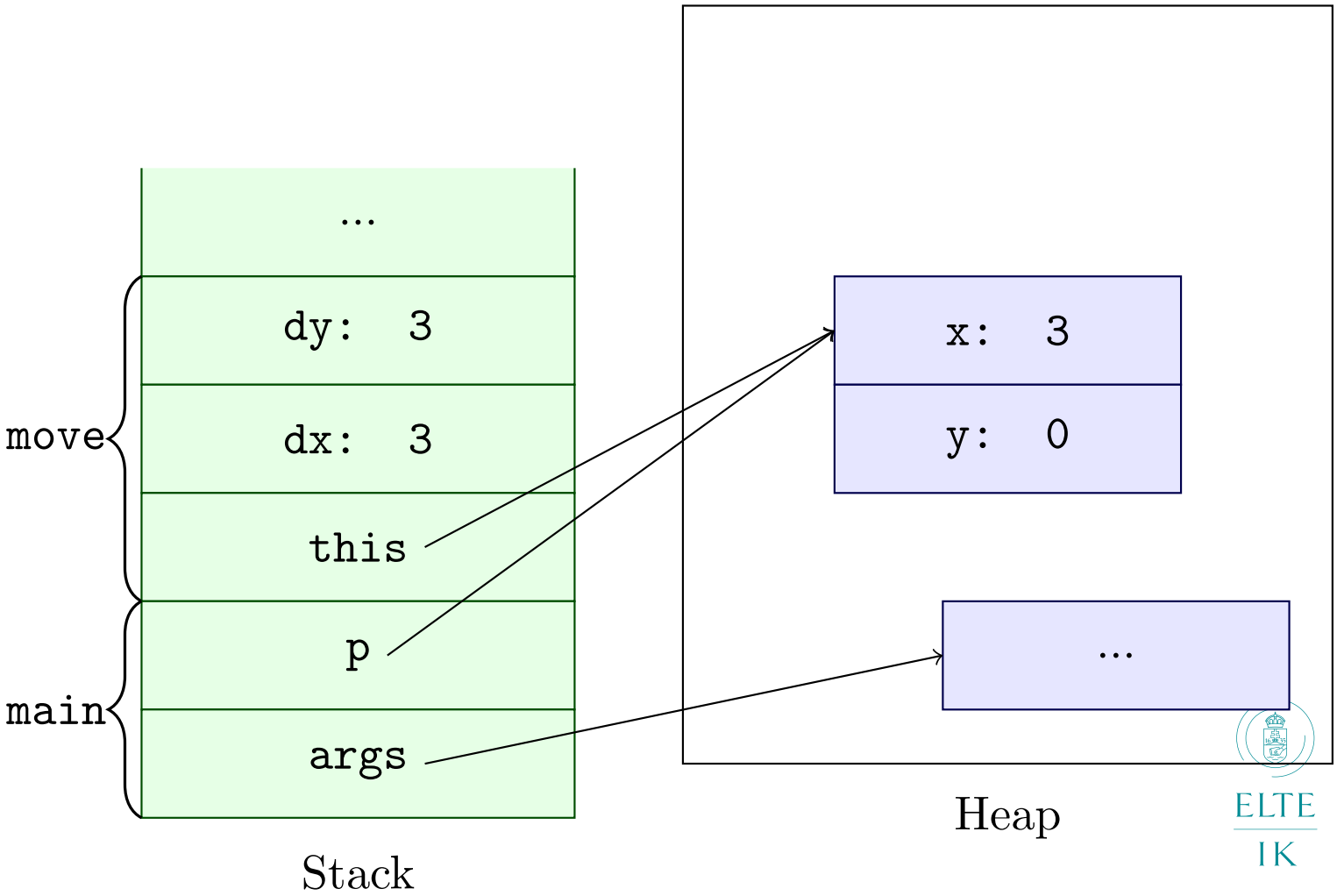
Hivatkozás (referencia)

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

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

ELTE
IK

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

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

ELTE
IK

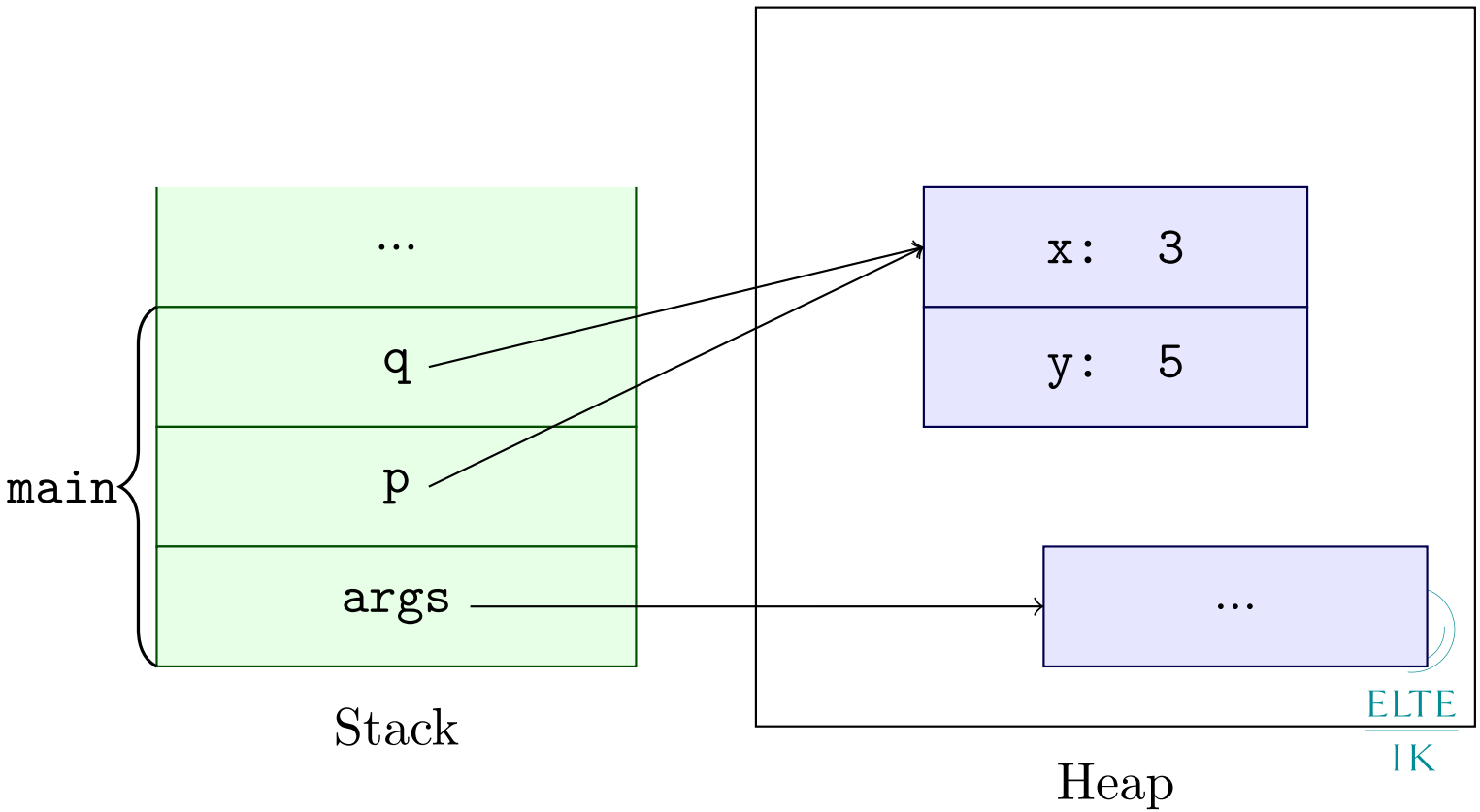
Objektumok élettartama

- 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;     // NullPointerExce
```

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

```
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 */
}
```

ELTE
IK

Statikus mezők

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

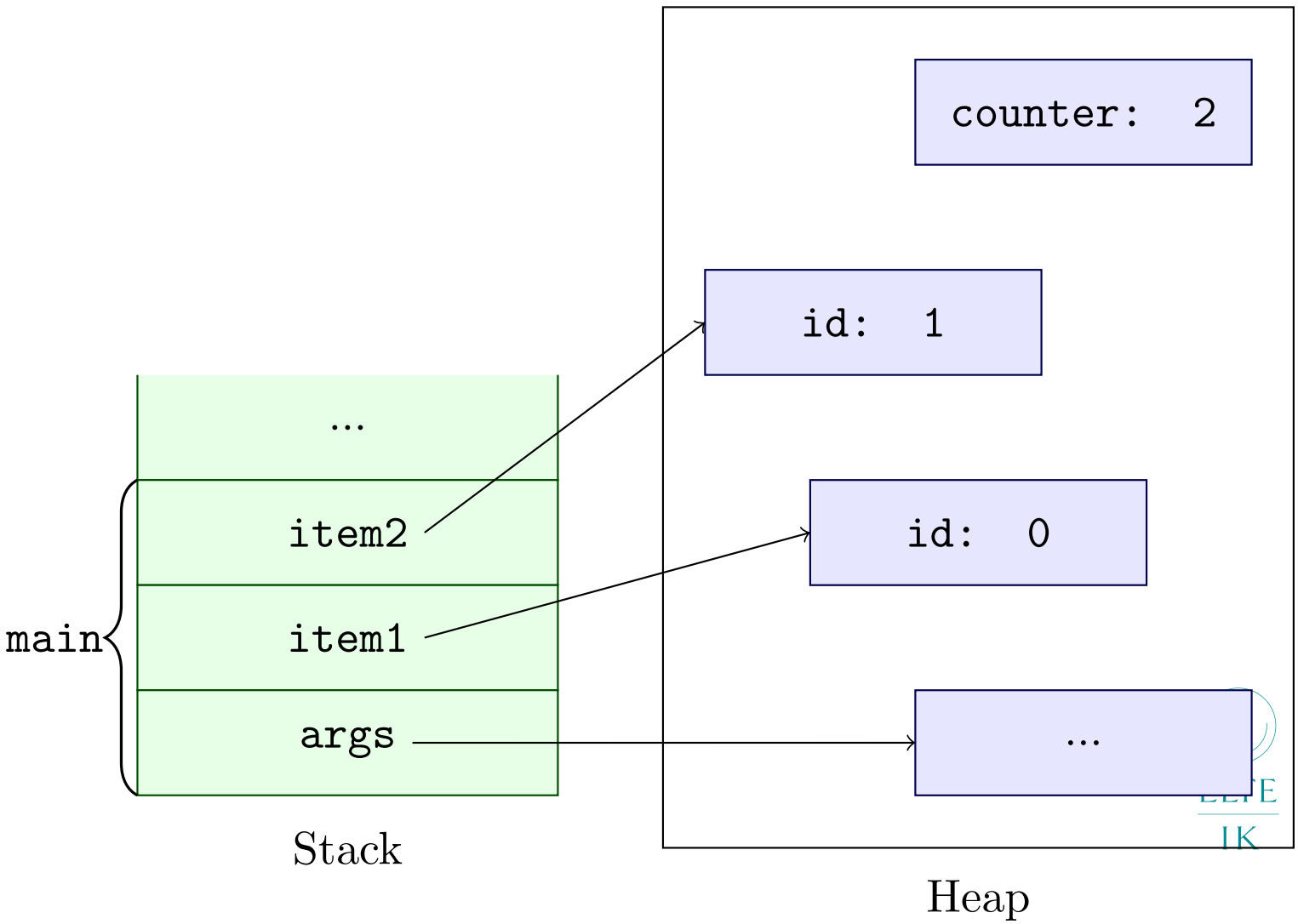
```
class Item {
    static int counter = 0;
}

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

Osztályszintű és példányszintű mezők

```
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(item1.counter); // érvényes, de csúf
    System.out.println(Item.counter);  // így jobb
  }
}
```

ELTE
IK

```
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 );
    }
}
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();
    }
}
```

ELTE
IK

Szemétgyűjtés

Feleslegessé vált objektumok felszabadítása

Helyes

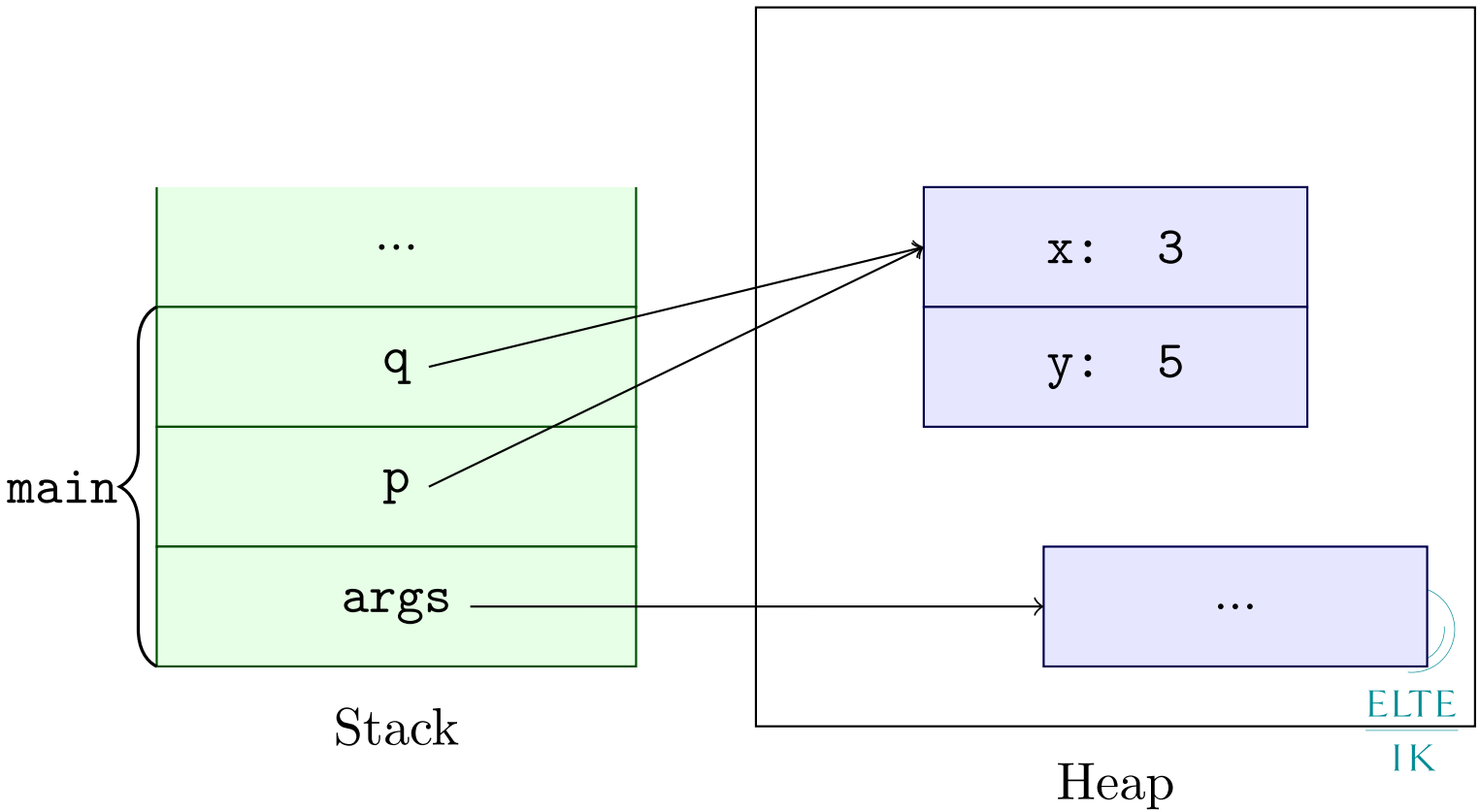
Csak olyat szabadít fel, amit már nem lehet elérni a programból

Teljes

Mindent felszabadít, amit nem lehet már elérni

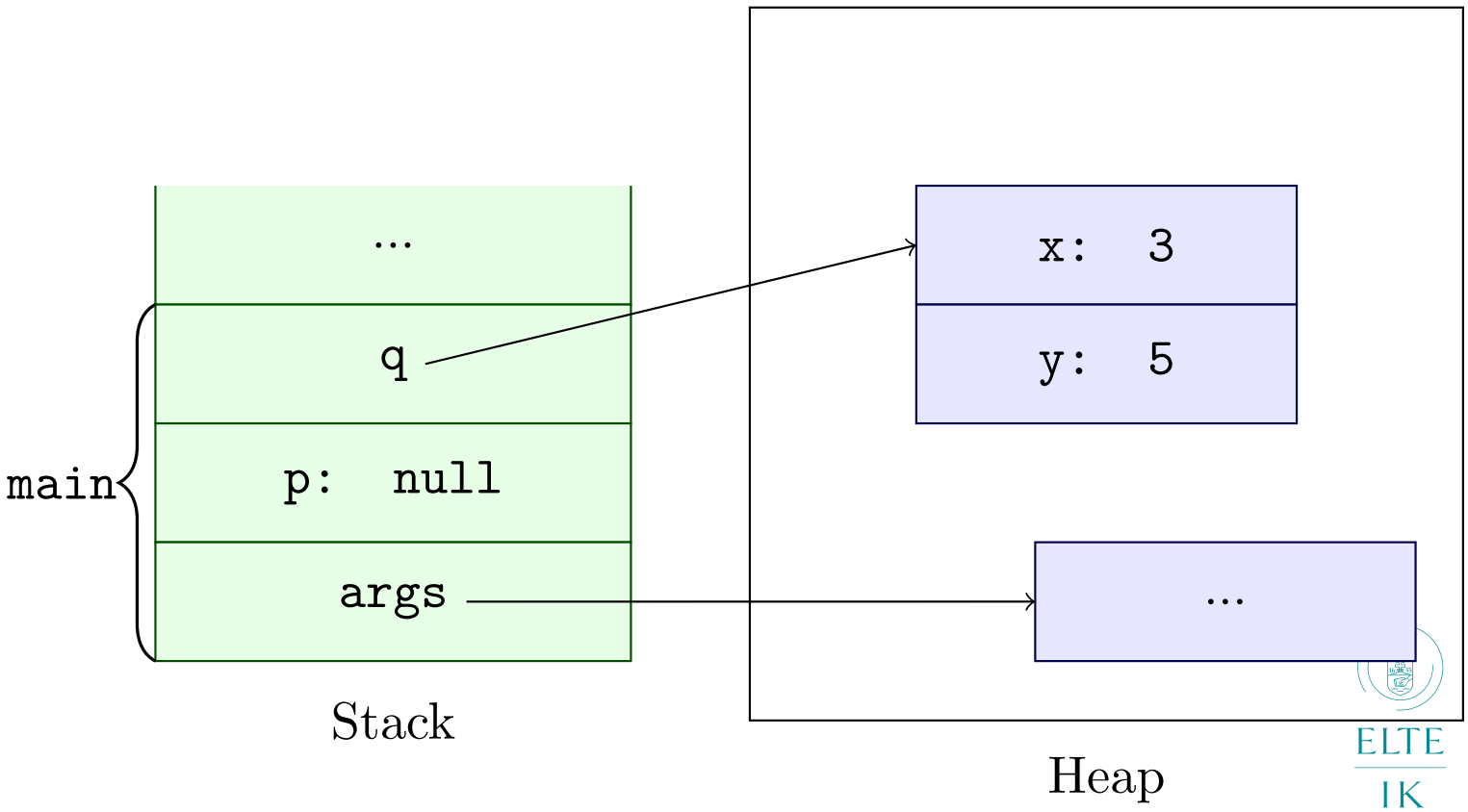
Még nem szabadítható fel

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



Még mindig nem szabadítható fel

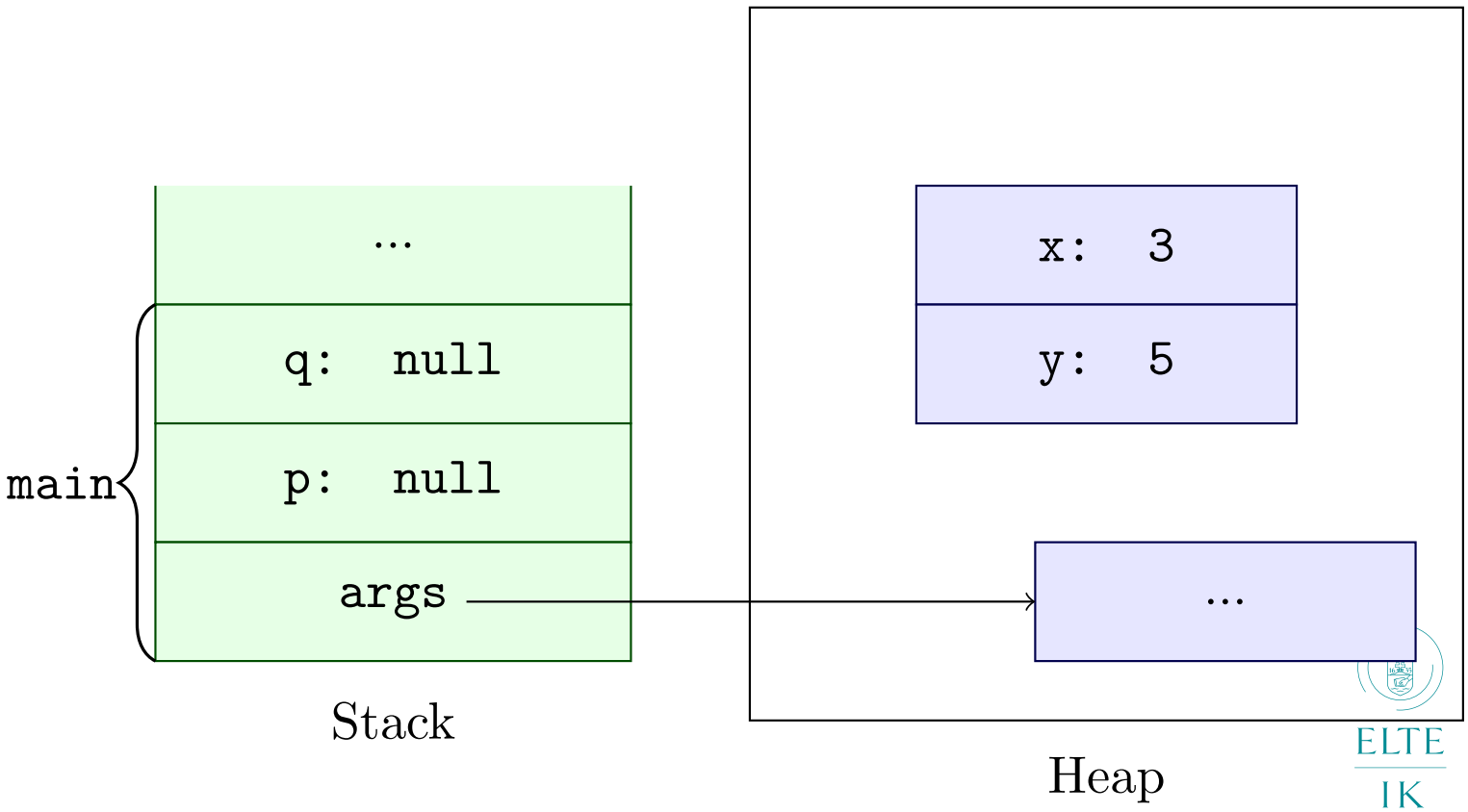
```
p = null;
```



ELTE IK

Már felszabadítható

```
q = null;
```



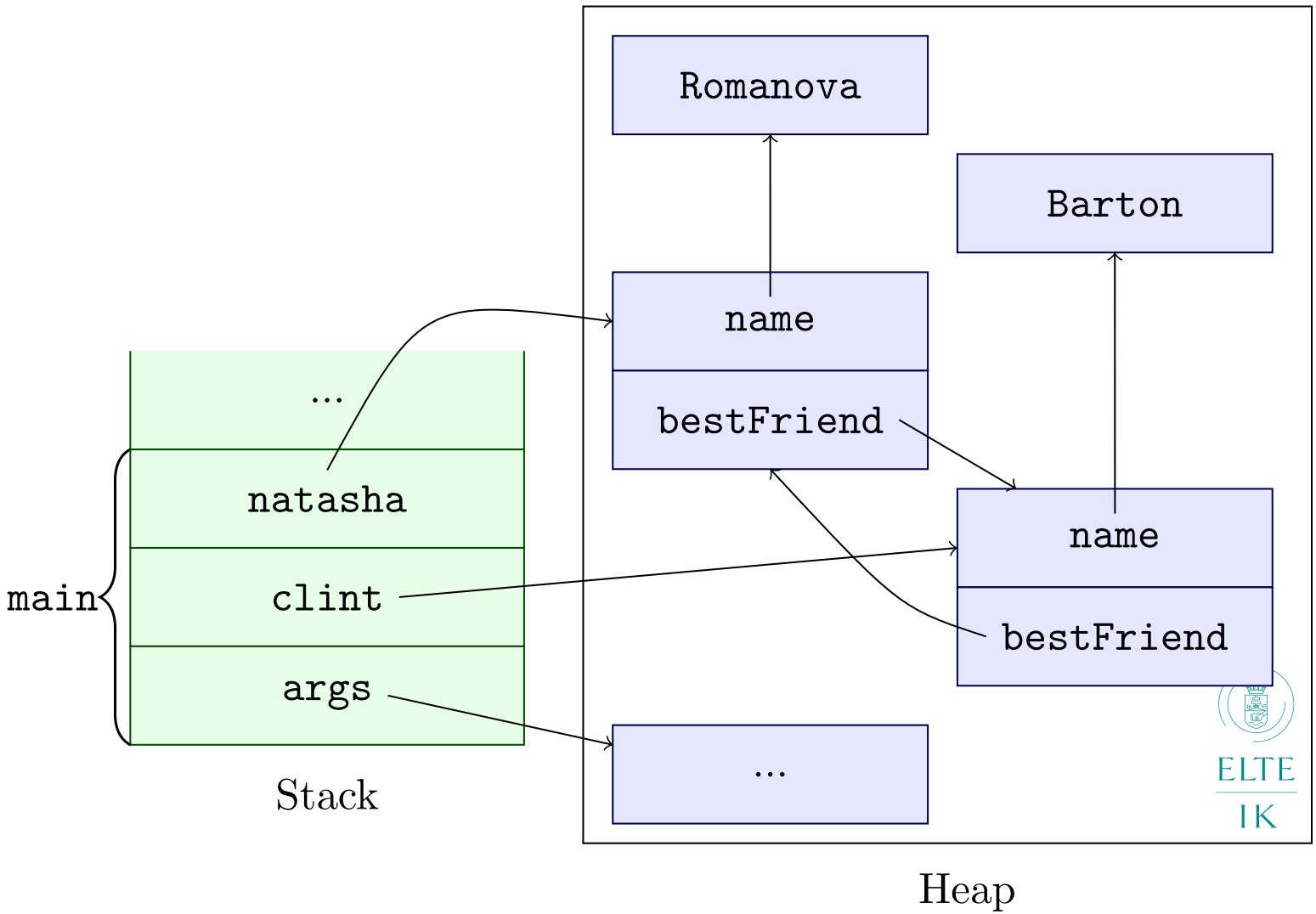
Bonyolultabb példa

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

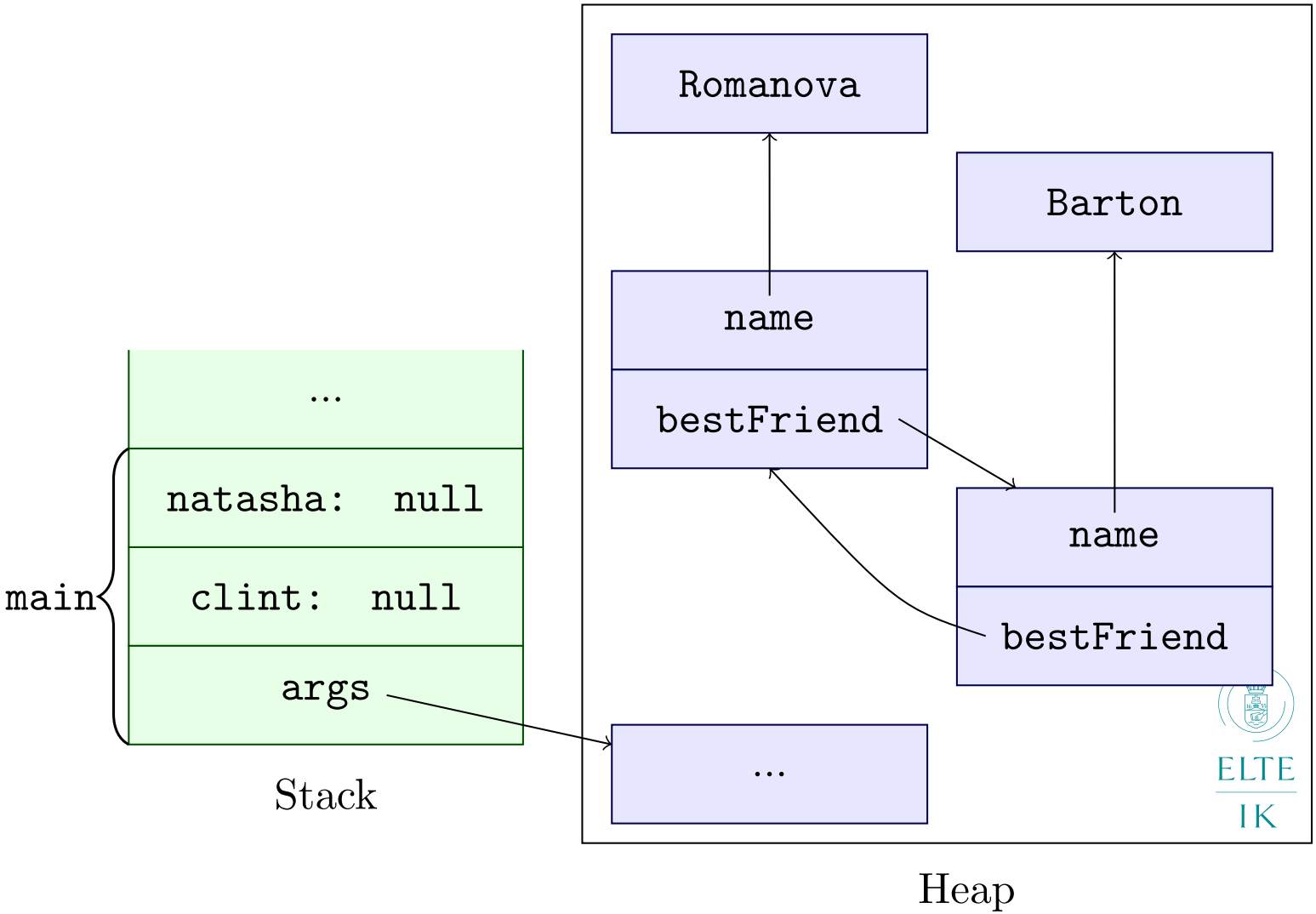
```
Hero clint   = new Hero();
Hero natasha = new Hero();

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

ELTE
IK

Hősök a memóriában



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

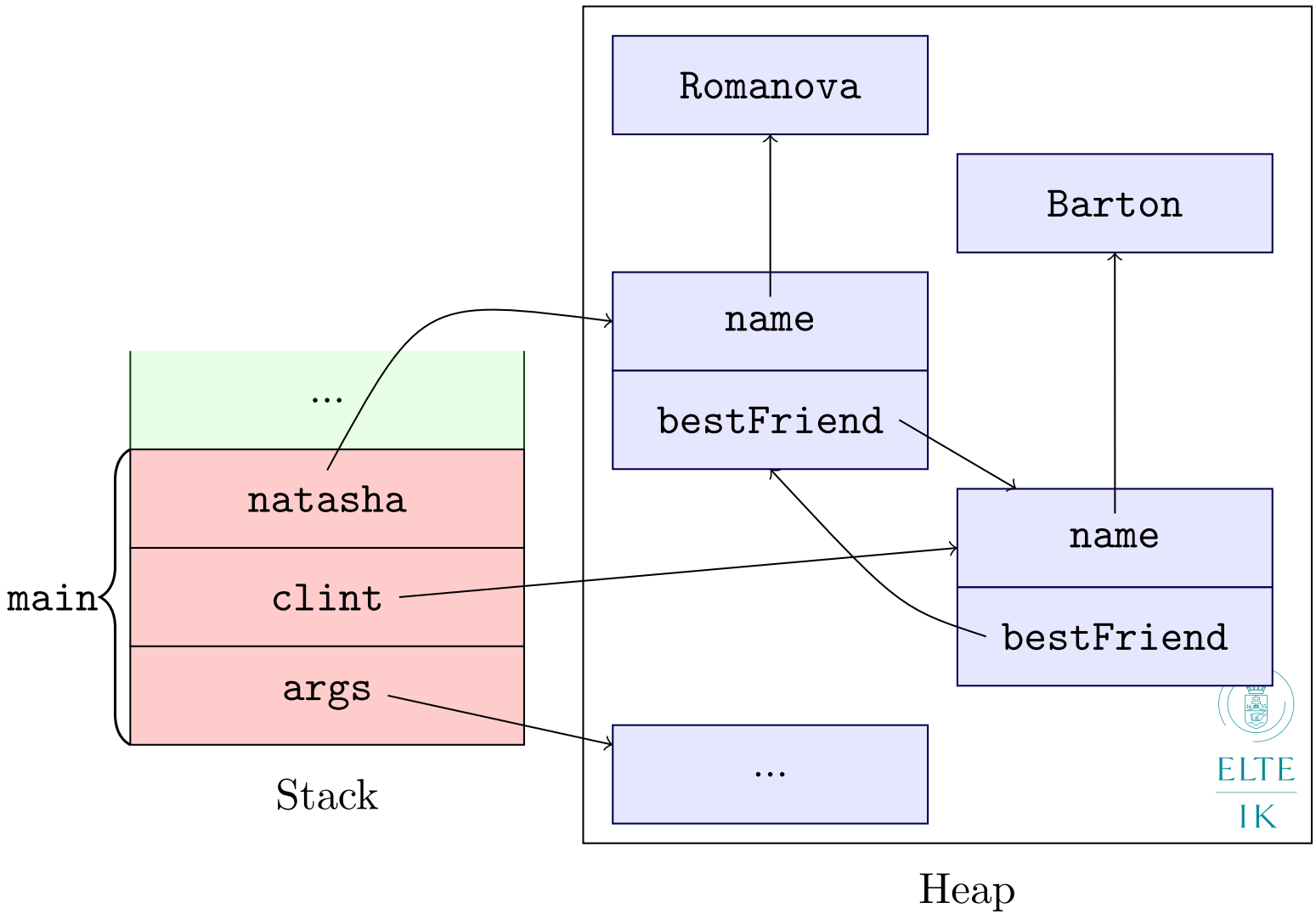


Mark-and-Sweep szemétgyűjtés

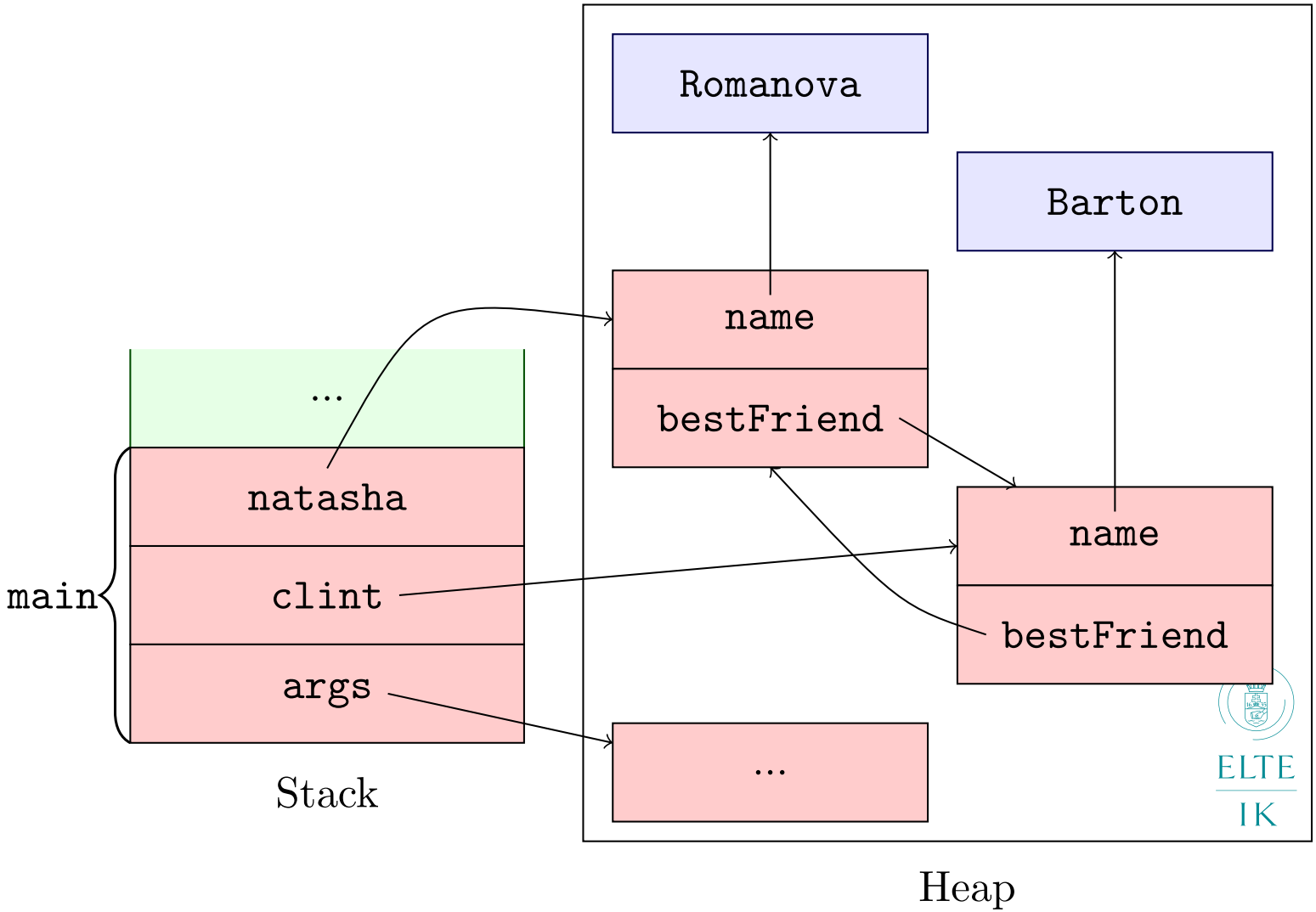
- Mark fázis
  - Kiindulunk a vermen lévő referenciákból
  - Megjelöljük a belőlük elérhető objektumokat
    - Megjelöljük az azokból elérhetőeke
    - ... amíg tudunk újabbat megjelölni (tranzitív lezárt)
- Sweep fázis
  - A jelöletlen objektumok felszabadíthatók

ELTE
IK

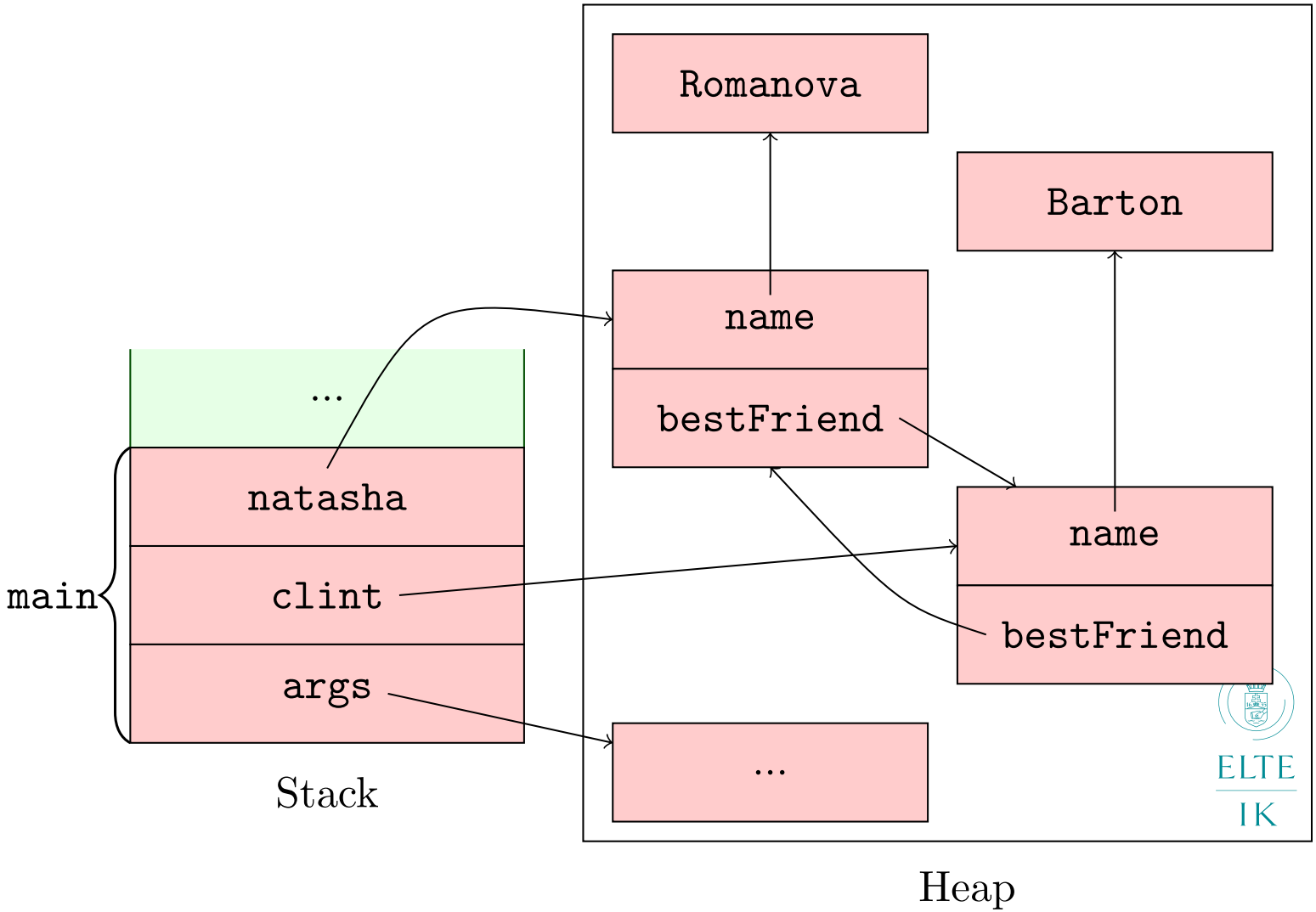
Mark-and-sweep: root set



Mark-and-sweep: propagálás

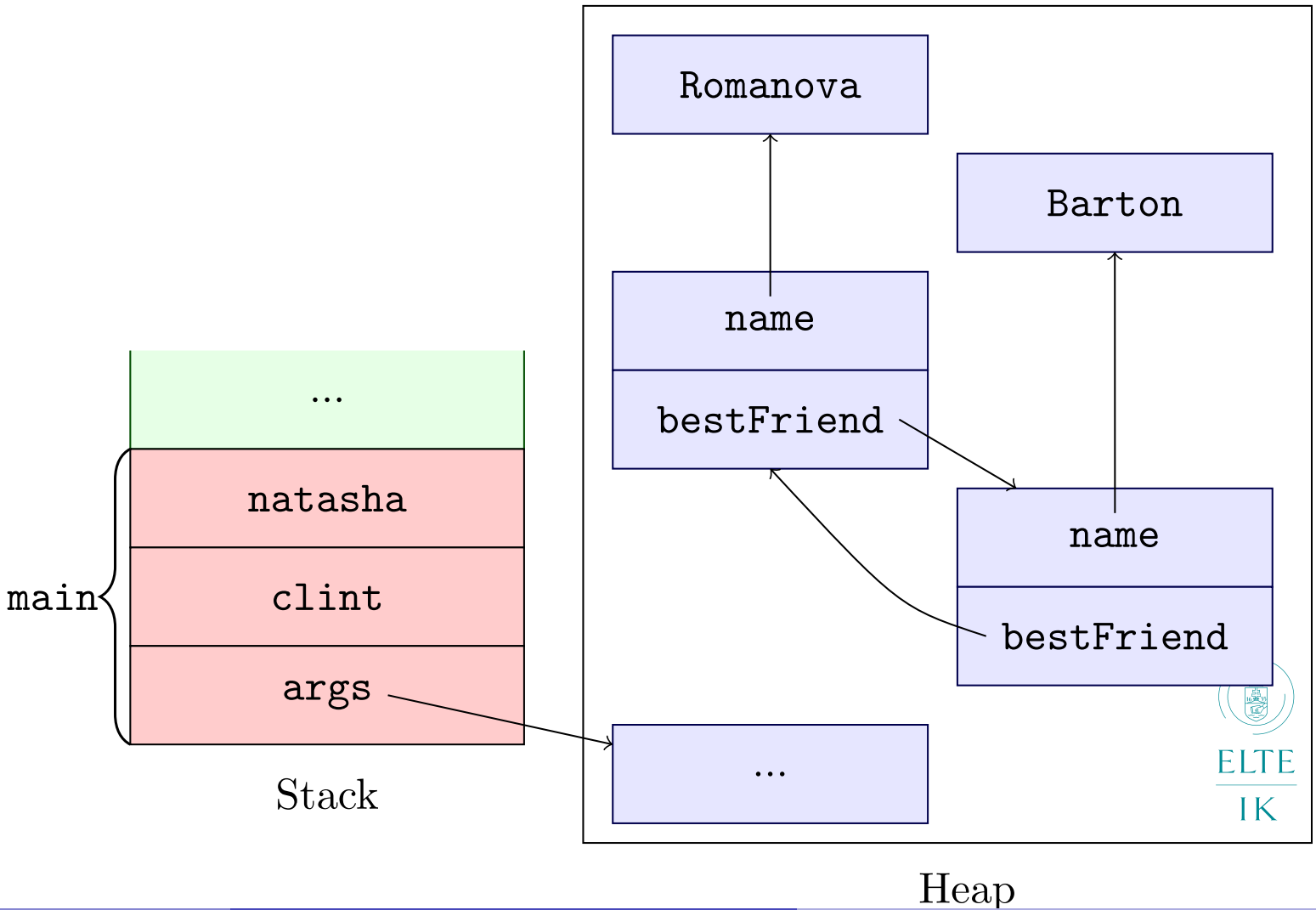


Mark-and-sweep: itt most mindegyik objektum elérhető



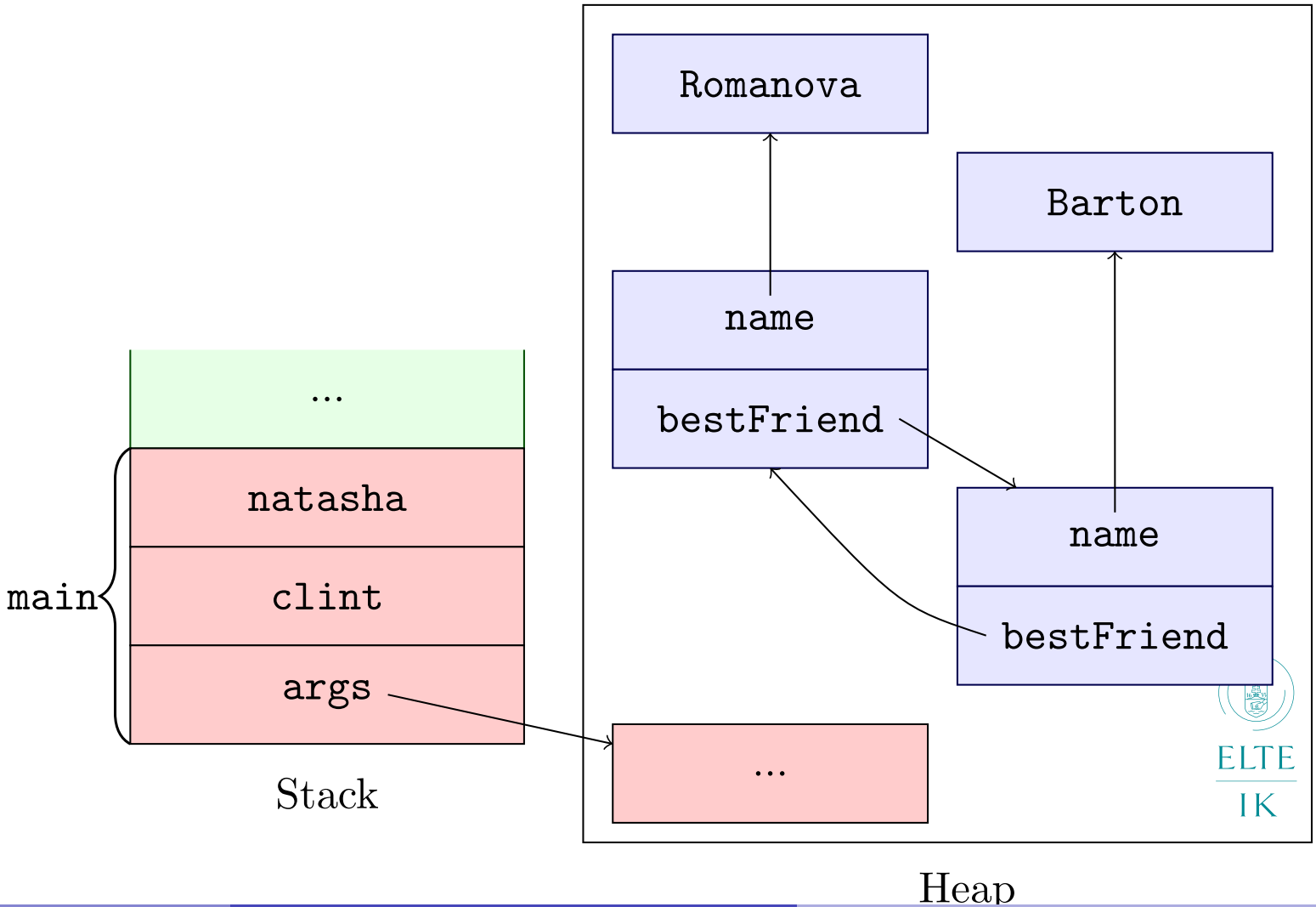
Mark-and-sweep: `natasha = clint = null;` ismét

```
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

ELTE
IK

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`

ELTE
IK

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

Statikus tagok importja

```
import static java.util.Arrays.sort;
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]);
    }

    sort(numbers);

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

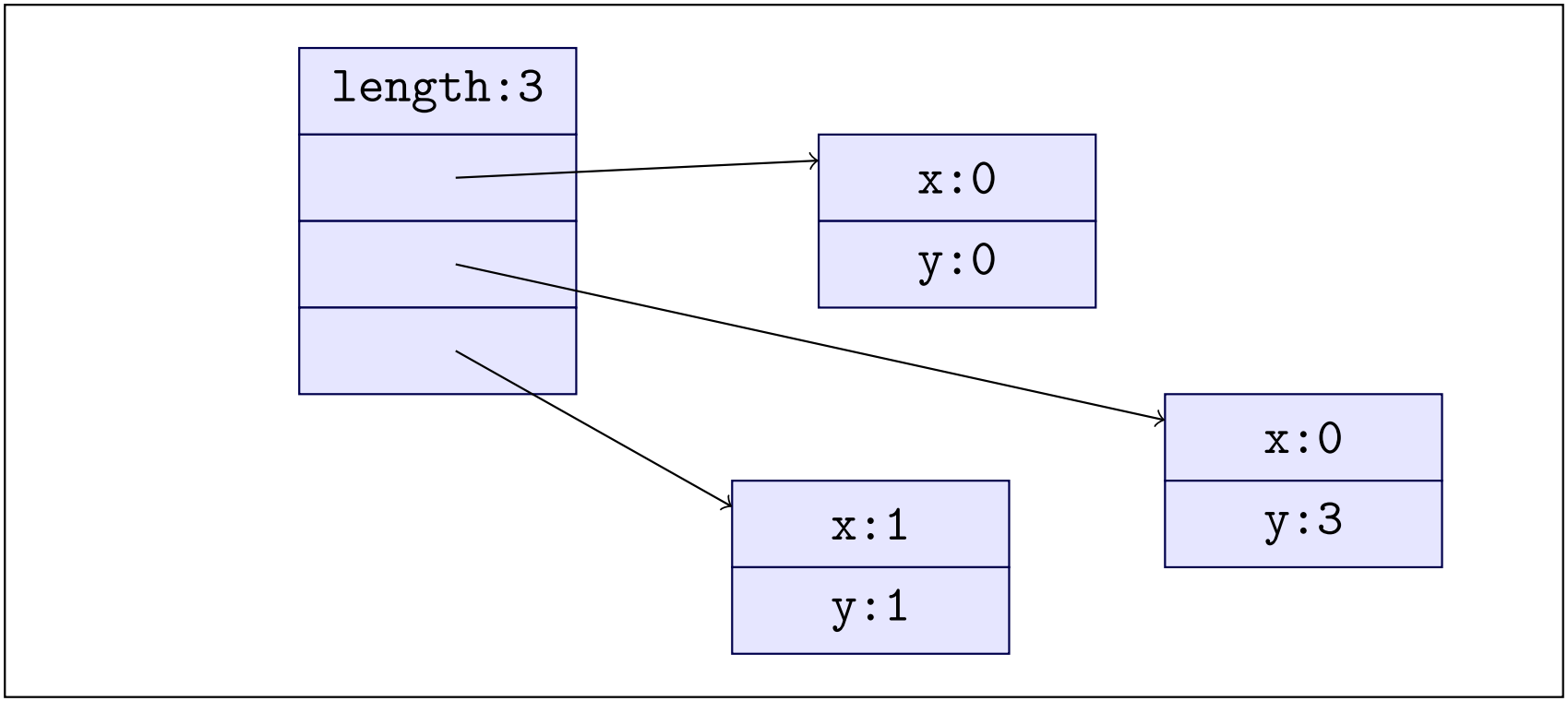
ELTE
IK

Referenciák tömbje

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

Referenciák tömbje

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



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);
```

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;
}
```

ELTE
IK

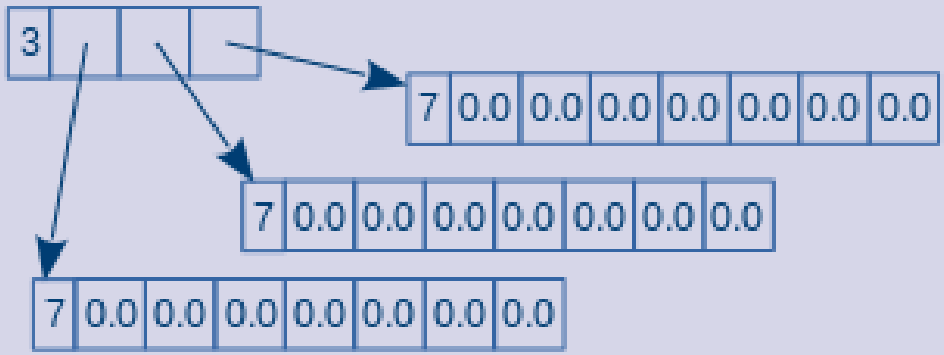
C versus Java

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];
```

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

Tömbök tömbjének tömbje Javában

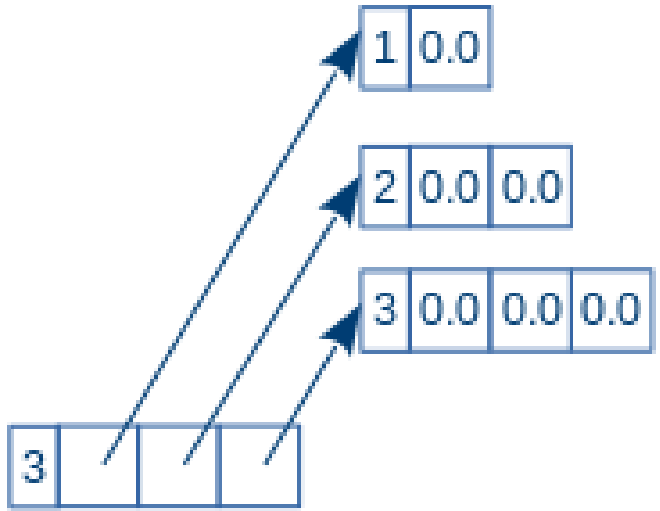
```
T[][][] t = new T[L][M][N];
```

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

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`)

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;       // also possible to "import static"
best = 3;                 // compilation error
int n = best;             // compilation error
int m = best.ordinal();   // 6
```

- 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

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

Programozási nyelvek – Java

Tesztelés

Kitlei Róbert

ELTE Eötvös Loránd Tudományegyetem

Egységteszt (Unit test)

- A program legkisebb, önálló részeinek kipróbálása
  - Egység lehet: metódus, **osztály**, komponens/modul
  - Nem egységteszt, ha külső függőségei vannak
    - Ilyen pl.: fájlrendszer, adatbázis, hálózat használata
- Kis, gyorsan lefutó, független tesztek
  - Futási időben működik
  - Fekete dobozos: az egység belső szerkezete nem ismert
    - Csak az osztály publikus interfészét (metódusait) használja
- Funkcionális helyességet tesztel: a lefutás az elvárt eredményt adja-e
  - Nem cél: hatékonyság tesztelése

Egységteszt: helyesség

- Nem *bizonyítja*, csak *alátámasztja* a helyességet
- Regressziók felfedése: hamar kiderül, ha hibás a kód
- Egyúttal dokumentálja, mi az elvárt működés
  - Együtt fejlődik a kóddal: ezt a fordítóprogram „érti'' és ellenőrzi
  - A szöveges dokumentáció elavulhat
- Lefedettség (code coverage)
- Sok hibát megelőz még fejlesztés alatt
  - Nagyobb munkaigény kezdetben
  - Olcsóbb lehet az utólagos hibajavításnál
    - Az éles rendszer jobban működik

Egységteszt: módszerek

- Tesztvezérelt fejlesztés (test driven development, TDD)
  1. Új teszteset hozzáadása, ami még "piros" (sikertelen)
  2. Kód írása/fejlesztése: minden teszteset legyen "zöld" (sikeres)
  3. A kód minőségének javítása (refaktorálás): minden "zöld"
- Egyéb tesztelési megközelítések
  - Naplózás, kiírások használata
  - Hibakeresés (debugging)
  - Összetettebb: integrációs ~, teljesítmény~, stressz/terhelési ~, automatizált ~, véletlenített/tulajdonság alapú ~, mock ~, folyamatos ~ (CI/CD), ...
  - Felhasználói élmény: elfogadási ~, biztonsági ~, használati ~, lokalizációs ~, ...
  - Formális helyességbizonyítás

Egységtesztelő: így használandó

- Egy tesztelő metódus egyetlen vizsgálatot tartalmaz
- A lehető legegyszerűbb szerkezet: ciklus, elágazás, véletlen, ... nélkül
- Saját kódot teszteljünk, ne könyvtárakat
- Lebegőpontos típusok tesztelése: az eredménynek lehet pontatlansága
  - Extra paraméter: tűréshatár (delta)
- Számítás adatainak struktúrája: egyszerűtől bonyolultig
  - `null`
  - üres szöveg, `0`
  - konstruktorhívás, majd getter
  - kis, pozitív értékek
  - egy-két lépéssel összeállított adatok
  - negatív/szokatlan/extrém értékek
    - pl. `Integer.MAX_VALUE` vagy `Double.MIN_VALUE`
    - kivételek
  - hosszabb "történet", több hívással

Egységtesztelés: FIRST

- Fast: μs-ms
- Isolated: egymástól és külvilágtól elkülönülő
- Repeatable: megismételhető
  - Nincsenek mellékhatások
  - Nincs nemdeterminisztikus futás
- Self-verifying: önellenőrző
  - Minden teszt elbukhat
  - Minden bukásnak pontosan egy oka lehet
- Timely: a kóddal együtt bővülnek/fejlődnek a tesztek
- vagy Thorough: lásd előző fólia

Outline

JUnit

- Java nyelvű megvalósítások közül a legnépszerűbb
- A jelenleg legújabb kiadás: JUnit 5, 1.9.2 verzió
- Innen letölthető a `jar` fájl
  - A letöltött fájl átnevezhető rövidebb névre, pl. `junit5.jar`
- Tesztelendő osztály: system under test (SUT)
  - Tegyük fel, hogy a `time.Time` osztályt teszteljük
  - A SUT kódja a `time/Time.java` fájlban van
  - A tesztelő kód a `time/TimeTest.java` fájlba kerül
- Fordítás: `javac -cp junit5.jar time/TimeTest.java`
- Futtatás: `java -jar junit5.jar -cp . -c time.TimeTest`

JUnit teszteset: Arrange-Act-Assert

```java
package time;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class DemoTest {
  @Test
  void testHour00_00() {
    // Step 1: Arrange
    Time sut = new Time(0, 0);
    // Step 2: Act
    int hour = sut.getHour();
    // Step 3: Assert
    assertEquals(0, hour);
  }
}
```

JUnit teszteset: Arrange-Act-Assert, röviden

```
package time;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class DemoTest {
  @Test
  void testHour00_00() {
    assertEquals(0, new Time(0, 0).getHour());
  }
}
```

JUnit teszteset kimenete

- Fontos: az elvárt érték az első paraméter
  - Ez mindig egy konstans legyen, ne számított érték

```
@Test
void wrongResultTest() { assertEquals(5, 2+2); }

org.opentest4j.AssertionFailedError: expected:<5> but was:<4>
  ... (sok, érdektelen információ)
  at testing.DemoTest.wrongResultTest(DemoTest.java:9)
  ... (még több sor)
```

JUnit teszteset kimenete

- Fontos: az elvárt érték az első paraméter
  - Ez mindig egy konstans legyen, ne számított érték

```
@Test
void wrongResultTest() { assertEquals(5, 2+2); }

org.opentest4j.AssertionFailedError: expected:<5> but was:<4>
  ... (sok, érdektelen információ)
  at testing.DemoTest.wrongResultTest(DemoTest.java:9)
  ... (még több sor)

@Test
void wrongOrderTest() { assertEquals(2+2, 5); }

org.opentest4j.AssertionFailedError: expected:<4> but was:<5>
  at testing.DemoTest.wrongOrderTest(DemoTest.java:9)
```

JUnit: ritkábban használatos eszközök

```
fail();

assertEquals("y", "x", "expected to be y");

assertEquals("y", "x", () -> "Also expected to be y");
```

```
... AssertionFailedError
    at time.JUnitDemoTest.testFail(JUnitDemoTest.java:19)
...: expected to be y ==> expected: <y> but was: <x>
    at time.JUnitDemoTest.testMessageV1(JUnitDemoTest.java:24)
...: Also expected to be y ==> expected: <y> but was: <x>
    at time.JUnitDemoTest.testMessageV2(JUnitDemoTest.java:29)
```

JUnit: ritkábban használatos eszközök

```java
@Test
public void testTrue() {
  assertTrue(2 + 2 == 4);
}

@Test
public void testFalse() {
  assertFalse("it's true" == "it's " + true);
}
```

- Az `assertEquals` jobb: precízebb a hibaüzenet
- Figyelem: a == nem helyes egyenlőségvizsgálat a `String` típuson!
  - Az ellenpárja, != szintén rossz

```
...: expected: <false> but was: <true>
    at time.JUnitDemoTest.testFalse(JUnitDemoTest.java:14)
```

JUnit: paraméterezett teszt: azonos működés több adaton

```java
@CsvSource("this is some text,4")
@ParameterizedTest
public void testSplit(String text, int partCount) {
    assertEquals(partCount, text.split(" ").length);
}

@DisplayName("Computing the Fibonacci numbers")
@ParameterizedTest(name = "fib({0}) = {1}")
@CsvSource({"13,6", "21,7"})
public void testFib(int expected, int num) {
  assertEquals(expected, Fibonacci.fib(num));
}
```

```
'-- Computing the Fibonacci numbers [OK]
  +-- fib(6) = 13 [OK]
  '-- fib(7) = 21 [OK]
```

JUnit: paraméterezett tesztek szövegblokkokkal

Forrás: JUnit 5 dokumentációja

```
@ParameterizedTest(name = "[{index}] {arguments}")
@CsvSource(useHeadersInDisplayName = true, textBlock = """
  FRUIT,         RANK
  apple,         1
  strawberry,    700_000
  'lemon, lime', 0xF1
  """)
public void testWithCsvSource(String fruit, int rank) {
  // ...
}
```

Kimenet:

```
[1] FRUIT = apple, RANK = 1
[2] FRUIT = strawberry, RANK = 700_000
[3] FRUIT = lemon, lime, RANK = 0xF1
```

JUnit: kivételek

```
@Test
public void testInvalidTime() {
  InvalidTimeException exception =
    assertThrows(InvalidTimeException.class, () -> {
      new Time(123, 456);
    };
    assertEquals("/ by zero", exception.getMessage());
  }
}
```

- `() -> { ... }`: a kivételt potenciálisan kiváltó kódrészlet ide kerül
- A `.class` tekinthető speciális adattagnak
- Itt megengedett két `assertX` írása is egy tesztelő metódusba
  - Sokszor nincs üzenet, akkor változó sem szükséges

JUnit: tömbök

- Tömbök tesztelése: külön `assertArrayEquals` művelettel
  - `assertEquals` nem jó
  - Más adatszerkezetek jól működnek

```
@Test
public void testFibArray() {
  int[] fibs = Fibonacci.fibsUpTo(6);
  assertArrayEquals(new int[] { 1, 1, 2, 3, 5, 8 }, fibs);
}
```

JUnit: életciklus

```
public class TimeTest {
  private Time time;

  @BeforeEach
  public void beforeEach() {
    time = new Time(12, 34);
  }

  @Test void test1() { assertEquals(12, time.getHour()); }
  @Test void test2() { assertEquals(34, time.getMin());  }
  @Test void test3() { assertEquals(35, time.inc().getHour()); }
}
```

- `@BeforeEach`: tesztesetek ismétlődő adatainak közös beállítása
  - A tesztesetek nem zavarják egymást, mert mindig újrainicializál
- `@AfterEach`: pl. átmeneti fájlok törlésére
- `@BeforeAll`, `@AfterAll`: ritkán használatos

Outline

CheckThat

- A szokásos JUnit tesztek a kód funkcionalitását vizsgálják
- Ez az eszköz a kód szerkezetét ellenőrzi
- Használata intuitív
- A megvalósító kód túlmutat a félév anyagán, nem kell megérteni

CheckThat példa

```
package time;

import static check.CheckThat.Condition.*;
import check.CheckThat;

import org.junit.jupiter.api.Test;

public class StructureTest01_Time {
  @Test
  public void test1() {
    CheckThat...
  }

  ...
}
```

CheckThat példa

```
CheckThat.theClass("time.Time")
  .thatIs(NOT_ABSTRACT, PUBLIC)
  .hasConstructorWithParams("int", "int")
    .thatIs(PUBLIC);
}

CheckThat.theClass("time.Time")
  .hasFieldOfType("hour", "int")
    .thatIs(PRIVATE, NOT_STATIC, MODIFIABLE)
    .has(GETTER, SETTER);

CheckThat.theClass("time.Time")
  .hasMethodWithParams("getEarlier", "Time")
    .thatIs(PUBLIC, NOT_STATIC)
    .thatReturns("Time");
```

CheckThat hibaüzenetek

```
org.opentest4j.MultipleFailuresError: Multiple Failures (1 failure)
    ...: Nincsen megfelelő GETTER metódus
                                ehhez az adattaghoz: Time.hour
```

További üzenetek:

```
...: A Time.hour visszatérése nem megfelelő
...: A Time.hour láthatósága nem megfelelő
```

- Egy változóval angolra is állítható

CheckThat használata

```
package time;
import org.junit.platform.suite.api.*;

@Suite
@SelectClasses({
  StructureTest01_Time.class,
  StructureTest02_WorldTimes.class
  ,TimeTest.class
  ,WorldTimesTest.class   // (*)
})
public class TestSuite {}
```

- Fordítás: `javac -cp junit5.jar time/TimeTestSuite.java`
- Futtatás: `java -jar junit5.jar -cp . -c time.TimeTestSuite`
- A tesztelő kódhoz nem kell hozzányúlni
  - Ha még csak a `Time` osztály van készen, `(*)` kikommentezendő

CheckThat használata, elkülönülő tesztelő kód

```
root
+ project
  + src
    + time
      + Time.java
  + test
    + time
      + StructureTest01_Time.java
      + StructureTest02_WorldTimes.java
      + TestSuite.java
      + TimeTest.java
```

```
root
+ tester
  + junit5.jar
  + check
    + CheckThat.java
```

- Továbbra is ugyanabban a csomagban van a SUT és a tesztelő
- Fordítás: `javac -cp ../tester/junit5.jar;../tester test/time/*.java src/time/*.java`
- Futtatás: `java -jar ../tester/junit5.jar -cp ../tester;test;src -c time.TestSuite`

CheckThat használata, elkülönülő tesztelő kód

```
'-- JUnit Platform Suite [OK]
  '-- TestSuite [OK]
    '-- JUnit Jupiter [OK]
      '-- StructureTest01_Time [OK]
        +-- test1() [OK]
        +-- test2() [OK]
        '-- test3() [OK]
```

Programozási nyelvek – Java

Típusok

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

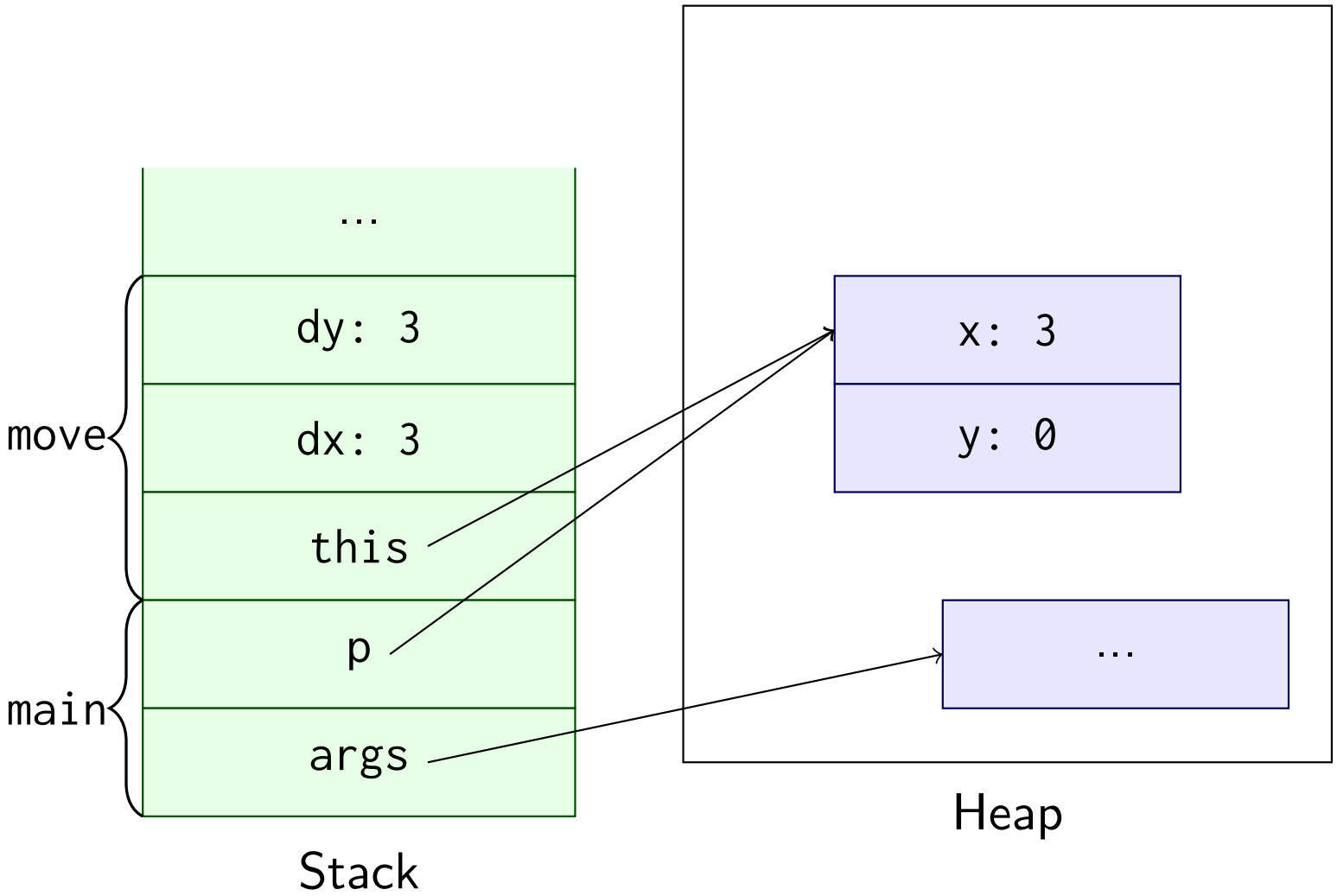
Outline

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)

Outline

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
            ...
        }
    }
}
```

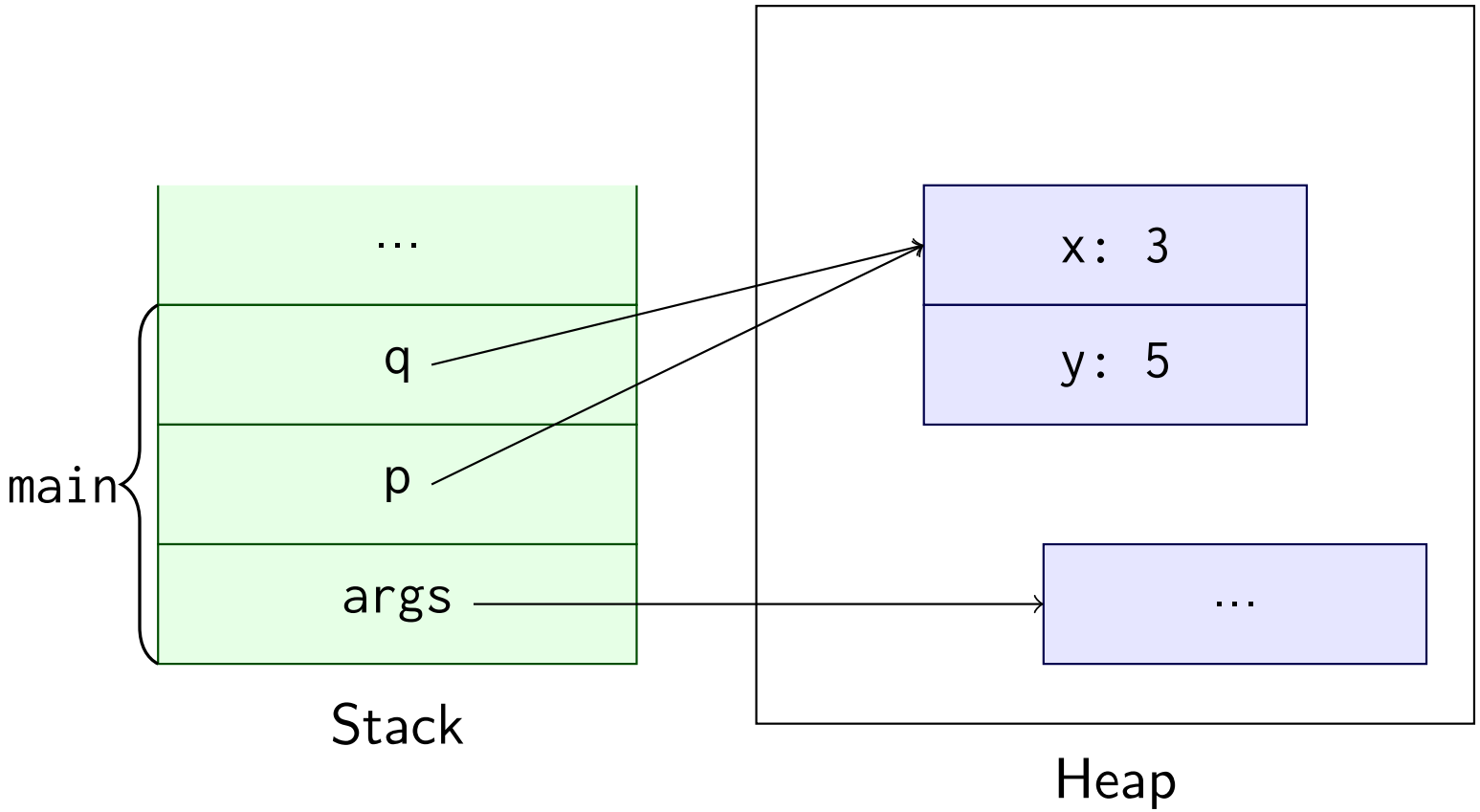
Objektumok élettartama

- 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
```

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

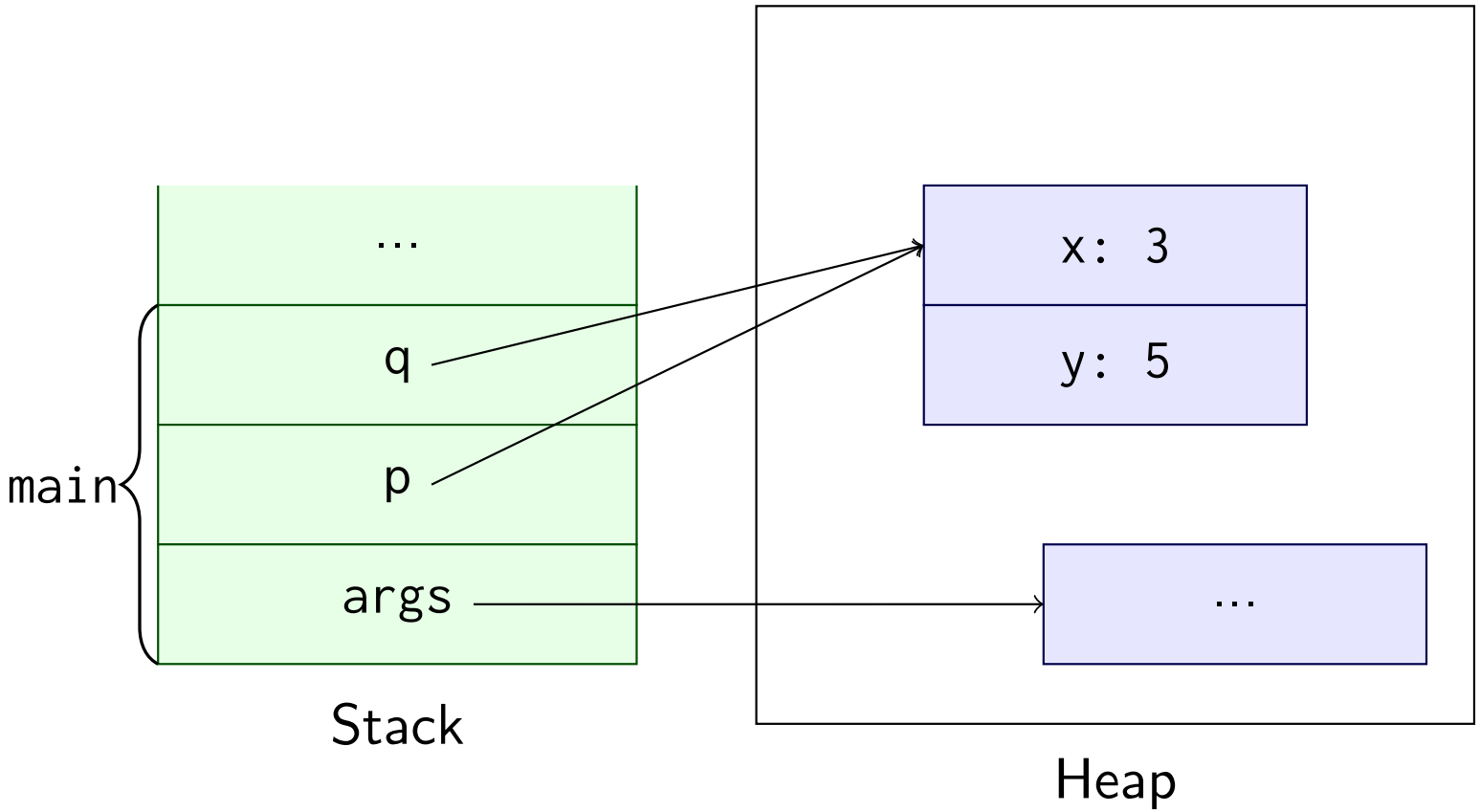
Csak olyat szabadít fel, amit már nem lehet elérni a programból

Teljes

Mindent felszabadít, amit nem lehet már elérni

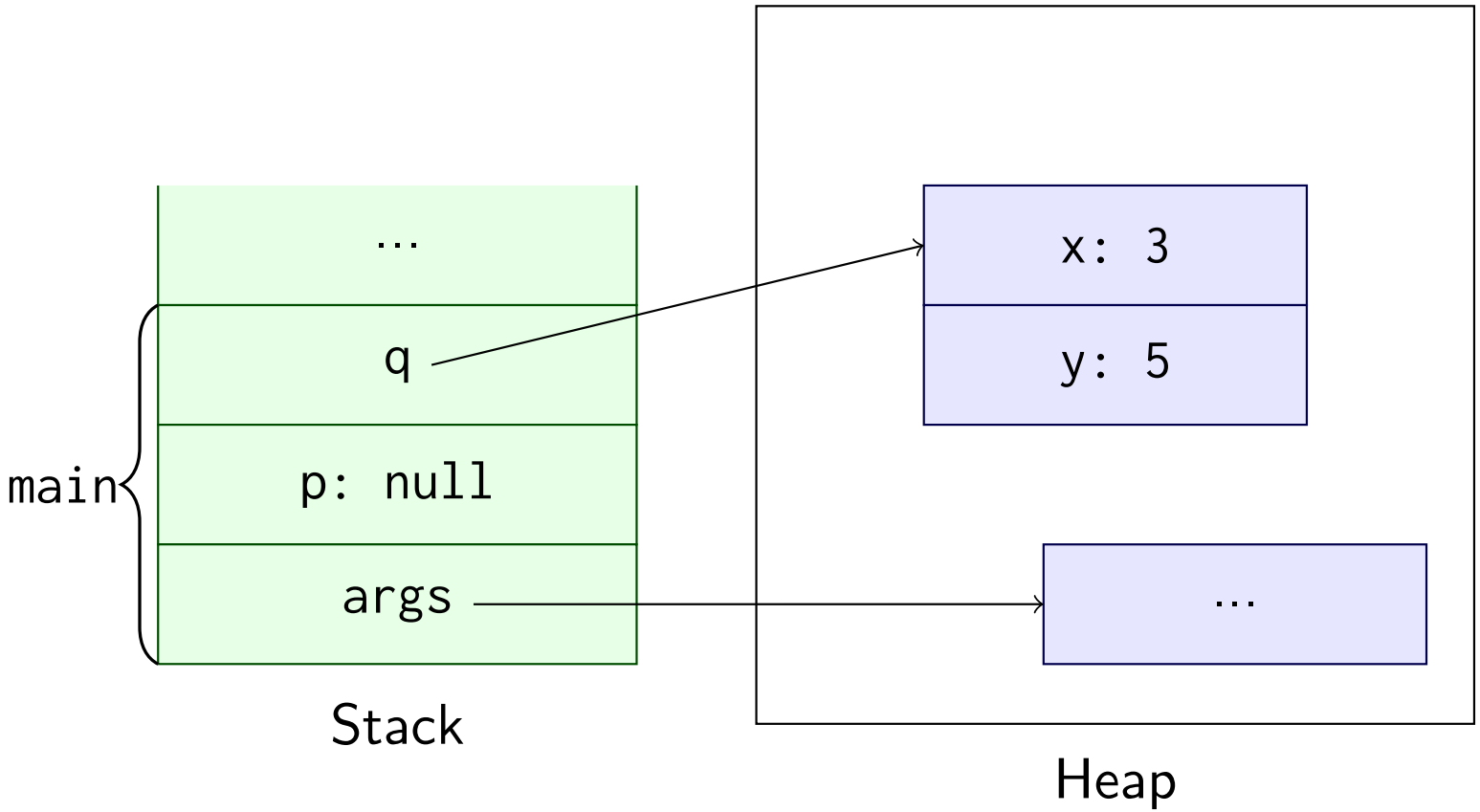
Még nem szabadítható fel

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



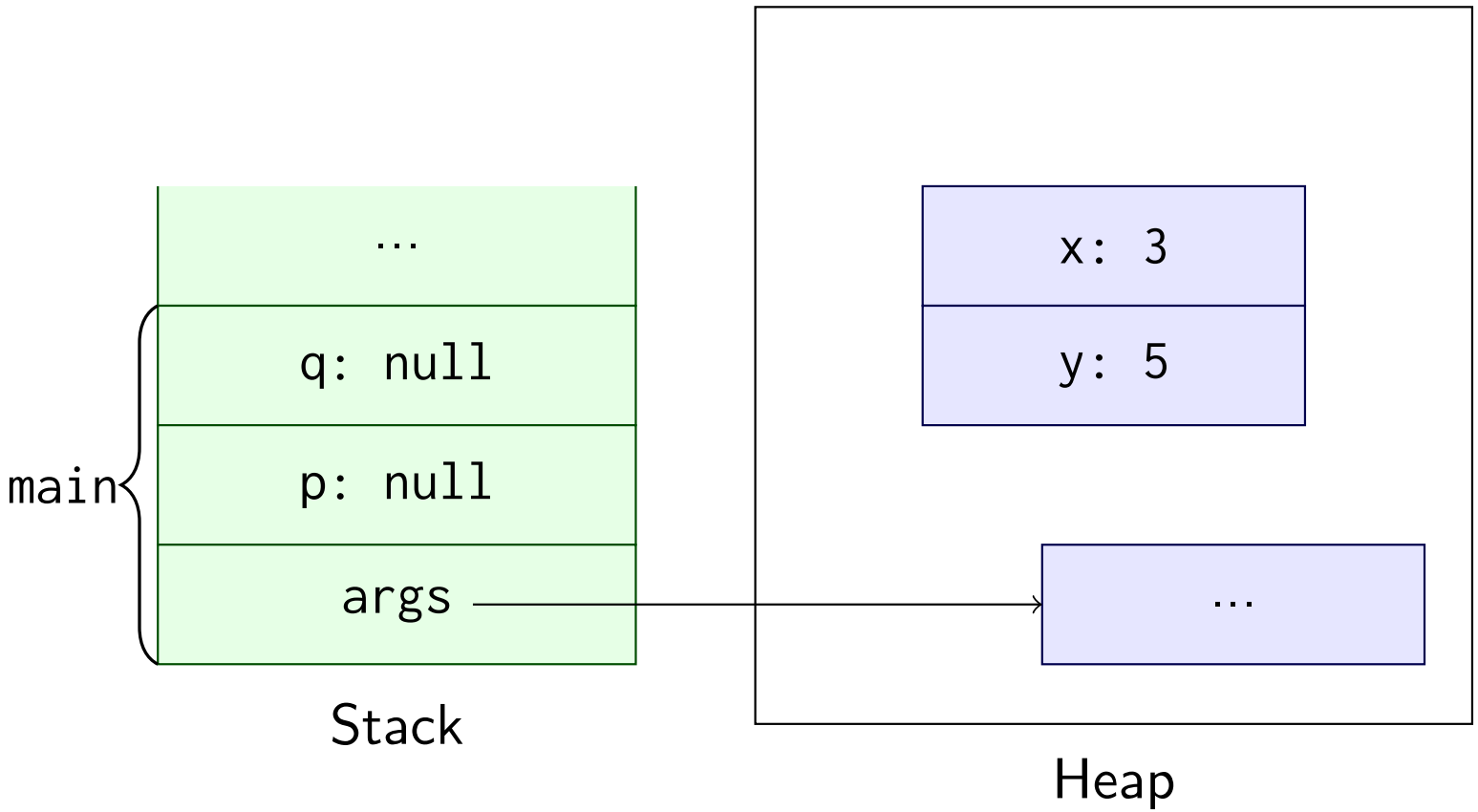
Még mindig nem szabadítható fel

```
p = null;
```



Már felszabadítható

```
q = null;
```

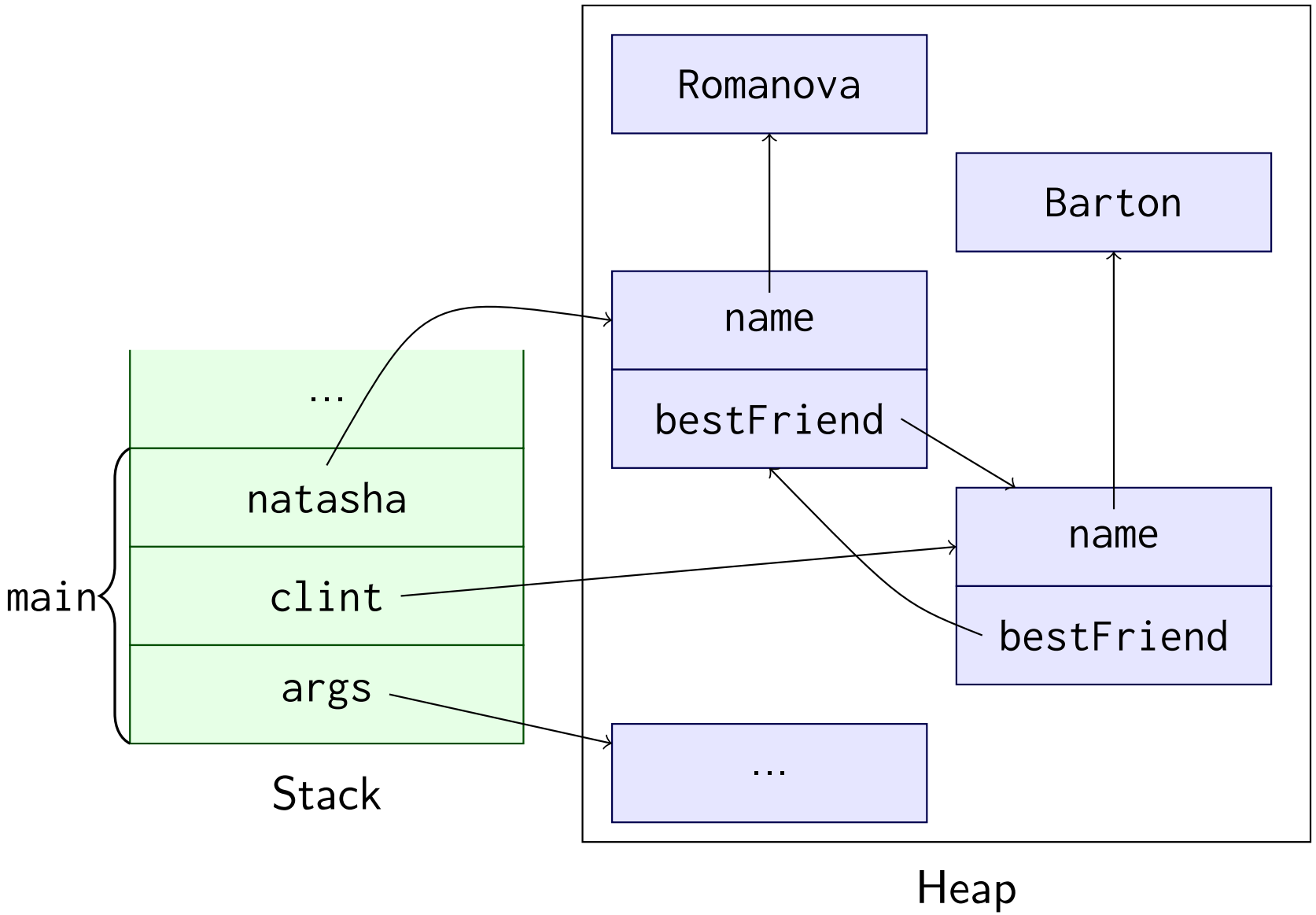


Bonyolultabb példa

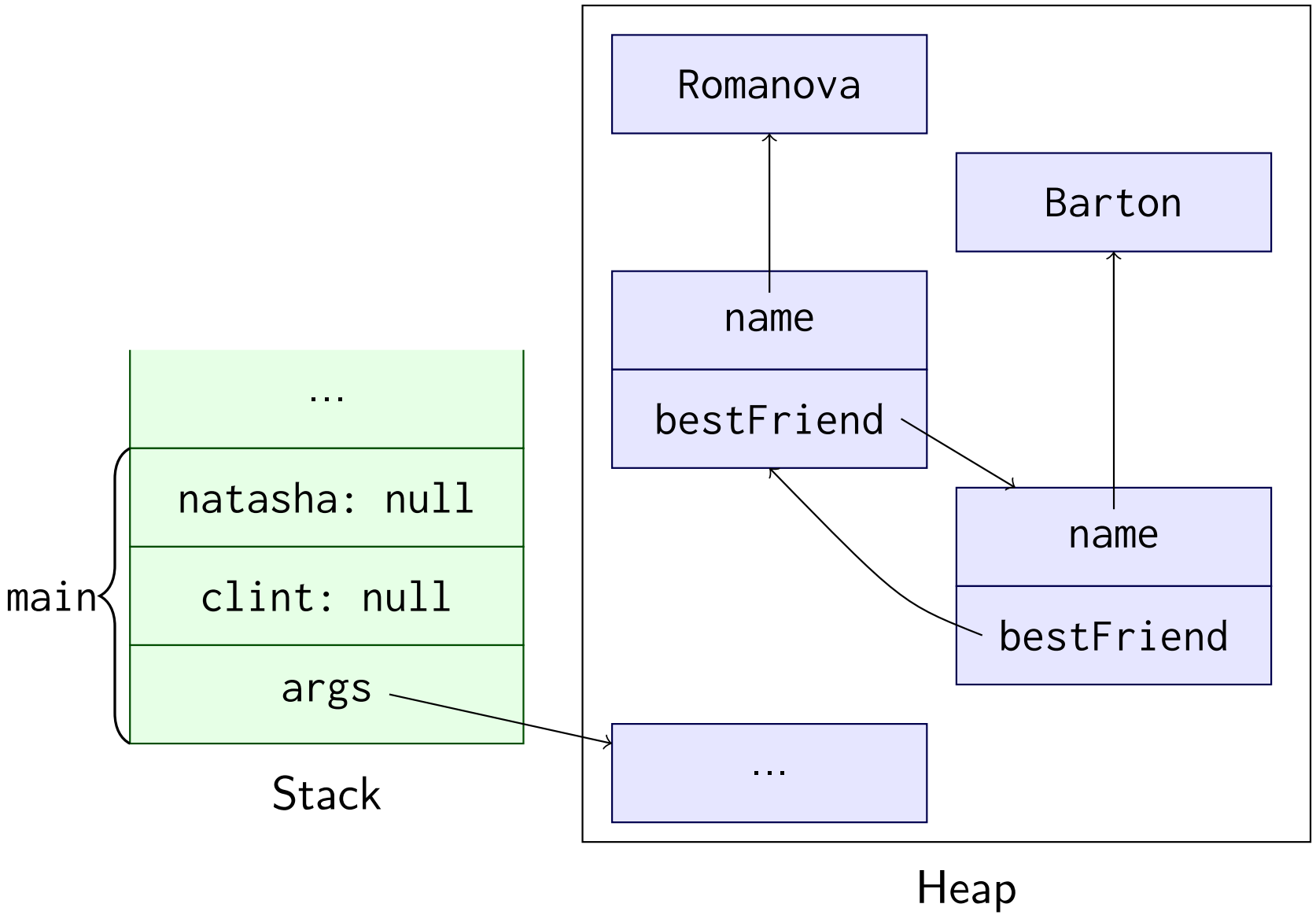
```
class Hero {
    String name;
    Hero bestFriend;
}
```

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

Hősök a memóriában



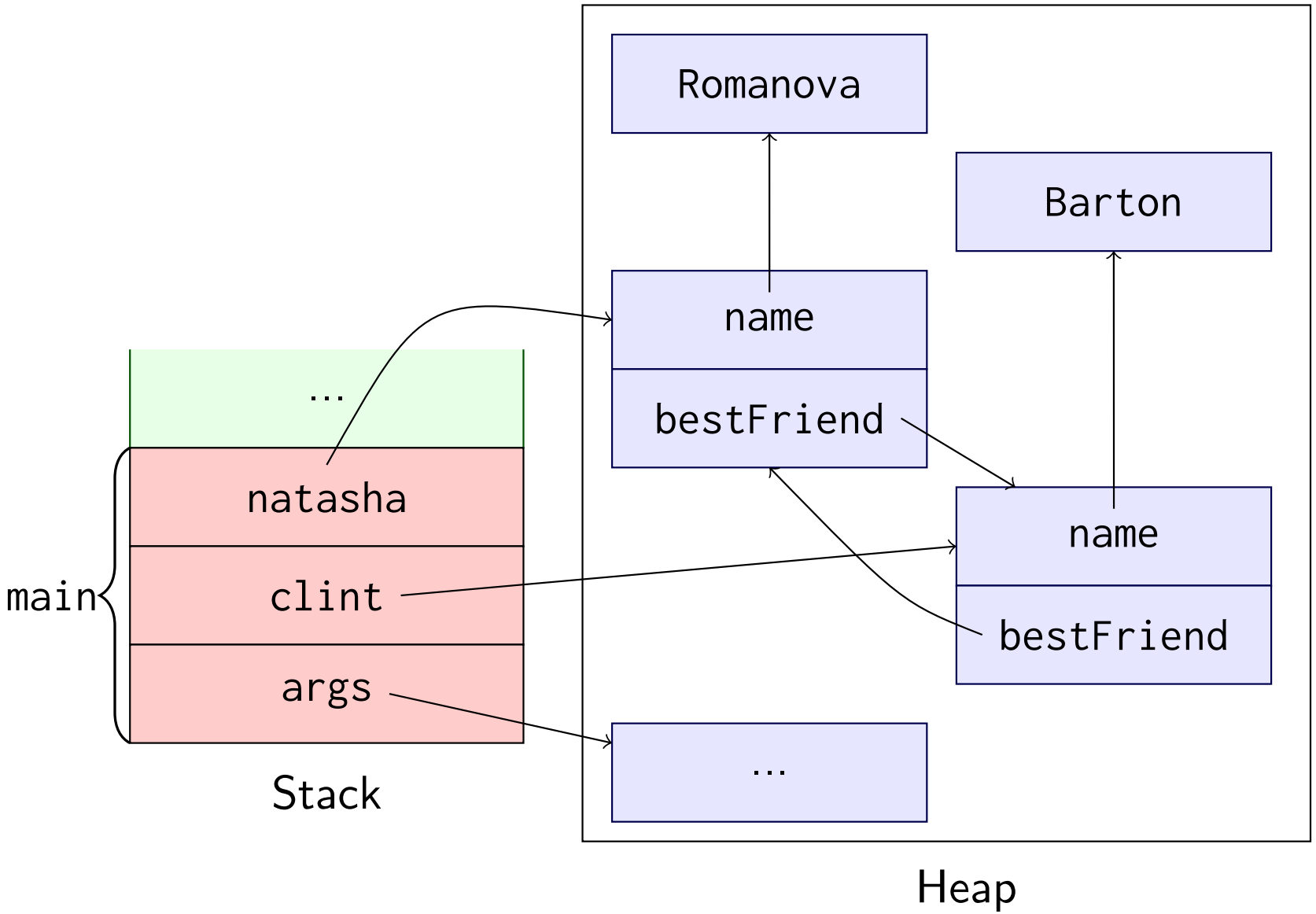
natasha = clint = null;



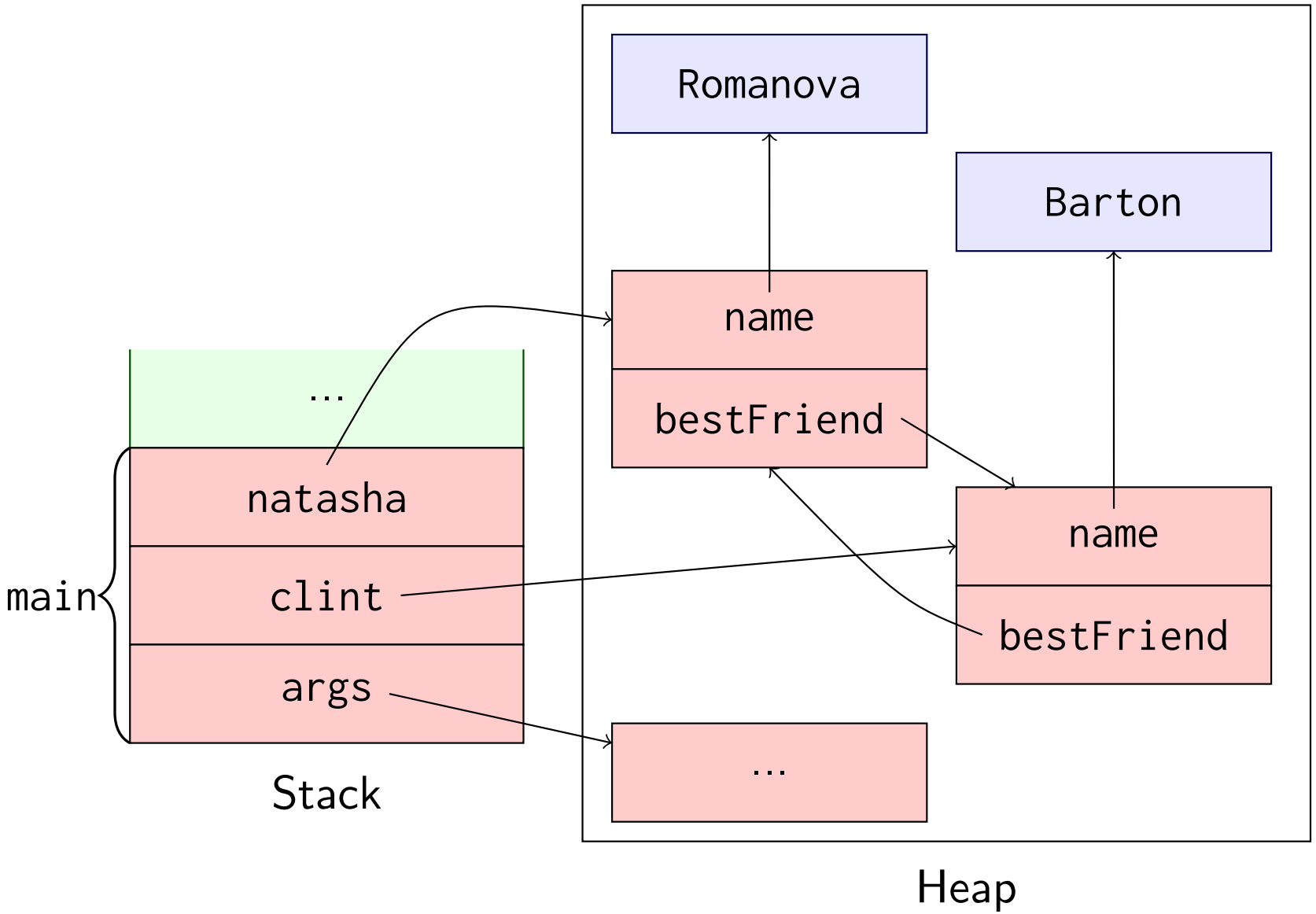
Mark-and-Sweep szemétgyűjtés

- Mark fázis
  - Kiindulunk a vermen lévő referenciákból
  - Megjelöljük a belőlük elérhető objektumokat
    - Megjelöljük a megjelöltekből elérhető objektumokat
    - ... amíg tudunk újabbat megjelölni (tranzitív lezárt)
- Sweep fázis
  - A jelöletlen objektumok felszabadíthatók

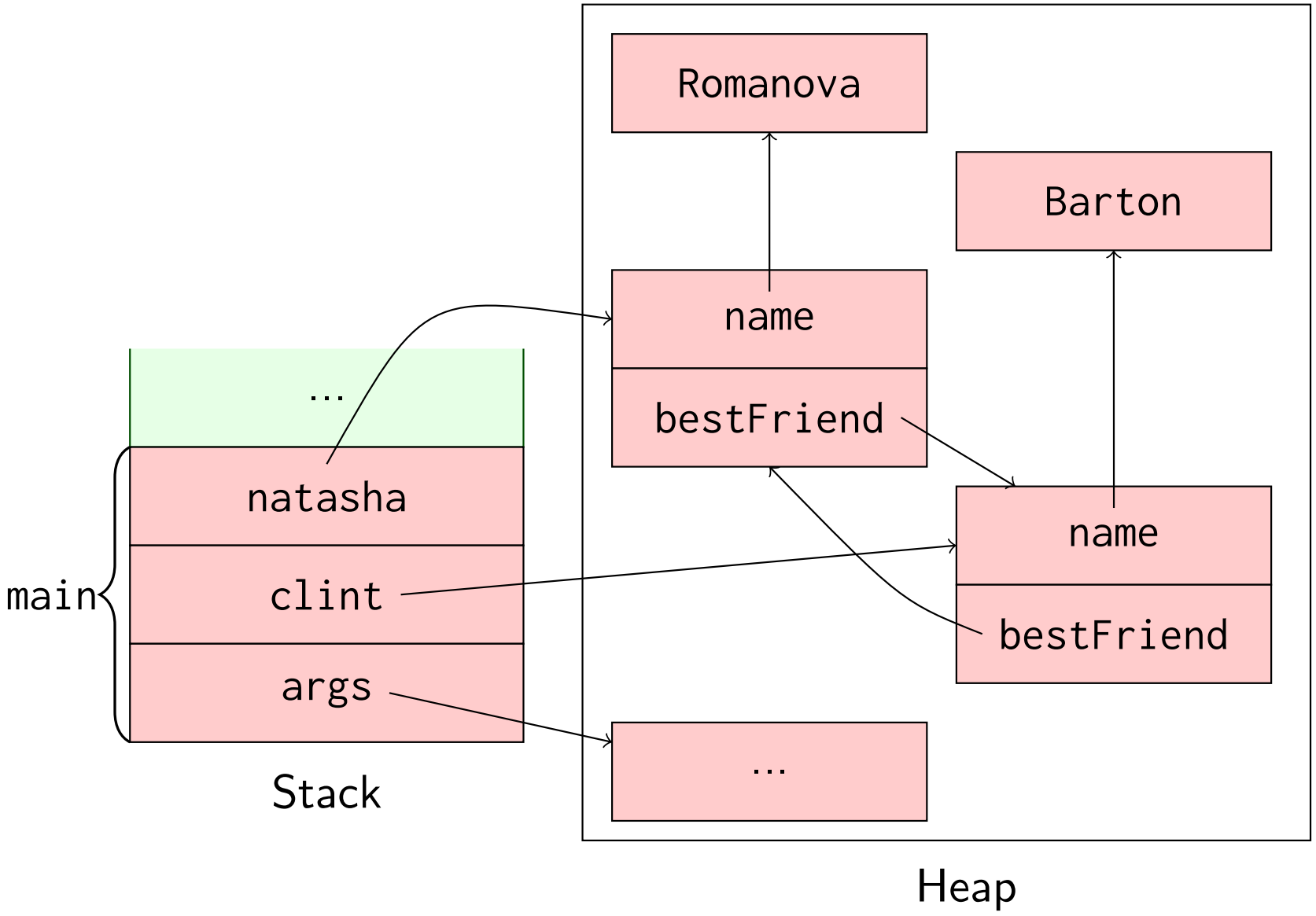
Mark-and-sweep: root set



Mark-and-sweep: propagálás

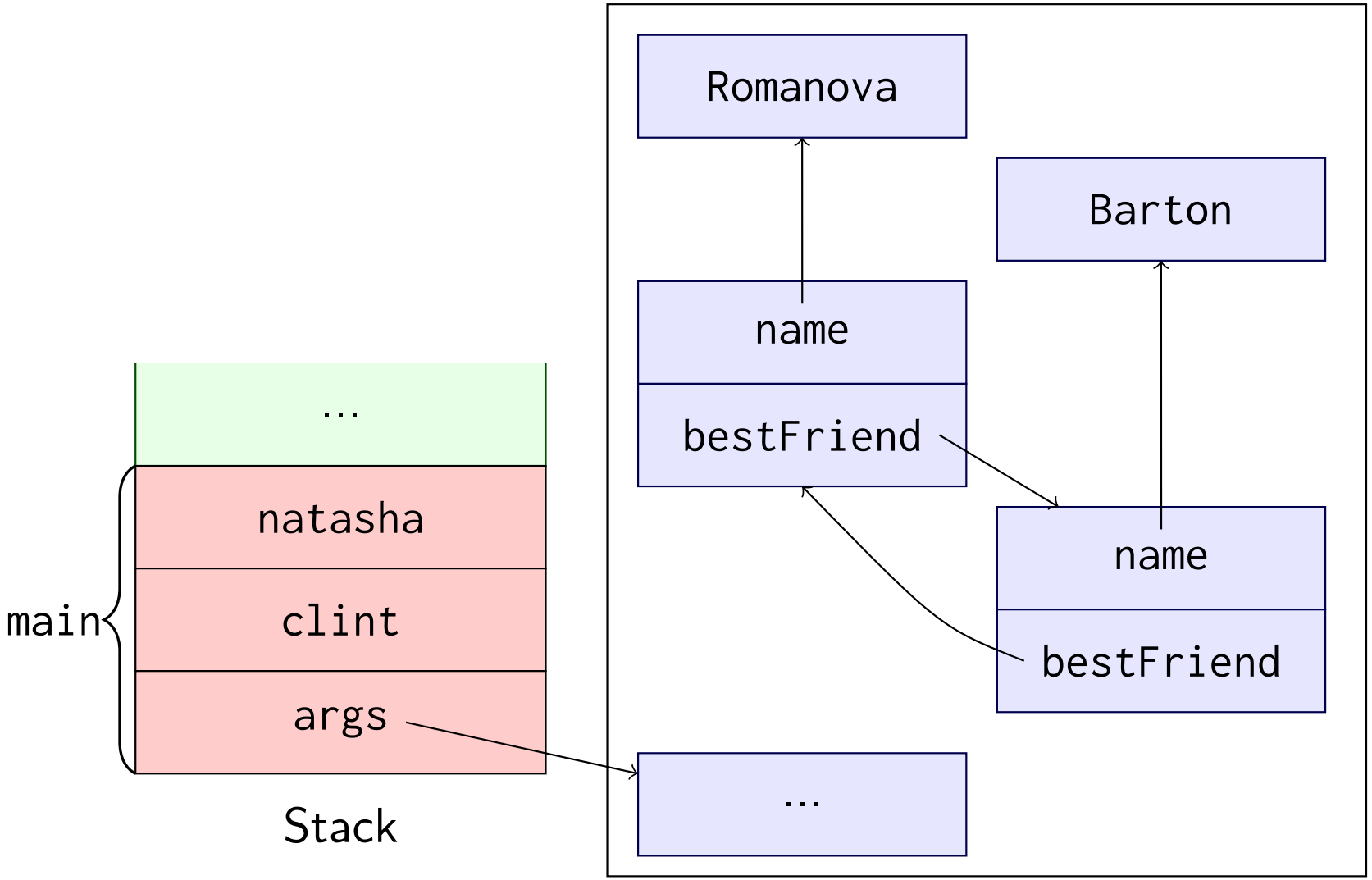


Mark-and-sweep: itt most mindegyik objektum elérhető



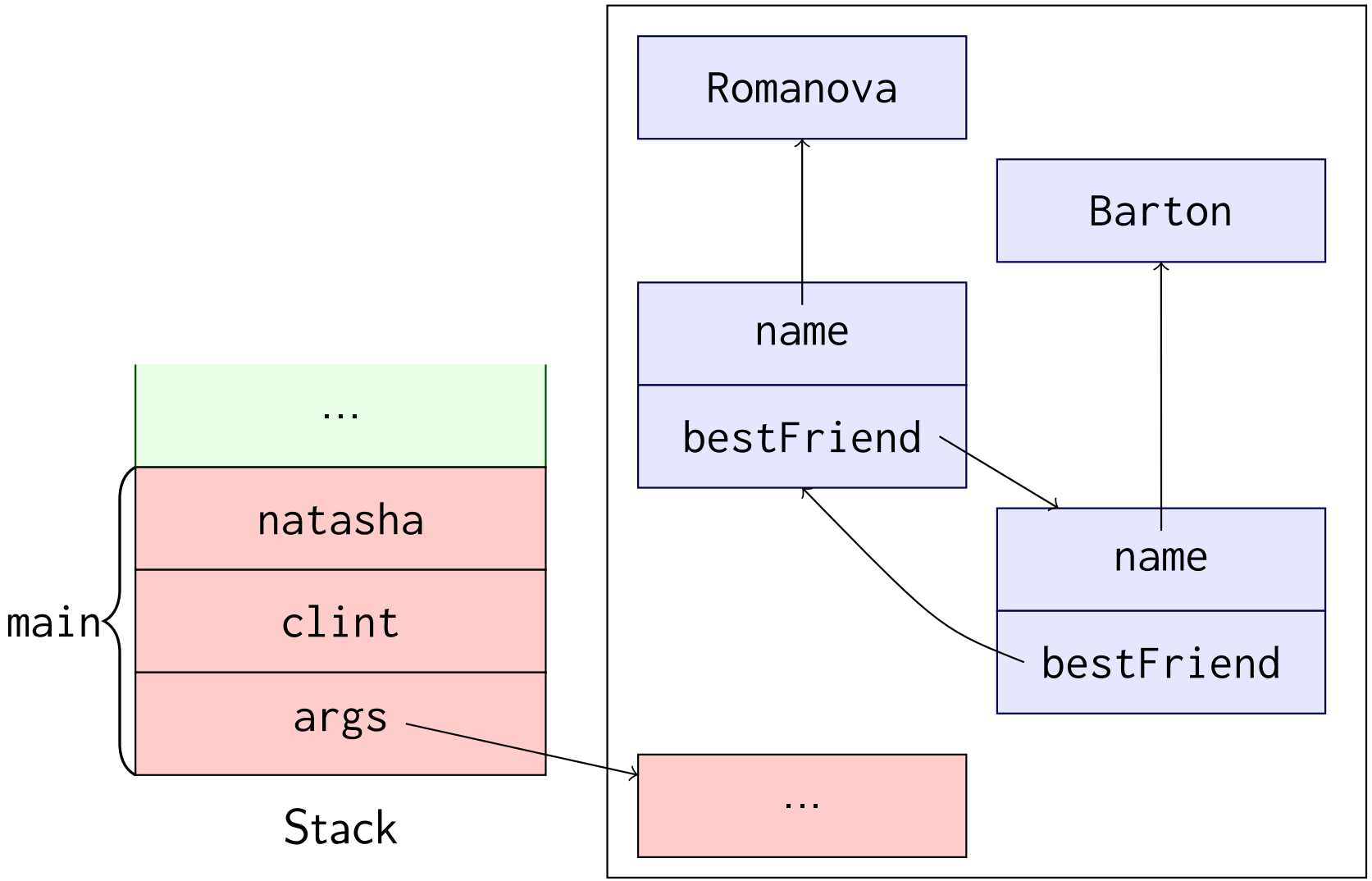
Mark-and-sweep: natasha = clint = null; ismét

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



Mark-and-sweep: mark fázis vége

- A sweep fázis felszabadítja az elérhetetlen objektumokat



Statikus mezők

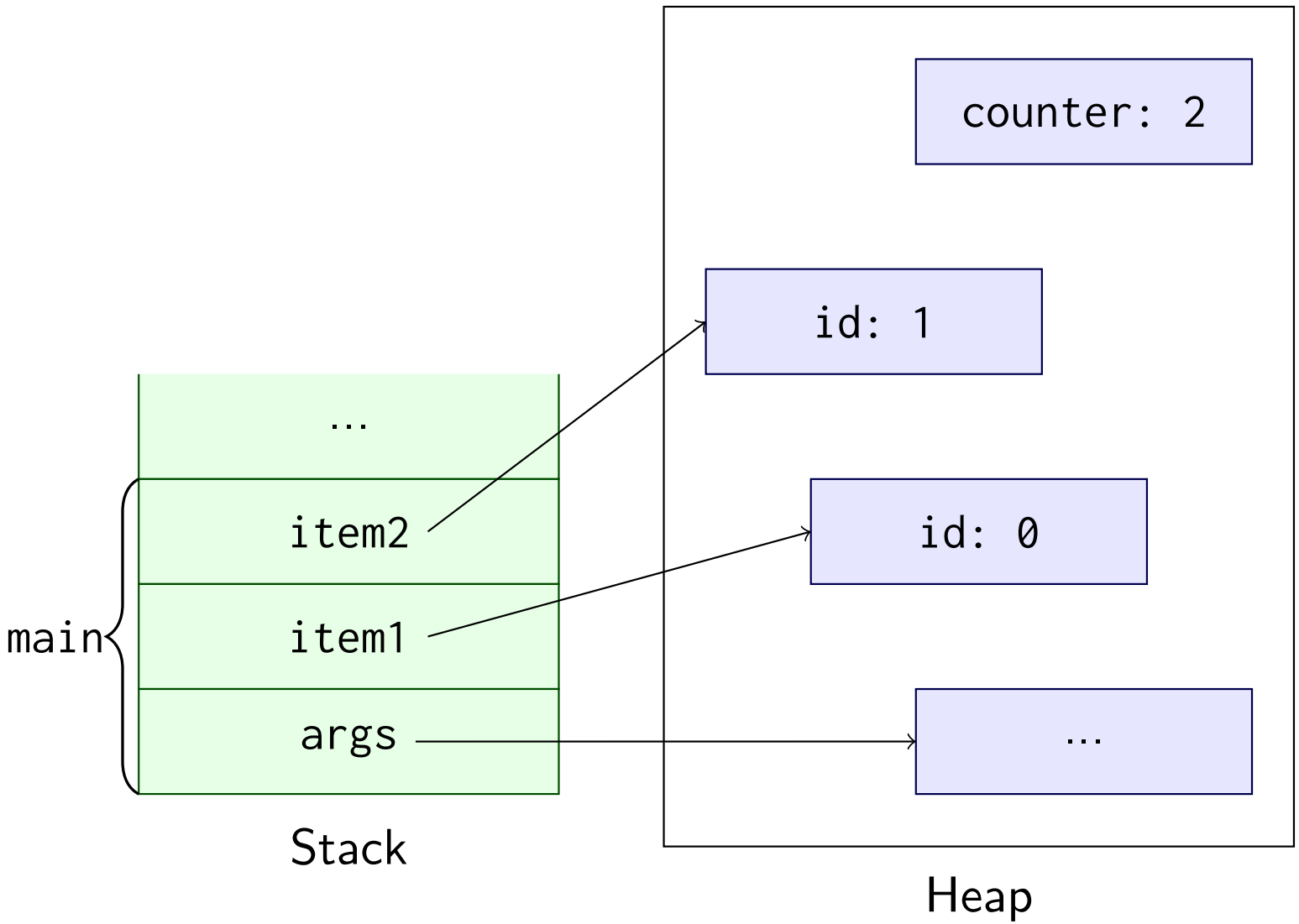
- Hasonló a C globális változóihoz
- Csak egy létezik belőle
- Az osztályon keresztül érhető el
- Mintha *statikus tárhelyen* lenne, nem az objektumokban

```
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 item1 = new Item(), item2 = new Item();
        System.out.println( item1.id );
        System.out.println( item2.id );
        System.out.println( item1.counter );  // csúf, jelentése:
                                              // Item.counter
    }
}
```

`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 );
    }
}
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();
    }
}
```

Outline

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, feltöltése, rendezése

```
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); }
    }
}
```

Statikus tagok importja

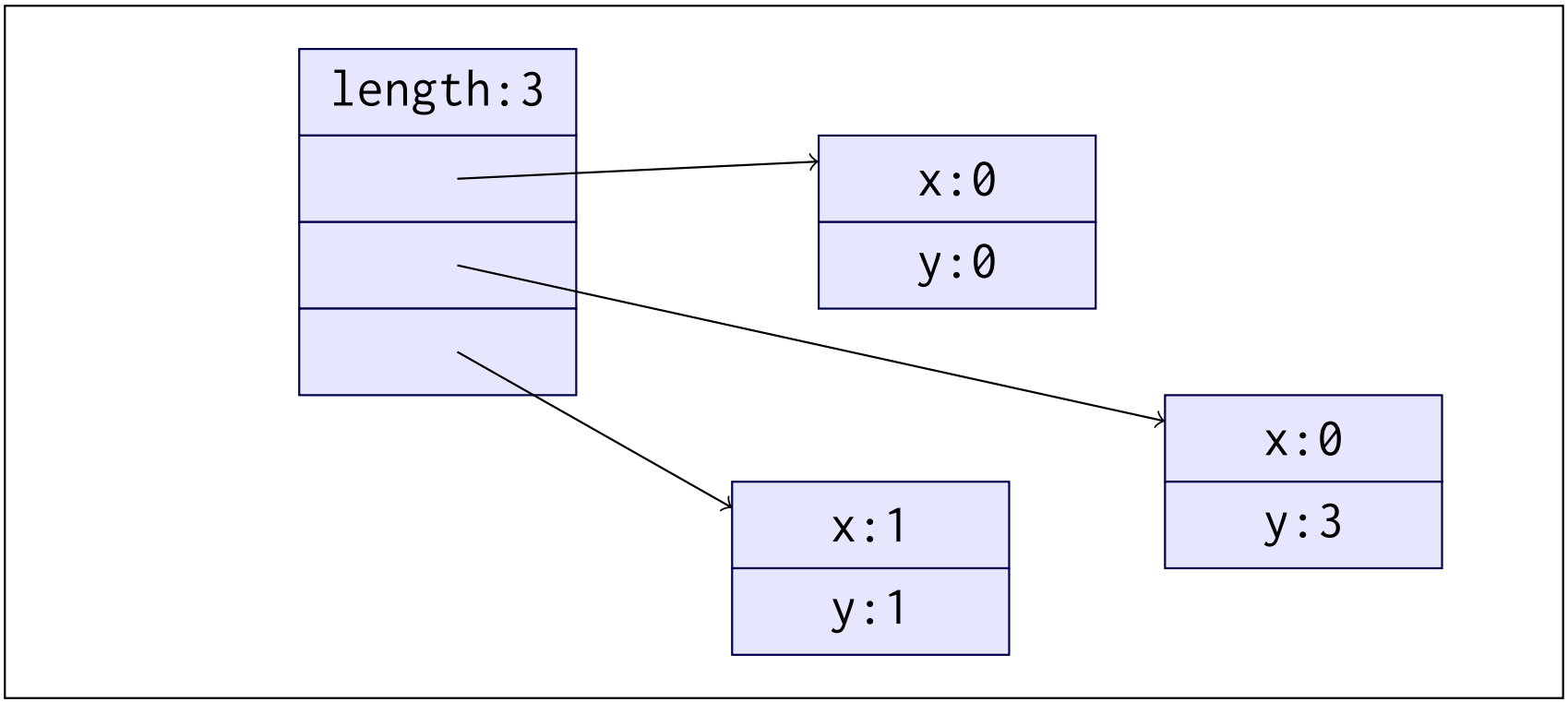
```
import static java.util.Arrays.sort;
class Sort {
    public static void main( String[] args ){

        int[] numbers = new int[args.length]; // 0-kkal feltöltve

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

        sort(numbers);

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

Referenciák tömbje

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

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 );
```

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 );
```

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");
    }
}
```

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;
}
```

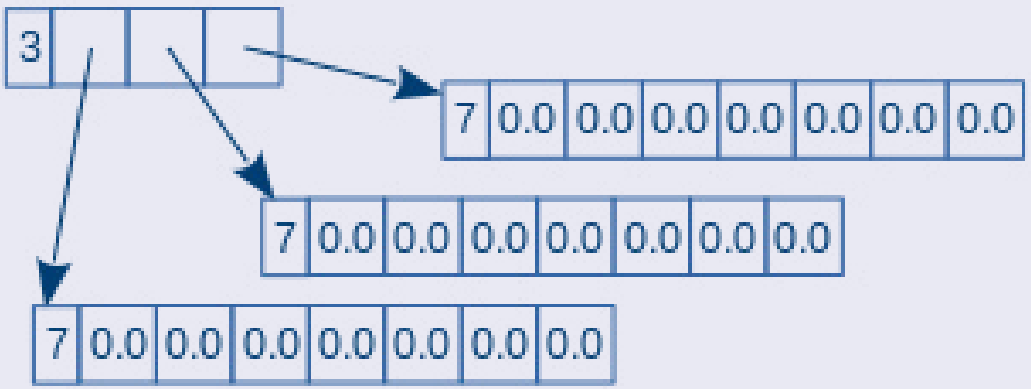
C versus Java

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];
```

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

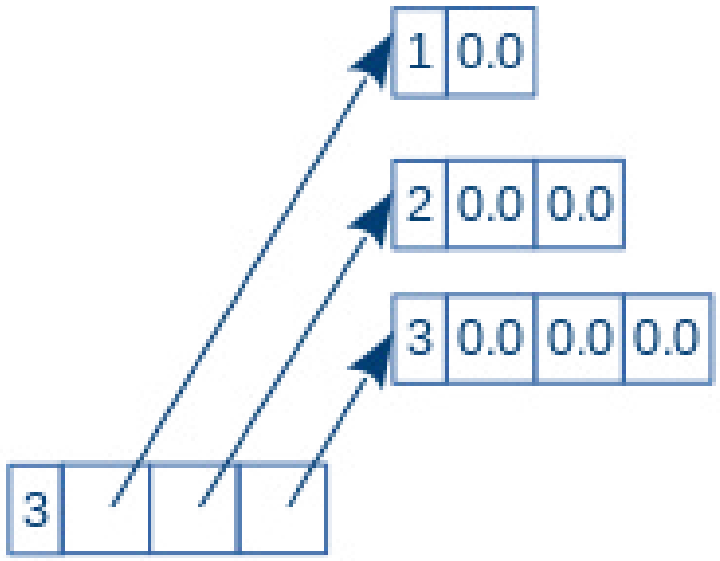
Tömbök tömbjének tömbje Javában

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

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

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

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

Outline

Referencia típusok Javában

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

Felsorolási típus

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

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

Felsorolási típus

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

- Referencia típus
- Értékek: objektumok, nem `intek`

```
Day best = Day.SAT;       // import static?
best = 3;                 // fordítási hiba
int n = best;             // fordítási hiba
int m = best.ordinal();   // 6
```

Fix, zárt típusértékhalmaz

Csak a felsorolt típusértékek!

- **nem hívható meg a konstruktor, pl.:** `new Day()`

Fix, zárt típusértékhalmaz

Csak a felsorolt típusértékek!

- **nem hívható meg a konstruktor, pl.:** `new Day()`

- reflection segítségével sem példányosítható
- nem örökölhetünk belőle
- klónozással sem jön létre új objektum
- objektumszerializációval sem hozható létre új objektum

Konstruktorok, tagok

```
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();
    }
}   // Forrás: Java Community Process (módosítva)
```

switch-utasításban

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

Programozási nyelvek – Java

Kifejezések, utasítások

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Forráskód felépítése

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

Outline

Karakterkódolási szabványok

character encodings

- Bacon's cipher, 1605 (Francis Bacon)
- Baudot-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)

Fordítás: karakterkódolás

- Legyen `Main.java` magyar Windowsos karakterkódolású.
- Legyen a gépemen más, pl. UTF-8 karakterkódolás.

Fordítás: karakterkódolás

- Legyen `Main.java` magyar Windowsos karakterkódolású.
- Legyen a gépemen más, pl. UTF-8 karakterkódolás.

Fordítás alapértelmezett karakterkódolással

```
$ javac Main.java
Main.java:2: error: unmappable character (0xE1) for encoding UTF-8
    String hib?s;
```

Fordítás: karakterkódolás

- Legyen `Main.java` magyar Windowsos karakterkódolású.
- Legyen a gépemen más, pl. UTF-8 karakterkódolás.

Fordítás alapértelmezett karakterkódolással

```
$ javac Main.java
Main.java:2: error: unmappable character (0xE1) for encoding UTF-8
    String hib?s;
```

Átváltás magyar Windowsos karakterkódolásra

```
$ javac -encoding Cp1250 Main.java
```

Outline

Lexikális elemek

- Kulcsszavak
- Azonosítók
- Operátorok
- Literálok
- Zárójelek: `(.) [.] {.} <.>`
- Speciális jelek: `. , : ; -> | ... :: @`
- Megjegyzések (egysoros, többsoros, „dokumentációs'')

Ebben a félévben tanulandó kulcsszavak és foglalt szavak

Utasítások: `if else switch case default while do for break continue return try catch finally throw assert yield`

Programszerkezet: `package import class enum interface extends implements`

Deklarációk: `public protected private abstract static final throws`

Típusok: `boolean char byte short int long float double void`

Speciális változók, konstruktorok: `this super`

Operátorok: `instanceof new`

Literálok: `true false null`

További kulcsszavak és foglalt szavak

Típuskikövetkeztetés

`var`

Deklarációkban

`synchronized volatile transient strictfp native`

Nem használt foglalt szavak

`_ const goto`

Moduldeklarációkban használt kulcsszavak

`module exports open opens provides requires uses with to transitive`

Azonosítók

- betűk, számjegyek, _ és $
- Unicode betűk, akár szökőév vagy ε

Konvenciók

```
package java.lang;
public final class Integer ... {
    ...
    public static final int MAX_VALUE = 2147483647;
    public int intValue(){ ... }
    ...
}
```

Literálok

- Logikai ~: `true` és `false`
- Karakter-~: `'c'`, `'\t'`, `'\''`, `'\\'`, `'\uBABE'`
- Szöveg-~: `"this is a string\ncontaining \u0032 lines"`
- Egész ~
  - `int` típusú: `1984`, `9_772_756`, `0123`, `0XBee`, `0xCAFE_BABE`, `0b1010101`
  - `long` típusú: `1984L`, `1984l`, `0xDEAD_BEEF_ADDED_C00L`
- Lebegőpontos ~
  - `double` típusú: `3.14159`, `.000_001`, `1E-6`, `6.022140857e23`, `3.`, `3D`, `3.14d`, `0x1.Bp-2` = `(1+11./16)/4`, `0X1DE.1P0D`
  - `float` típusú: `3.14159F`, `.000_001f`…

Outline

Kifejezések

- szintaxis: operátorok arítása, fixitása; zárójelezés
- kiértékelés
  - precedencia (`A + B * C`)
  - asszociativitás (`A - B - C`)
  - operandusok/paraméterek kiértékelési sorrendje (`A + B`, `f(A,B)`)
  - lustaság (`A ? B : C`)
  - mellékhatás (++x)

Példa mellékhatásos kifejezésre: *olvasás EOF-ig* idiómája

Legyen `in` egy bemeneti adatfolyam, pl. megnyitott fájl.

```
int v;
while( (v=in.read()) != -1 ){
    ...
}
```

Példa mellékhatásos kifejezésre: *olvasás EOF-ig* idiómája

Legyen `in` egy bemeneti adatfolyam, pl. megnyitott fájl.

```
int v;
while( (v=in.read()) != -1 ){
    ...
}
```

Két mellékhatás a ciklus feltételében!

Logikai műveletek kiértékelése

- Lusta: `A && B`, `A || B`

Logikai műveletek kiértékelése

- Lusta: `A && B`, `A || B`
- Mohó: `A & B`, `A | B`

(`A` és `B` típusa `boolean`)

Lusta és mohó művelettábla

Jelölje $\uparrow$, $\downarrow$, $\bot$ és ∞ a négy lehetséges eredményt egy logikai kifejezés kiértékeléséhez: igaz, hamis, kivétel, nem terminálló számítás. Az $\alpha \wedge \beta$ kifejezés értéke az α és β értékének függvényében (a mellékhatásoktól itt eltekintünk):

α && β	$\beta = \uparrow$	$\beta = \downarrow$	$\beta = \bot$	$\beta = \infty$
$\alpha = \uparrow$	$\uparrow$	$\downarrow$	$\bot$	∞
$\alpha = \downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$\alpha = \bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\alpha = \infty$	∞	∞	∞	∞

α & β	$\beta = \uparrow$	$\beta = \downarrow$	$\beta = \bot$	$\beta = \infty$
$\alpha = \uparrow$	$\uparrow$	$\downarrow$	$\bot$	∞
$\alpha = \downarrow$	$\downarrow$	$\downarrow$	$\bot$	∞
$\alpha = \bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\alpha = \infty$	∞	∞	∞	∞

Példa mohó logikai operátorra

```
int v1, v2;
while( ((v1 = in1.read()) != -1) | ((v2 = in2.read()) != -1) ){
    if( v1 == -1 ){
        out.write( v2 );
    } else if( v2 == -1 ){
        out.write( v1 );
    } else {
        out.write( v1+v2 );
    }
}
```

Bitműveletek

- Bitenkénti *és*elés és *vagy*olás: `A & B`, `A | B`
  (`A` és `B` típusa `int` vagy `long`)

Bitműveletek

- Bitenkénti *é*selés és *vagy*olás: `A & B`, `A | B` (`A` és `B` típusa `int` vagy `long`)
- XOR: `A ^ B`

Bitműveletek

- Bitenkénti *é*selés és *vagy*olás: `A & B`, `A | B` (`A` és `B` típusa `int` vagy `long`)
- XOR: `A ^ B`
- Bitenkénti ellentett: `~A`

Bitműveletek

- Bitenkénti *é*selés és *vagy*olás: `A & B`, `A | B` (`A` és `B` típusa `int` vagy `long`)
- XOR: `A ^ B`
- Bitenkénti ellentett: `~A`
- Léptetés: `A << B`, `A >> B`, `A >>> B`

Operátorok túlterhelése, új operátorok definiálása

- Csak a beépített operátorok (nem lehet újat definiálni)
- Csak a beépített jelentéssel (nem lehet túlterhelni)
  - Beépített túlterhelés: pl. + vagy &

Outline

Utasítások

- Kifejezéskiértékelő utasítás
  - Értékadások
  - Metódushívás
- `return`-utasítás és `yield`-utasítás
- Elágazások (`if`, `switch`)
- Ciklusok (`while`, `do`–`while`, `for`)
- Nem strukturált: `break`, `continue`
- Blokk-utasítás
- Deklaráció (pl. változó~)
- Kivételkezelő és -kiváltó utasítások
- `assert`-utasítás

Puzzle 22: Dupe of URL (Bloch, Gafter: Java Puzzlers)

```
class Main {
    public static void main( String[] args ){
        https://jdk.java.net/
        System.out.println();
    }
}
```

Puzzle 22: címkézett utasítás

```
class Main {
    public static void main( String[] args ){
        https://jdk.java.net/
        System.out.println();
    }
}
```

switch-utasítás

- egész típusokra
- felsorolási típusokra
- `String` típusra

switch-utasítás felsorolási típusra

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

switch-utasítás Stringre

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

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

```
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

```
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;
       };
```

yield-utasítás

```
int days = switch(month){
               case 4, 6, 9, 11 -> 30;
               case 2 -> { int leap = 0;
                           if( year % 4 == 0 ) leap = 1;
                           if( year % 100 == 0 ) leap = 0;
                           if( year % 400 == 0 ) leap = 1;
                           yield 28 + leap;
                         }
               default -> 31;
           };
```

Nem triviális túlcsorgás

```
enum States {RED, AMBER, GREEN};
...
switch ( trafficlight ){
    case RED:    stop();
                 break;
    case AMBER:  if( canSafelyStop() ){
                     stop();
                     break;
                 }
    case GREEN:  go();
}
```

Javában nem, de C-ben ilyen is írható

```
switch ( trafficlight ){
    case AMBER:  if( canSafelyStop() ){    // not valid in Java
    case RED:       stop();
                    break;
                }
    case GREEN:  go();
}
```

Programozási nyelvek – Java

Hibák és kivételek

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Hibajelzés kivétel kiváltásával

```
public class Time {
    private int hour, minute;   // 0 <= hour < 24, 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, minute;   // 0 <= hour < 24, 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

Outline

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`

Nem ellenőrzött kivételek

unchecked exception

- Pl. `NullPointerException`, `ArrayIndexOutOfBoundsException`
- Dinamikus szemantikai hiba
- „Bárhol'' keletkezhet

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-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 fn ){
    try {
        BufferedReader in = new BufferedReader(new FileReader(fn));
        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);
    }
}
```

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);
}
```

A *try-with-resources* utasítás

```
public static Time readTimeOrUseDefault( String fn ){
    try {
        try(
            BufferedReader in = new BufferedReader(new FileReader(fn))
        ){
            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);
    }
}
```

A *try-with-resources* utasítással még egyszerűbben

```
public static Time readTimeOrUseDefault( String fn ){
    try(
        BufferedReader in = new BufferedReader(new FileReader(fn))
    ){
        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);
    }
}
```

Több erőforrás használata

```
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

Karakterkódolási szabványok

character encodings

- Bacon's cipher, 1605 (Francis Bacon)
- Baudot-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)

Szövegfájlok írása

```
import java.io.*;
```

Kiírás a platform alapértelmezett kódolásával

```
try( FileWriter out = new FileWriter(fname) ){
     out.write("árvíztűrő ütvefúrógép", 0, 21);
}
```

Szövegfájlok írása

```
import java.io.*;
```

Kiírás a platform alapértelmezett kódolásával

```
try( FileWriter out = new FileWriter(fname) ){
     out.write("árvíztűrő ütvefúrógép", 0, 21);
}
```

Karakterkódolás explicit megadása

```
try( OutputStreamWriter out = new OutputStreamWriter(
                                   new FileOutputStream(fname),
                                   "Cp1250"
                               ) ){
     out.write("árvíztűrő ütvefúrógép", 0, 21);
}
```

Szövegfájlok olvasása soronként

```
import java.io.*;
```

Karakterkódolás explicit megadása

```
try(
     BufferedReader in = new BufferedReader(
                              new InputStreamReader(
                                   new FileInputStream(fname),
                                   "Cp1250"
                         )    )
){
     String line;
     while( (line=in.readLine()) != null ){ ... }
}
```

Programozási nyelvek – Java

Műveletek

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

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; }

    ...
}
```

Tervezett használat

```
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);              // 1/6
        println(q);              // 1/2
    }
    private static void println( Rational r ){
        System.out.println( r.getNumerator() + "/" +
                            r.getDenominator() );
    }
}
```

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;
    }
    ...
}
```

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);
```

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;
    }
}
```

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

Trükkös szabályok: „jobban illeszkedő”

```
static void m( long n ){ ... }
static void m( float n ){ ... }
public static void main( String[] args ){
    m(3);
}
```

Egyformán illeszkedő

```
static void m( long n, float m ){ ... }
static void m( float m, long n ){ ... }
public static void main( String[] args ){
    m(4,2);
}
```

```
Foo.java:5: error: reference to m is ambiguous
        m(4,2);
        ^
  both method m(long,float) in Foo
   and method m(float,long) in Foo match
1 error
```

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 (overloading)

- Több metódus ugyanazzal a névvel, több konstruktor

Túlterhelés (overloading)

- Több metódus ugyanazzal a névvel, több konstruktor
- Formális paraméterek eltérnek
  - Paraméterek száma
  - Paraméterek deklarált típusa

Túlterhelés (overloading)

- 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

Túlterhelés (overloading)

- 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

Jó ez így?

```
public class Rational {
    ...

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

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

Jogos túlterhelés

```
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( Rational that ){
        if( that == null ) throw new IllegalArgumentException();
        this.numerator = that.numerator;
        this.denominator = that.denominator;
    }
    ...
}
```

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);
    }
    ...
}
```

Alapértelmezett érték – a Java ezt nem engedi

```
public class Rational {
    ...
    public Rational( int numerator = 0, int denominator = 1 ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public void set( int numerator = 0, int denominator = 1 ){
        if( denominator <= 0 ) throw new IllegalArgumentException();
        this.numerator = numerator;
        this.denominator = denominator;
    }
    ...
}
```

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);              // legelső utasítás kell legyen
    }
    public Rational(){
        this(0);
    }
    ...
}
```

Konstruktor(ok) helyett gyártóművelet(ek)

factory method, pl. `new Rational(0)` helyett `Rational.zero()`

```
public class Rational {
    ...
    private Rational( int numerator, int denominator ){
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public static Rational make( int numerator, int denominator ){
        return new Rational(numerator,denominator);
    }
    public static Rational valueOf( int val ){return make(val,1);}
    public static Rational oneOver( int den ){return make(1,den);}
    public static Rational zero(){ return make(0,1); }
}
```

Outline

Egy másfajta megközelítés

```
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);              // 1/6
Rational r = p.times(q);
println(r);              // 1/12
println(p);              // 1/6
```

Megvalósítások

```
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;
    }
}
```

Megvalósítások

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

Operátor túlterhelés nincs a Javában

```
package numbers;
public class Rational {
    private int numerator, denominator;
    public Rational( int numerator, int denominator ){ ... }
    ...
    public Rational operator*( Rational that ){
        return new Rational( this.numerator * that.numerator,
                             this.denominator * that.denominator );
    }
    public Rational operator*=( Rational that ){
        this.numerator *= that.numerator;
        this.denominator *= that.denominator;
        return this;
    }
}
```

Sosem módosuló belső állapot

```
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 ){ ... }
    ...
}
```

Outline

Globális konstans

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

- Osztályszintű mező
- Picit olyan, mint a C-ben egy `#define`
- Hasonló a C-beli `const`-hoz is (de nem pont ugyanaz)
- 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)!

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

Módosíthatatlan lokális változó

```
public class Rational {
    ...
    public void simplify(){
        final int gcd = gcd(numerator, denominator);
        numerator /= gcd;
        denominator /= gcd;
    }
    ...
}
```

Módosíthatatlan formális paraméter

Hibás

```
static java.math.BigInteger factorial( final int n ){
    assert n > 0;
    java.math.BigInteger result = java.math.BigInteger.ONE;
    while( n > 1 ){
        result = result.multiply(java.math.BigInteger.valueOf(n));
        --n;
    }
    return result;
}
```

Módosíthatatlan formális paraméter

Helyes

```
static java.math.BigInteger factorial( final int n ){
    assert n > 0;
    java.math.BigInteger result = java.math.BigInteger.ONE;
    for( int i=n; i>1; --i ){
        result = result.multiply(java.math.BigInteger.valueOf(i));
    }
    return result;
}
```

Mutable versus Immutable

Módosítható belső állapot

```
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

```
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 ){ ... }
```

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() );
```

Más elnevezési konvenció + mutable + túlterhelés

```
public class Rational {

    private int numerator, denominator;

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

    ...
}
```

```
p.numerator(3);
System.out.println( p.numerator() );
```

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!

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 ){ ... }
}
```

Kitérő

```
int[] t = new int[3];
t = new int[4];

int[] s = {1,2,3};
s = {1,2,3,4};  // fordítási hiba
s = new int[]{1,2,3,4};
```

final referencia

```
final Rational p = new Rational(1,2);
p.setNumerator(3);
p = new Rational(1,4); // fordítási hiba
```

final referencia

```
final Rational p = new Rational(1,2);
p.setNumerator(3);
p = new Rational(1,4); // fordítási hiba

final int[] data = new int[2];
data[0] = 3;
data[1] = 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;
```

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();
```

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("");
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);
```

Outline

Osztályszintű metódus (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);
```

Osztályszintű metódus (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);
```

Outline

Paraméterátadási technikák

- Szövegszerű helyettesítés
- Érték szerinti
- Érték-eredmény szerinti
- Eredmény szerinti
- Cím szerinti
- Megosztás szerinti
- Név szerinti
- Igény szerinti

Paraméterátadás Javában

Érték szerinti (call-by-value)

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-value)

```
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);
```

Változó számú paraméter

```
static int sum( int[] nums ){
    int s = 0;
    for( int n: nums ){ s += n; }
    return s;
}
                          sum( new int[]{1,2,3,4,5,6} )
```

Változó számú paraméter

```
static int sum( int[] nums ){
    int s = 0;
    for( int n: nums ){ s += n; }
    return s;
}
```

```
sum( new int[]{1,2,3,4,5,6} )
```

```
static int sum( int... nums ){
    int s = 0;
    for( int n: nums ){ s += n; }
    return s;
}
```

```
sum(1,2,3,4,5,6)
```

Outline

Íme egy jól kinéző osztálydefiníció...

```
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;
    }
}
```

És ha a paraméterek nem diszjunktak?

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

```
Rational p = new Rational(1,2);
p.divideBy(p);
```

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;
    }
}
```

```
int[] cheat = {3,4};
Rational p = new Rational(1,2);  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.name = 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

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 )
```

Programozási nyelvek – Java

Parametrikus polimorfizmus

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

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

Outline

Alternatíva: java.util.ArrayList

kényelmes szabványos könyvtár, hasonló belső működés

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


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

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

Alternatíva: java.util.ArrayList

kényelmes szabványos könyvtár, hasonló belső működés

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


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

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

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

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

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> namez = 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
  - Haskell: függvény
  - Java: típus (osztály), metódus
- Típussal paraméterezhető kód
  - Haskell: bármilyen típussal
  - Java: referenciatípusokkal

Típusparaméter

Primitív típussal helytelen

```
ArrayList<int> numbers
```

Típusparaméter

Primitív típussal helytelen

```
ArrayList<int> numbers
```

Referenciatípussal helyes

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

Típusparaméter

Primitív típussal helytelen

```
ArrayList<int> numbers
```

Referenciatípussal helyes

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

Furcsamód ez is helyes

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(42);                    // auto-boxing:    int -> Integer
int fourtytwo = numbers.get(1);     // auto-unboxing:  Integer -> int
```

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           );
```

Adatszerkezetek a java.util csomagban

Sorozat

```
ArrayList<String> colors = new ArrayList<>();
colors.add("red"); colors.add("white"); colors.add("red");
String third = colors.get(2);
```

Adatszerkezetek a java.util csomagban

Sorozat

```
ArrayList<String> colors = new ArrayList<>();
colors.add("red"); colors.add("white"); colors.add("red");
String third = colors.get(2);
```

Halmaz

```
HashSet<String> colors = new HashSet<>();
colors.add("red"); colors.add("white"); colors.add("red");
int two = colors.size();
```

Adatszerkezetek a java.util csomagban

Sorozat

```
ArrayList<String> colors = new ArrayList<>();
colors.add("red"); colors.add("white"); colors.add("red");
String third = colors.get(2);
```

Halmaz

```
HashSet<String> colors = new HashSet<>();
colors.add("red"); colors.add("white"); colors.add("red");
int two = colors.size();
```

Leképezés

```
HashMap<String,String> colors = new HashMap<>();
colors.put("red","piros"); colors.put("white","fehér");
String whiteHu = colors.get("white");
```

Outline

Generikus osztály

```
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 ){ ... }
    ...
}
```

Implementálás

```
public class ArrayList<T> {
    private T[] data;
    private int size = 0;
    ...
    public T get( int index ){
        if( index < size ) return data[index];
        else throw new IndexOutOfBoundsException();
    }
    ...
}
```

Implementálás

```
public class ArrayList<T> {
    private T[] data;
    private int size = 0;
    ...
    public void add( T item ){
        if( size == data.length ){
            data = java.util.Arrays.copyOf(data,data.length+1);
        }
        data[size] = item;
        ++size;
    }
    ...
}
```

Allokálás: fordítási hiba

```
public class ArrayList<T> {
    private T[] data;
    private int size = 0;
    ...
    public ArrayList(){ this(256); }
    public ArrayList( int initialCapacity ){
        data = new T[initialCapacity];
    }
    ...
}
```

```
ArrayList.java:6: error: generic array creation
        data = new T[initialCapacity];
```

Típustörlés

type erasure

- Típusparaméter: statikus típusellenőrzéshez
- Tárgykód: típusfüggetlen (mint a Haskellben)
- Más, mint a C++ *template*
- Kompatibilitási okok
- Futás közben nem használható a típusparaméter

Így *képzelhetjük el* a tárgykódot

```
public class ArrayList {
    private Object[] data;
    ...
    public ArrayList(){ ... }
    public Object get( int index ){ ... }
    public void set( int index, Object item ){ ... }
    public void add( Object item ){ ... }
    public Object remove( int index ){ ... }
    ...
}
```

Kompatibilitás: nyers típus

raw type

```
import java.util.ArrayList;
...
ArrayList<String> paraméteres = new ArrayList<>();
paraméteres.add("Romeo");
paraméteres.add(12);        // fordítási hiba
String s = paraméteres.get(0);
```

Kompatibilitás: nyers típus

raw type

```
import java.util.ArrayList;
...
ArrayList<String> paraméteres = new ArrayList<>();
paraméteres.add("Romeo");
paraméteres.add(12);        // fordítási hiba
String s = paraméteres.get(0);

ArrayList nyers = new ArrayList();
nyers.add("Romeo");
nyers.add(12);
Object o = nyers.get(0);
```

Allokálás: még mindig rosszul

```
public class ArrayList<T> {
    private T[] data;
    private int size = 0;
    ...
    public ArrayList(){ this(256); }
    public ArrayList( int initialCapacity ){
        data = new Object[initialCapacity];
    }
    ...
}
```

```
ArrayList.java:6: error: incompatible types: Object[] cannot be
converted to T[]
        data = new Object[initialCapacity];
                          ^
  where T is a type-variable:
```

Allokálás – így már működik

```
public class ArrayList<T> {
    private T[] data;
    private int size = 0;
    ...
    public ArrayList(){ this(256); }
    public ArrayList( int initialCapacity ){
        data = (T[])new Object[initialCapacity];
    }
    ...
}
```

javac ArrayList.java

```
Note: ArrayList.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Allokálás – így már működik, de azért nem az igazi...

```
public class ArrayList<T> {
    private T[] data;
    private int size = 0;
    ...
    public ArrayList(){ this(256); }
    public ArrayList( int initialCapacity ){
        data = (T[])new Object[initialCapacity];
    }
    ...
}
```

```
javac -Xlint:unchecked ArrayList.java
```

```
ArrayList.java:6: warning: [unchecked] unchecked cast
  required: T[]         found: Object[]
```

Kényszerítsünk máshol?

```
public class ArrayList<T> {
    private Object[] data;
    private int size = 0;
    ...
    public T get( int index ){
        if( index < size ) return (T)data[index];
        else throw new IndexOutOfBoundsException();
    }
    ...
}
```

```
javac -Xlint:unchecked ArrayList.java
```

```
ArrayList.java:10: warning: [unchecked] unchecked cast
  required: T        found: Object
```

Warning-mentesen

```
public class ArrayList<T> {
    private Object[] data;
    private int size = 0;
    ...
    @SuppressWarnings("unchecked")
    public T get( int index ){
        if( index < size ) return (T)data[index];
        else throw new IndexOutOfBoundsException();
    }
    ...
}
```

java Searching 42

```
import java.util.ArrayList;
class Searching {
    public static void main( String[] args ){
        ArrayList<String> seq = new ArrayList<>();
        seq.add("42");
        System.out.println( seq.contains("42") );
        System.out.println( seq.contains(args[0]) );
    }
}
```

```
true
true
```

Keresés az adatszerkezetben

```
public class ArrayList<T> {
   private Object[] data;
   private int size = 0;
   ...
   public boolean contains( T item ){
      for( int i = 0; i < size; ++i ){
         if( data[i] == item ) return true;
      }
      return false;
   }
}
```

java Searching 42

```
ArrayList<String> seq = new ArrayList<>();
seq.add("42");                          // true false
System.out.print( seq.contains("42") +" "+ seq.contains(args[0]) );
```

Tartalmi összehasonlítás

```
public class ArrayList<T> {
   private Object[] data;
   private int size = 0;
   ...
   public boolean contains( T item ){
      for( int i = 0; i < size; ++i ){
         if( java.util.Objects.equals(data[i],item) ) return true;
      }
      return false;
   }
}
```

java Searching 42

```
ArrayList<String> seq = new ArrayList<>();
seq.add("42");                    // true true
System.out.print( seq.contains("42") +" "+ seq.contains(args[0]) );
```

Programozási nyelvek – Java

Öröklődés

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Öröklődés (inheritance)

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

- Egy típust egy másik típusból származtatunk
  - Csak a különbségeket kell megadni: A Δ B
  - Újrafelhasználás

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
  - child class
  - parent class

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
  - child class
  - parent class
- Tranzitivitás: leszármazott osztály – ősosztály
  - alosztály: subclass, derived class
  - bázisosztály: super class, base class

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
  - child class
  - parent class
- Tranzitivitás: leszármazott osztály – ősosztály
  - alosztály: subclass, derived class
  - bázisosztály: super class, base class
- Körkörösség kizárva!

Példa öröklődésre

```
public class Time {
    private int hour, minute;    // initialized to 00:00
    public int getHour(){ ... }
    public int getMinute(){ ... }
    public void setHour( int hour ){ ... }
    public void setMinute( int minute ){ ... }
    public void aMinutePassed(){ ... }
}
```

Példa öröklődésre

```
public class Time {
    private int hour, minute;    // initialized to 00:00
    public int getHour(){ ... }
    public int getMinute(){ ... }
    public void setHour( int hour ){ ... }
    public void setMinute( int minute ){ ... }
    public void aMinutePassed(){ ... }
}
```

```
public class ExactTime extends Time {
    private int second;          // initialized to 00
    public int getSecond(){ ... }
    public void setSecond( int second ){ ... }
    public boolean earlierThan( ExactTime that ){ ... }
}
```

Implicit szülőosztály

```
public class Time extends java.lang.Object {
    private int hour, minute;    // initialized to 00:00
    public int getHour(){ ... }
    public int getMinute(){ ... }
    public void setHour( int hour ){ ... }
    public void setMinute( int minute ){ ... }
    public void aMinutePassed(){ ... }
}
```

```
public class ExactTime extends Time {
    private int second;          // initialized to 00
    public int getSecond(){ ... }
    public void setSecond( int second ){ ... }
    public boolean earlierThan( ExactTime that ){ ... }
}
```

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 ){ ... }
    ...
}
```

Outline

A konstruktorok függetlenek az öröklődéstől

```
public class Time {
    private int hour, minute;
    public Time( int hour, int minute ){
        if( hour < 0 || hour > 23 || minute < 0 || minute > 59 )
            throw new IllegalArgumentException();
        this.hour = hour;
        this.minute = minute;
    }
    ...
}
```

A konstruktorok függetlenek az öröklődéstől

```
public class Time {
    private int hour, minute;
    public Time( int hour, int minute ){
        if( hour < 0 || hour > 23 || minute < 0 || minute > 59 )
            throw new IllegalArgumentException();
        this.hour = hour;
        this.minute = minute;
    }
    ...
}
```

```
public class ExactTime extends Time {
    private int second;
```

A konstruktorok függetlenek az öröklődéstől

```
public class Time {
    private int hour, minute;
    public Time( int hour, int minute ){
        if( hour < 0 || hour > 23 || minute < 0 || minute > 59 )
            throw new IllegalArgumentException();
        this.hour = hour;
        this.minute = minute;
    }
    ...
}
```

```
public class ExactTime extends Time {
    private int second;

    public ExactTime( int hour, int minute, int second ){ ? }
}
```

A gyermekosztályba is kell konstruktort írni!

```
public class Time {
    private int hour, minute;
    public Time( int hour, int minute ){ ... }
    ...
}
```

```
public class ExactTime extends Time {
    private int second;
    public ExactTime( int hour, int minute, int second ){
```

A gyermekosztályba is kell konstruktort írni!

```
public class Time {
    private int hour, minute;
    public Time( int hour, int minute ){ ... }
    ...
}
```

```
public class ExactTime extends Time {
    private int second;
    public ExactTime( int hour, int minute, int second ){

        super(hour,minute); // meghívandó a szülő konstruktora
        if( second < 0 || second > 59 )
            throw new IllegalArgumentException();
        this.second = second;
    }
}
```

super(...)-konstruktorhívás

- Szülőosztály valamelyik konstruktora
- Megörökölt tagok inicializálása
- Legelső utasítás kell legyen

super(...)-konstruktorhívás

- Szülőosztály valamelyik konstruktora
- Megörökölt tagok inicializálása
- Legelső utasítás kell legyen

Hibás!!!

```
public class ExactTime extends Time {
    private int second;
    public ExactTime( int hour, int minute, int second ){
        if( second < 0 || second > 59 )
            throw new IllegalArgumentException();
        super(hour,minute);
        this.second = second;
    }
}
```

Miért helyes? Hiányzik a super?!

```
public class Time extends Object {
    private int hour, minute;
    public Time( int hour, int minute ){
        if( hour < 0 || hour > 23 || minute < 0 || minute > 59 )
            throw new IllegalArgumentException();
        this.hour = hour;
        this.minute = minute;
    }
    ...
}
```

Implicit super()-hívás

```
public class Time extends Object {
    private int hour, minute;
    public Time( int hour, int minute ){
        super();
        if( hour < 0 || hour > 23 || minute < 0 || minute > 59 )
            throw new IllegalArgumentException();
        this.hour = hour;
        this.minute = minute;
    }
    ...
}
```

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

Implicit szülőosztály, implicit konstruktor, implicit super

```
class A {}
```

Implicit szülőosztály, implicit konstruktor, implicit super

```
class A {}
```

```
class A extends java.lang.Object {
    A(){
        super();
    }
}
```

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 {}
```

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(); }
}
```

Outline

Öröklődéssel definiált osztály

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

Példánymetódus felüldefiniálása

újradefiniálás, redefinition, overriding

```
package java.lang;
public class Object {
    ...
    public String toString(){ ... }   // java.lang.Object@4f324b5c
}
```

Példánymetódus felüldefiniálása

újradefiniálás, redefinition, overriding

```
package java.lang;
public class Object {
    ...
    public String toString(){ ... }   // java.lang.Object@4f324b5c
}
```

```
public class Time {
    ...
    public String toString(){
        return hour + ":" + minute;   // 8:5
    }
}
```

Picit ügyesebben

újradefiniálás, redefinition, overriding

```
package java.lang;
public class Object {
    ...
    public String toString(){ ... }   // java.lang.Object@4f324b5c
}
```

```
public class Time {
    ...
    public String toString(){          // 8:05
        return String.format("%1$d:%2$02d", hour, minute);
    }
}
```

Opcionális @Override annotációval

újradefiniálás, redefinition, overriding

```
package java.lang;
public class Object {
    ...
    public String toString(){ ... }   // java.lang.Object@4f324b5c
}
```

```
public class Time {
    ...
    @Override public String toString(){          // 8:05
        return String.format("%1$d:%2$02d", hour, minute);
    }
}
```

super.toString() hívása

```
package java.lang;                          // java.lang.Object@4f324b5c
public class Object { ... public String toString(){ ... } ...  }
```

```
public class Time {
    ...
    @Override public String toString(){    // 8:05
        return String.format("%1$d:%2$02d", hour, minute);
    }
}
```

```
public class ExactTime extends Time {
    ...
    @Override public String toString(){    // 8:05:17
        return super.toString() + String.format(":%1$02d", second);
    }
}
```

Túlterhelés és felüldefiniálás

```
package java.lang;
public final class Integer extends Number {
    ...
    public static    int parseInt( String str )          { ... }
    ...
    public @Override 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

Statikus és dinamikus kiválasztódás

System.out

```
public void println(    int value ){ ... }
public void println( Object value ){ ... }   // value.toString()
...
```

```
System.out.println( 7 );                     // 7
System.out.println( "Samurai" );             // Samurai
System.out.println( new Time(21,30) );       // 21:30
```

Programozási nyelvek – Java

Altípusos polimorfizmus

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Az öröklődés két aspektusa

- Kódöröklés
- Altípusképzés

Outline

Öröklődés: altípusképzés

$$A \ \Delta \ B \ \Rightarrow \ A <: B$$

Öröklődés: altípusképzés

$$A \ \Delta \ B \ \Rightarrow \ A <: B$$

```
public class ExactTime extends Time { ... }
```

- Az `ExactTime` mindent tud, amit a `Time`
- Amit lehet `Time`-mal, lehet `ExactTime`-mal is
- `ExactTime` $<:$ `Time`

Öröklődés: altípusképzés

$$A \ \Delta \ B \ \Rightarrow \ A <: B$$

```
public class ExactTime extends Time { ... }
```

- Az `ExactTime` mindent tud, amit a `Time`
- Amit lehet `Time`-mal, lehet `ExactTime`-mal is
- `ExactTime` $<:$ `Time`

- $\forall T$ osztályra : $T <:$ `java.lang.Object`

Altípus

```
public class Time {
    ...
    public void aMinutePassed(){ ... }
    public boolean sameHourAs( Time that ){ ... }
}
```

```
public class ExactTime extends Time {
    ...
    public boolean isEarlierThan( ExactTime that ){ ... }
}
```

```
ExactTime time = new ExactTime();          // 0:00:00
time.aMinutePassed();                      // 0:01:00
time.sameHourAs( new ExactTime() )         // true
```

Liskov-féle helyettesítési elv

LSP: Liskov's Substitution Principle

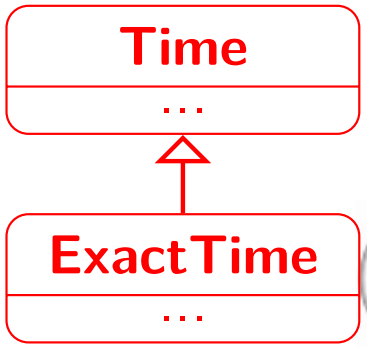
Egy A típus altípusa a B (bázis-)típusnak, ha az A egyedeit használhatjuk a B egyedei helyett, anélkül, hogy ebből baj lenne.

Polimorf referenciák

```
public class Time {
    ...
    public void aMinutePassed(){ ... }
    public boolean sameHourAs( Time that ){ ... }
}
```

```
public class ExactTime extends Time {
    ...
    public boolean isEarlierThan( ExactTime that ){ ... }
}
```

```
ExactTime time1 = new ExactTime();
Time      time2 = new ExactTime();  // upcast
time2.sameHourAs( time1 )
```



Statikus és dinamikus típus

Statikus típus: változó vagy paraméter *deklarált* típusa

- A programszövegből következik
- Állandó
- A fordítóprogram ez alapján típusellenőriz

```
Time time
```

Statikus és dinamikus típus

Statikus típus: változó vagy paraméter *deklarált* típusa

- A programszövegből következik
- Állandó
- A fordítóprogram ez alapján típusellenőriz

```
Time time
```

Dinamikus típus: változó vagy paraméter *tényleges* típusa

- Futási időben derül ki
- Változékony
- A statikus típus altípusa

```
time = ... ? new ExactTime() : new Time()
```

Outline

Felüldefiniálás

```
package java.lang;                          // java.lang.Object@4f324b5c
public class Object { ... public String toString(){ ... } ...  }
```

```
public class Time {
    ...
    @Override public String toString(){    // 8:05
        return String.format("%1$d:%2$02d", hour, minute);
    }
}
```

```
public class ExactTime extends Time {
    ...
    @Override public String toString(){    // 8:05:17
        return super.toString() + String.format(":%1$02d", second);
    }
}
```

Túlterhelés versus felüldefiniálás

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)

```
ExactTime e = new ExactTime();
Time      t = e;
Object    o = t;

System.out.println( e.toString() );     // 0:00:00
System.out.println( t.toString() );     // 0:00:00
System.out.println( o.toString() );     // 0:00:00
```

Példánymetódus hívásánál a 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 Time();
o.setHour(8);              // fordítási hiba
```

Dinamikus típus

Melyik implementációját egy felüldefiniált műveletnek?

```
Object o = new Time();
System.out.println(o);  // toString() impl. kiválasztása
```

- Dinamikus típusellenőrzés

Példa öröklődésre

```
package company.hr;
public class Employee {
    String name;
    int basicSalary;
    java.time.ZonedDateTime startDate;
    ...
}
```

Példa öröklődésre

```
package company.hr;
public class Employee {
    String name;
    int basicSalary;
    java.time.ZonedDateTime startDate;
    ...
}
```

```
package company.hr;
import java.util.*;
public class Manager extends Employee {
    final HashSet<Employee> underlings = new HashSet<>();
    ...
}
```

Szülőosztály

```
package company.hr;
import java.time.ZonedDateTime;
import static java.time.temporal.ChronoUnit.YEARS;
public class Employee {
   ...
   private ZonedDateTime startDate;
   public int yearsInService(){
      return (int) startDate.until(ZonedDateTime.now(), YEARS);
   }
   private static int bonusPerYearInService = 0;
   public int bonus(){
      return yearsInService() * bonusPerYearInService;
   }
}
```

Gyermekosztály

```
package company.hr;
import java.util.*;
public class Manager extends Employee {
   // megörökölt: startDate, yearsInService() ...
   ...
   private final HashSet<Employee> underlings = new HashSet<>();
   public void addUnderling( Employee underling ){
      underlings.add(underling);
   }
   private static int bonusPerUnderling = 0;
   @Override public int bonus(){
      return underlings.size() * bonusPerUnderling + super.bonus();
   }
}
```

Dinamikus kötés megörökölt metódusban is!

```
public class Employee {
   ...
   private int basicSalary;
   public int bonus(){
      return yearsInService() * bonusPerYearInService;
   }
   public int salary(){ return basicSalary + bonus(); }
}
```

```
public class Manager extends Employee {
   ...
   @Override public int bonus(){
      return underlings.size() * bonusPerUnderling + super.bonus();
   }
}
```

Dinamikus kötés megörökölt metódusban is!

```
Employee jack = new Employee("Jack", 10000);
Employee pete = new Employee("Pete", 12000);
Manager eve = new Manager("Eve", 12000);
Manager joe = new Manager("Joe", 12000);
eve.addUnderling(jack);
joe.addUnderling(eve);              // polimorf formális paraméter
joe.addUnderling(pete);

Employee[] company = {joe, eve, jack, pete};    // <-- heterogén
                                                // adatszerkezet
int totalSalaryCosts = 0;
for( Employee e: company ){
   totalSalaryCosts += e.salary();
}
```

Dinamikus kötés

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

Mező és osztályszintű metódus nem definiálható felül

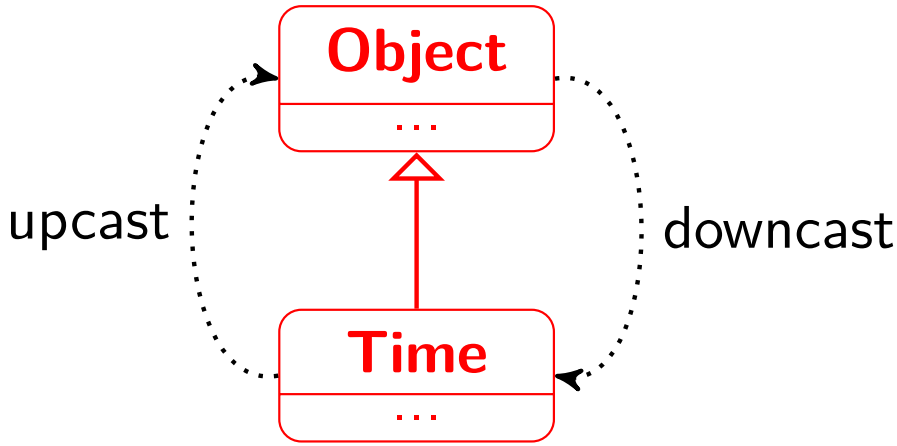
```
class Base {
    int field = 3;
    int iMethod(){ return field; }
    static int sMethod(){ return 3; }
}

class Sub extends Base {
    int field = 33;                          // elfedés
    static int sMethod(){ return 33; }       // elfedés
}
```

```
Sub sub = new Sub();          Base base = sub;

sub.sMethod()  ==  33         base.sMethod()  ==  3
sub.field      ==  33         base.field      ==  3
sub.iMethod()  ==  3          base.iMethod()  ==  3
```

Outline

Konverziók referenciatípusokon

- Automatikus (upcast) – altípusosság
- Explicit (downcast) – type-cast operátor



Típuskényszerítés (downcast)

- A „`(Time)o`'' kifejezés statikus típusa `Time`

Típuskényszerítés (downcast)

- A „`(Time)o`'' kifejezés statikus típusa `Time`
- Ha `o` dinamikus típusa `Time`:

```
Object o = new Time(3,20);
o.aMinutePassed();            // fordítási hiba
((Time)o).aMinutePassed();    // lefordul, működik
```

Típuskényszerítés (downcast)

- A „`(Time)o`'' kifejezés statikus típusa `Time`
- Ha `o` dinamikus típusa `Time`:

```
Object o = new Time(3,20);
o.aMinutePassed();            // fordítási hiba
((Time)o).aMinutePassed();    // lefordul, működik
```

- Ha nem, `ClassCastException` lép fel

```
Object o = "Három óra húsz";
o.aMinutePassed();            // fordítási hiba
((Time)o).aMinutePassed();    // futási hiba
```

Dinamikus típusellenőrzés

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

instanceof-operátor

```
Object o = new ExactTime(3,20,0);
...
if( o instanceof Time ){
    ((Time)o).aMinutePassed();
}
```

- Kifejezés dinamikus típusa altípusa-e a megadottnak

instanceof-operátor

```
Object o = new ExactTime(3,20,0);
...
if( o instanceof Time ){
    ((Time)o).aMinutePassed();
}
```

- Kifejezés dinamikus típusa altípusa-e a megadottnak
- Statikus típusa ne zárja ki a megadottat

```
"apple" instanceof Integer
```

instanceof-operátor

```
Object o = new ExactTime(3,20,0);
...
if( o instanceof Time ){
    ((Time)o).aMinutePassed();
}
```

- Kifejezés dinamikus típusa altípusa-e a megadottnak
- Statikus típusa ne zárja ki a megadottat

```
"apple" instanceof Integer
```

- null-ra false

Dinamikus típus ábrázolása futás közben

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

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

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);
}
```

Öröklődés – altípusosság

- `class A extends B` …
- $A <: B$
- $\forall T: \quad T <:$ `java.lang.Object`

Automatikus „konverzió'' bázistípusra (upcast)

```
String str = "Java";
Object o = str;  // OK
str = o;         // fordítási hiba
```

Kényszerítés altípusra (downcast)

```
String str = "Java";
Object o = str;  // OK
str = (String)o; // OK, dinamikus típusellenőrzés
```

ClassCastException

```
String str = "Java";
Object o = str;
Integer i = (Integer)o;
```

Típusba tartozás (altípusosság)

```
String str = "Java";
Object o = str;
Integer i = (o instanceof Integer) ? (Integer)o : null;
```

Dinamikus típusra típusegyezés

```
String str = "Java";
Object o = str;
Integer i = o.getClass().equals(Integer.class) ? (Integer)o : null;
```

Programozási nyelvek – Java

Interface

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

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 );
    ...
}
```

abstract műveletek: csak deklaráljuk őket

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

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 ){ ... }
}
```

Simplified.java

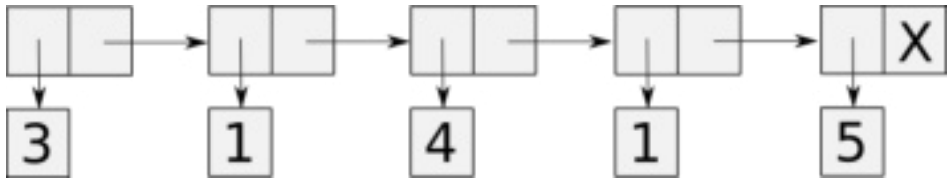
```
public class Simplified implements Rational {
    ...
    public int getNumerator(){ ... }
    public int getDenominator(){ ... }
    Rational times( Rational that ){ ... }
}
```

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> { ... }
class LinkedList<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
```

Iterable és Iterator

- Iterálható (pl. egy adatszerkezet): ha kérhetünk tőle iterátort
- Iterátor: az adatszerkezet elemeinek egymás utáni lekérdezéséhez

java.lang.Iterable

```
public interface Iterable<T> {
    java.util.Iterator<T> iterator();
    ...
}
```

java.util.Iterator

```
public interface Iterator<T> {
    boolean hasNext();
    T next();
    ...
}
```

Iterator elképzelt megvalósítása

```
package java.util;
public class ArrayList<T> implements Iterable<T> {
    Object[] data;
    int size = 0;
    ...
    public Iterator<T> iterator(){ return new ALIterator<>(this); }
}
class ALIterator<T> implements Iterator<T> {
    private final ArrayList<T> theArrayList;
    private int index = 0;
    ALIterator( ArrayList<T> al ){ theArrayList = al; }
    public boolean hasNext(){ return index < theArrayList.size; }
    @SuppressWarnings("unchecked") public T next(){
        return (T)theArrayList.data[index++];
    }
}
```

Iterable és Iterator – polimorfizmus

```
long sum( Iterable<Integer> is ){
    long sum = 0L;
    Iterator<Integer> it = is.iterator();
    while( it.hasNext() ){
        sum += it.next();
    }
    return sum;
}
```

```
List<Integer> list = new LinkedList<>();
...
long sum = sum(list);
```

Iteráló ciklus

```
long sum( Iterable<Integer> is ){
    long sum = 0L;

    for( Integer item: is ){
        sum += item;
    }
    return sum;
}
```

```
List<Integer> list = new LinkedList<>();
...
long sum = sum(list);
```

Többszörös iterálás

```
List<Pair<Integer,Integer>> pairs( List<Integer> ns ){
    List<Pair<Integer,Integer>> ps = new LinkedList<>();
    Iterator<Integer> it = ns.iterator();
    while( it.hasNext() ){
        Integer item = it.next();
        Iterator<Integer> it2 = ns.iterator();
        while( it2.hasNext() ){
            ps.add(new Pair<Integer,Integer>(item,it2.next()));
        }
    }
    return ps;
}
```

Interface lambdák típusozásához

```
interface IntIntToInt {
    int apply( int left, int right );
}
```

```
static int[] zipWith( IntIntToInt fun, int[] left, int[] right ){
    int[] result = new int[Integer.min(left.length, right.length)];
    for( int i=0; i<result.length; ++i ){
        result[i] = fun.apply( left[i], right[i] );
    }
    return result;
}
```

```
zipWith(  (n, m) -> n*m,  new int[]{1,2,3},  new int[]{6,5,4}  )
```

Outline

Functional Interface és `@FunctionalInterface`

- Lambdák típusa lehet
- Csak egy implementálandó metódus
- Speciális *annotáció* használható

```
@FunctionalInterface interface IntIntToInt {
    int apply( int left, int right );
}
```

További példák

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

java.util.Arrays.setAll

```
public static void setAll( int[] array, IntUnaryOperator op )
```

package java.util.function;

```
@FunctionalInterface public interface IntUnaryOperator {
    int applyAsInt( int operand );
    ...
}
```

Streamek

java.util.Arrays.stream

```
public static IntStream stream( int[] array )
```

package java.util.stream;

```
public interface IntStream {
    IntStream filter( IntPredicate predicate );
    IntStream map( IntUnaryOperator mapper );
    int reduce( int identity, IntBinaryOperator op );
    void forEach( IntConsumer action );
    int[] toArray();
    ...
}
```

Mi lehet egy `interface`-ben?

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

Globális konstansok interface-ekben

```
package java.awt;
public interface Transparency {

    public static final   int OPAQUE = 1;
  /*public static final*/ int BITMASK = 2;
  /*public static final*/ int TRANSLUCENT = 3;

    int getTransparency();

}
```

Globális konstans versus enum

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

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

default és static metódusok, tagtípus

```
package java.util.stream;
public interface IntStream {
    ...
    default IntStream dropWhile( IntPredicate predicate ){
        ...
    }
    static IntStream iterate( int seed, IntUnaryOperator f ){
        ...
    }
    public static interface Builder {
        void accept( int v );
        ...
    }
}
```

Marker interface

- Metódusok nélküli
- Mégis jelent valamit

```
package java.lang;
public interface Cloneable {}
```

```
package java.io;
public interface Serializable {}
```

Annotációtípusok

```
@Documented
@Retention(RUNTIME)
@Target(TYPE)
public @interface FunctionalInterface {}
```

Programozási nyelvek – Java

Egyenlőségvizsgálat

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Object identity

```
Time t1 = new Time(13,30);
Time t2 = new Time(13,30);
System.out.println( t1 == t2 );

t2 = t1;
System.out.println( t1 == t2 );
```

Tartalmi egyenlőségvizsgálat referenciatípusokra?

```
Time t1 = new Time(13,30);
Time t2 = new Time(13,30);

System.out.println( t1 == t2 );

System.out.println( t1.equals(t2) );
```

Tartalmi egyenlőségvizsgálat referenciatípusokra

```
ArrayList<Integer> seq1 = new ArrayList<>();
seq1.add(1984); seq1.add(2001);

ArrayList<Integer> seq2 = seq1;
System.out.println( seq1 == seq2 );

seq2 = new ArrayList<>();
seq2.add(1984); seq2.add(2001);

System.out.println( seq1 == seq2 );

System.out.println( seq1.equals(seq2) );
```

„Az” equals-metódus

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

- Felüldefiniálható (pl. `Time`-ra is)
- Egy metódus sok (rész)implementációval
- Együttesen adnak egy összetett implementációt

„Az” equals-metódus

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

- Felüldefiniálható (pl. `Time`-ra is)
- Egy metódus sok (rész)implementációval
- Együttesen adnak egy összetett implementációt

- ... ha jól csináljuk!

Az `equals` szerződése betartandó

- Determinisztikus
- Ekvivalencia-reláció (RST)
- Ha `a != null`, akkor `!a.equals(null)`
  - Viszont `null.equals(a)` $\equiv$ `NullPointerException`
- Konzisztens a `hashCode()` metódussal
  - egyenlő objektumok hashCode-ja egyezzen meg
  - [különböző objektumok hashCode-ja jó, ha különböző]

Alapértelmezett viselkedés

```
package java.lang;
public class Object {
    ...

    public boolean equals( Object that ){
        return this == that;
    }

    public int hashCode(){ ... }
}
```

Szabályos felüldefiniálás

```
public class Time {
    ...

    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }

    @Override public int hashCode(){ return 60*hour + minute; }
}
```

Szabályos felüldefiniálás + „előző sáv”

```
public class Time {
    ...

    @Override public boolean equals( Object that ){
        if( this == that ) return true;
        if( that != null && getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }

    @Override public int hashCode(){ return 60*hour + minute; }
}
```

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 Time {
    ...
    @Override public boolean equals( Time that ){
        return that != null && hour == that.hour && ...
    }
    @Override public int hashCode(){ return 60*hour + minute; }
}
```

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 Time {
    ...
    public boolean equals( Time that ){
        return that != null && hour == that.hour && ...
    }
    @Override public int hashCode(){ return 60*hour + minute; }
}
```

`@Override` jelentősége

- Explicit módon kifejezi a programozó szándékát
- A fordítóprogram szól, ha elrontottuk a felüldefiniálást

Használjuk!

Túlterhelés altípuson

```
static void connect( Employee e, Manager m ){
    m.addUnderling(e);
}
static void connect( Manager m, Employee e ){
    m.addUnderling(e);
}
```

Túlterhelés altípuson

```
static void connect( Employee e, Manager m ){
    m.addUnderling(e);
}
static void connect( Manager m, Employee e ){
    m.addUnderling(e);
}

Employee eric = new Employee("Eric",12000);
Manager mary = new Manager("Mary",14000);

connect(eric,mary);      connect(mary,eric);
```

Túlterhelés altípuson

```
static void connect( Employee e, Manager m ){
    m.addUnderling(e);
}
static void connect( Manager m, Employee e ){
    m.addUnderling(e);
}

Employee eric = new Employee("Eric",12000);
Manager mary = new Manager("Mary",14000);

connect(eric,mary);       connect(mary,eric);

Manager mike = new Manager("Mike",13000);
connect(mike,mary);
```

Túlterhelés altípuson

```
static void connect( Employee e, Manager m ){
    m.addUnderling(e);
}
static void connect( Manager m, Employee e ){
    m.addUnderling(e);
}

Employee eric = new Employee("Eric",12000);
Manager mary = new Manager("Mary",14000);

connect(eric,mary);       connect(mary,eric);

Manager mike = new Manager("Mike",13000);
connect(mike,mary);

connect(mike,(Employee)mary);
```

Ökölszabály

Soha ne terheljünk túl altípuson!

Ökölszabály

Soha ne terheljünk túl altípuson!

```
class Object {
    public boolean equals( Object that ){ ... }
    ...
}
class Time {
    public boolean equals( Time that ){ ... }
    ...
}
```

Ökölszabály

Soha ne terheljünk túl altípuson!

```
class Object {
    public boolean equals( Object that ){ ... }
    ...
}
class Time {
    public boolean equals( Time that ){ ... }
    ...
}

Time t = new Time(11,22);
Object o = new Time(11,22);

t.equals(t)     t.equals(o)     o.equals(t)     o.equals(o)
```

Öröklődés és equals

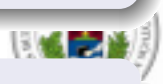
```
public class Time { ...
    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
    @Override public int hashCode(){ return 60*hour + minute; }
}
```

```
public class ExactTime extends Time { ...
    @Override public boolean equals( Object that ){
        return super.equals(that) && second == ((ExactTime)that).second;
    }
    @Override public int hashCode(){
        return 60*super.hashCode() + second;
    }
}
```

Altípusosság?

```
public class Time {
    ...
    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
}
```

```
public class ExactTime extends Time {
    ...
    @Override public boolean equals( Object that ){
        return super.equals(that) && second == ((ExactTime)that).second;
    }
}
```



```
new Time(11,22).equals( new ExactTime(11,22,33) )
```

instanceof + „előző sáv”

```
public class Time {
    ...
    @Override public boolean equals( Object that ){
        if( this == that ) return true;
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
    @Override public int hashCode(){ return 60*hour + minute; }
}
```

instanceof

```
public class Time {
    ...
    @Override public boolean equals( Object that ){
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
    @Override public int hashCode(){ return 60*hour + minute; }
}
```

```
Time t = new Time(11,22);
ExactTime e = new ExactTime(11,22,33);

t.equals(e)
```

„Igazi” egyenlőségvizsgálat

```
public class Time {
    ...
    @Override public boolean equals( Object that ){
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
    @Override public int hashCode(){ return 60*hour + minute; }
}
```

```
ExactTime e1 = new ExactTime(11,22,44);
ExactTime e2 = new ExactTime(11,22,33);

e1.equals(e2)
```

Szimmetria?

```
public class Time {
    @Override public boolean equals( Object that ){
        if( that instanceof Time ){ ...
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        ...
```

```
public class ExactTime extends Time {
    @Override public boolean equals( Object that ){
        if( that instanceof ExactTime ){ ...
            ExactTime t = (ExactTime)that;
            return super.equals(that) && second == t.second;
        ...
```

Szimmetria?

```
public class Time {
    @Override public boolean equals( Object that ){
        if( that instanceof Time ){ ...
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        ...
```

```
public class ExactTime extends Time {
    @Override public boolean equals( Object that ){
        if( that instanceof ExactTime ){ ...
            ExactTime t = (ExactTime)that;
            return super.equals(that) && second == t.second;
        ...
```

```
Time t = new Time(11,22), e = new ExactTime(11,22,33);
t.equals(e)       e.equals(t)
```

Tranzitivitás?

```
public class ExactTime extends Time {
    @Override public boolean equals( Object that ){
        if( that instanceof ExactTime ){ ...
            ExactTime t = (ExactTime)that;
            return super.equals(that) && second == t.second;
        } else if( that instanceof Time ){
            return that.equals(this);
        } else {
            return false;
    ...
```

Tranzitivitás?

```
public class ExactTime extends Time {
    @Override public boolean equals( Object that ){
        if( that instanceof ExactTime ){ ...
            ExactTime t = (ExactTime)that;
            return super.equals(that) && second == t.second;
        } else if( that instanceof Time ){
            return that.equals(this);
        } else {
            return false;
    ...
```

```
Time t = new Time(11,22);
ExactTime e1 = new ExactTime(11,22,33),
          e2 = new ExactTime(11,22,44);
e1.equals(t)       t.equals(e2)       e1.equals(e2)
```

`final` metódus

- nem definiálható felül

final metódus

- nem definiálható felül

```
public class Time {
    ...
    @Override public final boolean equals( Object that ){
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
}
```

final metódus

- nem definiálható felül

```
public class Time {
    ...
    @Override public final boolean equals( Object that ){
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
}
```

Fordítási hiba

```
public class ExactTime extends Time {
    @Override public boolean equals( Object that ){
...
```

final metódus

- nem definiálható felül

```
public class Time {
    ...
    @Override public final boolean equals( Object that ){
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
}
```

```
ExactTime e1 = new ExactTime(11,22,44);
ExactTime e2 = new ExactTime(11,22,33);

e1.equals(e2)  // RST, de nem ,,igazi" egyenlőség
```

final class

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

- Nem lehet belőle leszármaztatni
- Nem lehet specializálni, felüldefiniással belepiszkálni, elrontani
- Módosíthatatlan (immutable) esetben nagyon hasznos
- `java.lang.Class`, `java.lang.Integer` és egyéb csomagoló osztályok, `java.math.BigInteger` stb.

final class: végleges egyenlőségvizsgálat

```
public final class Time {
    ...
    @Override public boolean equals( Object that ){
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
}
```

final class: végleges egyenlőségvizsgálat

```
public final class Time {
    ...
    @Override public boolean equals( Object that ){
        if( that instanceof Time ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
}
```

Fordítási hiba

```
public class ExactTime extends Time { ... }
```

Öröklődésre tervezés

- Ha azt akarjuk, hogy egy osztályból lehessen újabbakat származtatni, tervezzük olyanra!
  - `equals`
  - `protected` láthatóság
  - legyen jól dokumentált, hogyan kell származtatni belőle
  - legyen időtálló
- Ha nem akarjuk, hogy származtassanak belőle, tegyük `final`lé!

Öröklődés kiváltása kompozícióval

```
public class ExactTime {
    private final Time time;
    private int second;
    public ExactTime( int hour, int minute, int second ){
        time = new Time(hour,minute);
        if( 0 <= second && second < 60) this.second = second;
        else throw new IllegalArgumentException();
    }
    public int getSecond(){ return second; }
    public int getMinute(){ return time.getMinute(); }
    public void aMinutePassed(){ time.aMinutePassed(); }
    ...
}
```

Öröklődés kiváltása kompozícióval: egyenlőség

```
public final class ExactTime {
    private final Time time;
    private int second;
    ...
    @Override public boolean equals( Object that ){
        if( this == that ) return true;
        if( that instanceof ExactTime ){
            ExactTime et = (ExactTime) that;
            return time.equals(et.time) && second == et.second;
        } else return false;
    }
}
```

Heterogén egyenlőség

```
ArrayList<Integer> aList = new ArrayList<>();
LinkedList<Integer> lList = new LinkedList<>();

aList.add(19);
lList.add(20-1);

aList.equals(lList)
```

Adatszerkezetek

- `ArrayList`, `HashSet`, `HashMap`
- Az `equals` és a `hashCode` helyességén alapszanak

```
ArrayList.contains(item)
HashSet.add(item)
HashMap.get(key)
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]

set.remove(new Time(5,30)); System.out.println(set);  // []
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]

set.remove(new Time(5,30)); System.out.println(set);  // []

set.add(t);
t.setHour(6);
set.remove(new Time(5,30));
System.out.println(set);                          // [6:30]
```

Mutable objektum adatszerkezetben

```
Time t = new Time(5,30);
HashSet<Time> set = new HashSet<>();

set.add(t); set.add(t); System.out.println(set);  // [5:30]

set.remove(new Time(5,30)); System.out.println(set);  // []

set.add(t);
t.setHour(6);
set.remove(new Time(5,30));
System.out.println(set);                          // [6:30]

set.remove(new Time(6,30));
System.out.println(set);                          // [6:30]
```

Stringek egyenlőségvizsgálata

```
String verb = "ring";
String noun = "ring";

verb.equals(noun)
verb == noun
```

Stringek egyenlőségvizsgálata

```
String verb = "ring";
String noun = "ring";

verb.equals(noun)
verb == noun

String mathematical = new String("ring");
noun.equals(mathematical)
noun == mathematical
```

Stringek egyenlőségvizsgálata

```
String verb = "ring";
String noun = "ring";

verb.equals(noun)
verb == noun

String mathematical = new String("ring");
noun.equals(mathematical)
noun == mathematical
```

Használjunk mindig equals()-t!

Integerek egyenlőségvizsgálata

```
Integer nineteen = 19;
Integer twentyButOne = 20-1;

nineteen.equals(twentyButOne)
nineteen == twentyButOne
```

Integerek egyenlőségvizsgálata

```
Integer nineteen = 19;
Integer twentyButOne = 20-1;

nineteen.equals(twentyButOne)
nineteen == twentyButOne

Integer dog = -123456;
Integer pup = -123456;

dog.equals(pup)
dog == pup
```

Integerek egyenlőségvizsgálata

```
Integer nineteen = 19;
Integer twentyButOne = 20-1;

nineteen.equals(twentyButOne)
nineteen == twentyButOne

Integer dog = -123456;
Integer pup = -123456;

dog.equals(pup)
dog == pup
```

Használjunk mindig equals()-t!

Objektumok tartalmi összehasonlítása

- Használjunk mindig equals()-t!

Objektumok tartalmi összehasonlítása

- Használjunk mindig equals()-t!

Felsorolási típus

- Garantáltan működik az == is.

```
enum Color { RED, WHITE, GREEN }
...
if( color1 == color2 ) ...
```

Objektumok tartalmi összehasonlítása

- Használjunk mindig equals()-t!

Felsorolási típus

- Garantáltan működik az == is.

```
enum Color { RED, WHITE, GREEN }
...
if( color1 == color2 ) ...
```

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

Tömbök összehasonlítása

```
int[] x = {1,2}, y = {1,2};
! x.equals(y)
```

Tömbök összehasonlítása

```
int[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean is_equal( int[] x, int[] y ){
    if( x == y ){ return true; }
    if( x == null || y == null || x.length != y.length ){
        return false;
    }
    for( int i=0; i<x.length; ++i ){
        if( x[i] != y[i] ){ return false; }
    }
    return true;
}
```

Tömbök összehasonlítása

```
int[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean is_equal( int[] x, int[] y ){
    if( x == y ){ return true; }
    if( x == null || y == null || x.length != y.length ){
        return false;
    }
    for( int i=0; i<x.length; ++i ){
        if( x[i] != y[i] ){ return false; }
    }
    return true;
}
```

```
java.util.Arrays.equals(x,y)
```

Referenciák tömbjének összehasonlítása

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)
```

Referenciák tömbjének összehasonlítása

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean is_equal( Integer[] x, Integer[] y ){
    if( x == y ){ return true; }
    if( x == null || y == null || x.length != y.length ){
        return false;
    }
    for( int i=0; i<x.length; ++i ){
        if( !x[i].equals(y[i]) )
            return false;
    }
    return true;
}
```

Referenciák tömbjének összehasonlítása: pontosabban

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)

static boolean is_equal( Integer[] x, Integer[] y ){
    if( x == y ){ return true; }
    if( x == null || y == null || x.length != y.length ){
        return false;
    }
    for( int i=0; i<x.length; ++i ){
        if( x[i] != y[i] && (x[i] == null || !x[i].equals(y[i])) )
            return false;
    }
    return true;
}
```

Referenciák tömbjének összehasonlítása: pontosabban

```
Integer[] x = {1,2}, y = {1,2};
! x.equals(y)
```

```
static boolean is_equal( Integer[] x, Integer[] y ){
    if( x == y ){ return true; }
    if( x == null || y == null || x.length != y.length ){
        return false;
    }
    for( int i=0; i<x.length; ++i ){
        if( x[i] != y[i] && (x[i] == null || !x[i].equals(y[i])) )
            return false;
    }
    return true;
}
```

```
java.util.Arrays.equals(x,y)
```

Tömbök tömbjének összehasonlítása

```
int[][] x = {{1,2}}, y = {{1,2}};
! x.equals(y)
! java.util.Arrays.equals(x,y)
java.util.Arrays.deepEquals(x,y)
```

Tömbök tömbjének összehasonlítása

```
int[][] x = {{1,2}}, y = {{1,2}};
! x.equals(y)
! java.util.Arrays.equals(x,y)
java.util.Arrays.deepEquals(x,y)

int[][][] x = {{{1,2}}}, y = {{{1,2}}};
java.util.Arrays.deepEquals(x,y)
```

Egyenlőségvizsgálat primitív mezők esetén

```
public class Time {
    ...

    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }

    @Override public int hashCode(){ return 60*hour + minute; }
}
```

Mély vizsgálat referencia típusú mezőkre

```
public class Interval {
    private Time from, to;
    ...
    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Interval u = (Interval)that;
            return from.equals( u.from ) && to.equals( u.to );
        } else return false;
    }
    ...
}
```

- Csak ha `from` és `to` nem lehet `null`!

java.util.Objects osztály

```
java.util.Objects.equals( Object a, Object b )
java.util.Objects.deepEquals( Object a, Object b )
java.util.Objects.hash( Object a... )
```

null-okat toleráló equals és jó hashCode

```
import java.util.Objects;
public class Interval {
    private Time from, to;   // nulls are allowed
    ...
    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Interval u = (Interval)that;
            return Objects.equals( from, u.from ) &&
                   Objects.equals( to, u.to );
        } else return false;
    }
    @Override public int hashCode(){
        return Objects.hash( from, to );
    }
}
```

Programozási nyelvek – Java

Típushierarchia

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

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 kötés: csak példánymetódusra

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

Kell a kitüntetett paraméter (dinamikus típusa)

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

Outline

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<String> { ... }
```

OK

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

Hibás

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

Hipotetikusan

```
class Base1 {
    int x;
    void setX( int x ){ this.x = x; }
    ...
}

class Base2 {
    int x;
    void setX( int x ){ this.x = x; }
    ...
}

class Sub extends Base1, Base2 { ... }
```

Hipotetikusan: diamond-shaped inheritance

```
class Base0 {
    int x;
    void setX( int x ){ this.x = x; }
    ...
}

class Base1 extends Base0 { ... }

class Base2 extends Base0 { ... }

class Sub extends Base1, Base2 { ... }
```

Különbség class és interface között

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

Többszörös öröklés interfészekből

```
interface Base1 {
    abstract void setX( int x );
    ...
}

interface Base2 {
    abstract void setX( int x );
    ...
}

class Sub implements Base1, Base2 {
    public void setX( int x ){ ... }
    ...
}
```

Outline

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
    ...
}
```

Részleges megvalósítás

```
public abstract class AbstractCollection<E> ... {
    ...
    public abstract int size();
    public boolean isEmpty(){
        return size() == 0;
    }
    public abstract Iterator<E> iterator();
    public boolean contains( Object o ){
        Iterator<E> iterator = iterator();
        while( iterator.hasNext() ){
            E e = iterator.next();
            if( o==null ? e==null : o.equals(e) ) return true;
        }
        return false;
    }
}
```

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
}
```

Outline

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` ⊆ félnyilvános (package-private) ⊆ `protected` ⊆ `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;
    }
}
```

Outline

Öröklődés $\Rightarrow$ altípusosság

`class A implements I`

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

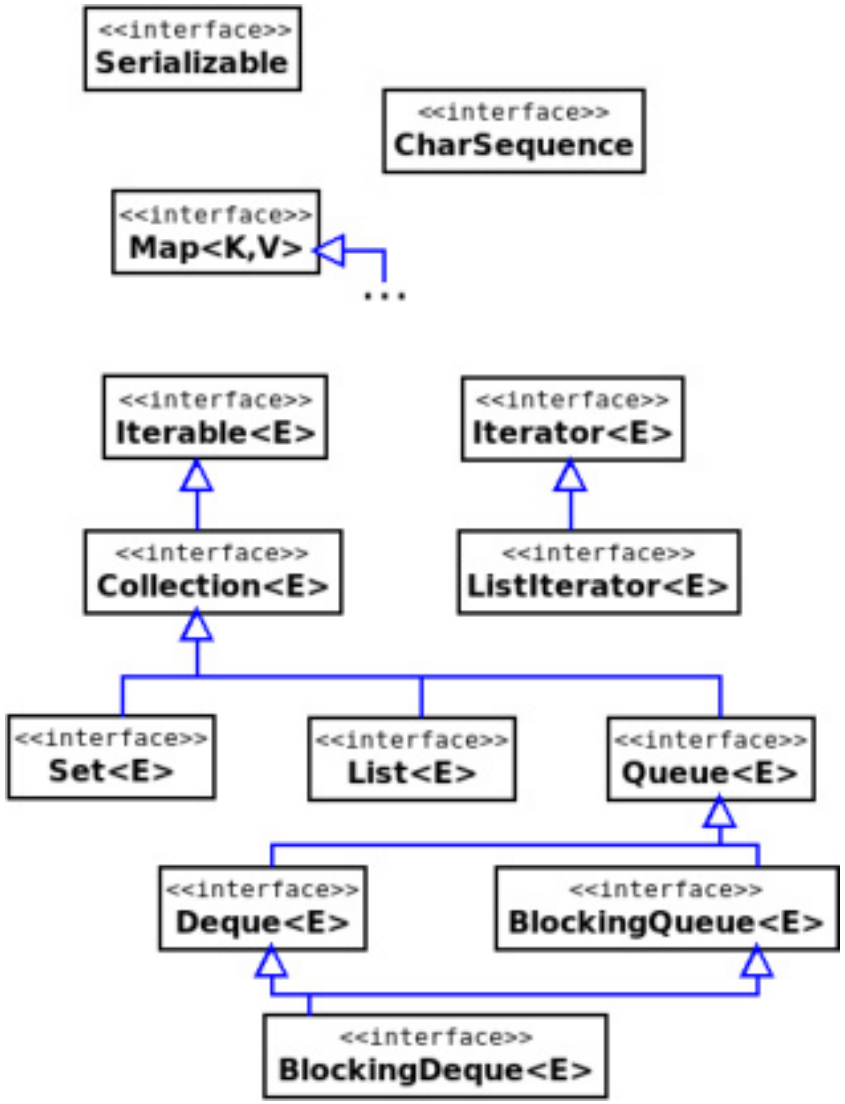
`class A extends B`

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

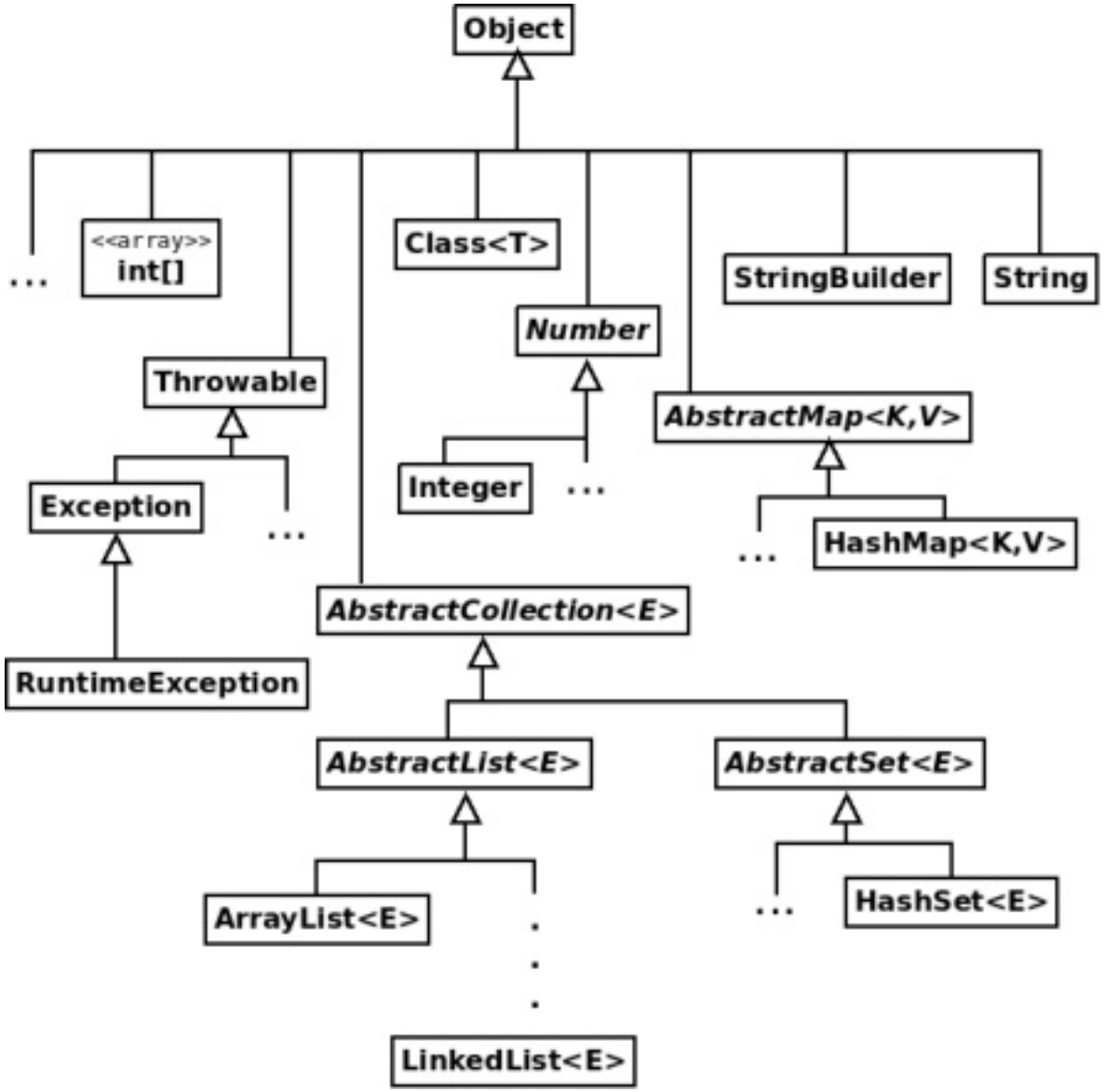
`interface I extends J`

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

Interface-ek hierarchiája a Javában (részlet)



Osztályok hierarchiája a Javában (részlet)



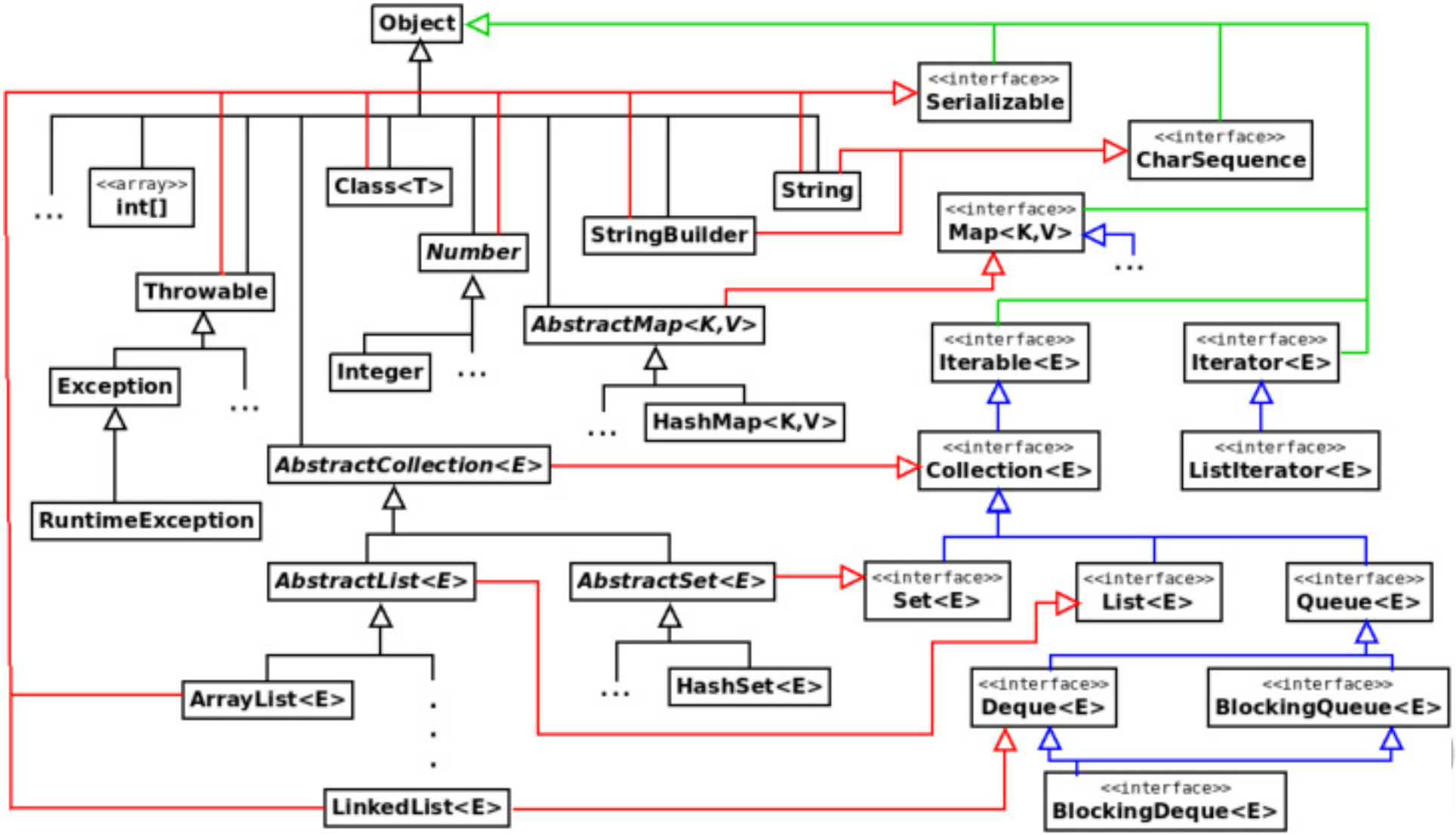
java.lang.Object

Minden osztály belőle származik, kivéve önmagát!

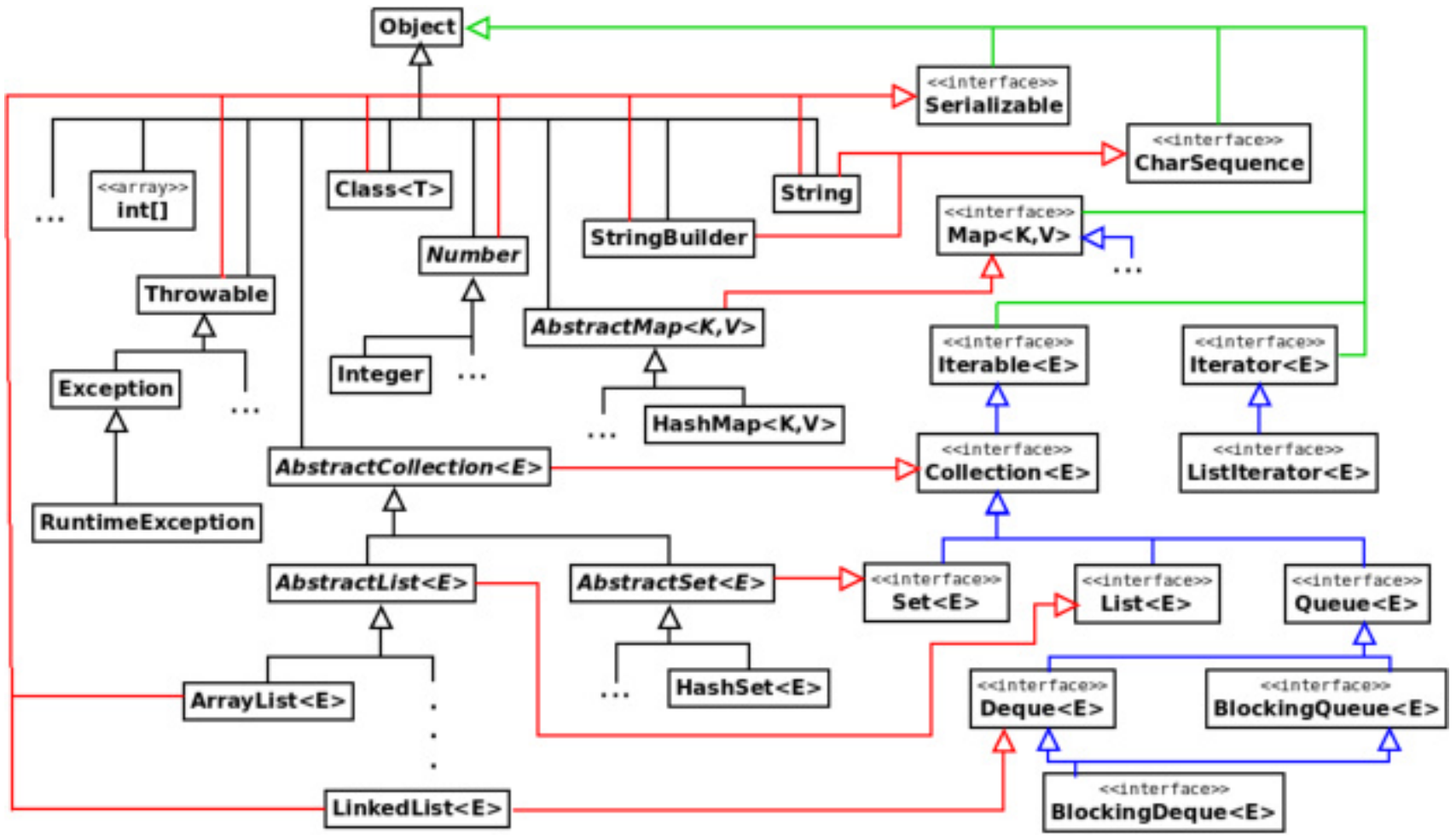
```
package java.lang;
public class Object {
    public Object(){ ... }
    public String toString(){ ... }
    public int hashCode(){ ... }
    public boolean equals( Object that ){ ... }
    public Class getClass(){ ... }
    ...
}
```

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

Outline

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
- `A a; B b; ... b = a;` helyes

Altípusos polimorfizmus (subtype polymorphism)

Ha egy kódbázist megírtunk, újrahasznosíthatjuk speciális típusokra!

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

Újrafelhasználhatóság!

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`

Többszörös kódöröklés?

- Kódöröklés: osztályok mentén
  - csak egyszeres öröklődés

Többszörös kódöröklés?

- Kódöröklés: osztályok mentén
  - csak egyszeres öröklődés
- `interface`-ekből
  - többszörös öröklődés
  - Korlátozott mértékű kódöröklés: `default` implementációjú példánymetódusok

Outline

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

Motiváló példa

```
package java.lang;
public interface CharSequence {
    int length();
    char charAt( int index );
    ...
}
```

Implementáló osztályok

- java.lang.String
- java.lang.StringBuilder
- java.lang.StringBuffer
- ...

Írjunk lexikografikus összehasonlítást!

```
static boolean less( CharSequence left, CharSequence right ){ ... }
```

Elegendő az altípusos polimorfizmus

```
static boolean less( CharSequence left, CharSequence right ){ ... }
```

```
less( "cool", "hot" )

StringBuilder sb1 = new StringBuilder(); ...
StringBuilder sb2 = new StringBuilder(); ...
less( sb1, sb2 )

less( "cool", sb1 )
```

Nem elegendő az altípusos polimorfizmus

```
static boolean less( CharSequence left, CharSequence right ){ ... }

static CharSequence min( CharSequence left, CharSequence right ){
    return less(left,right) ? left : right;
}
```

Nem elegendő az altípusos polimorfizmus

```
static boolean less( CharSequence left, CharSequence right ){ ... }

static CharSequence min( CharSequence left, CharSequence right ){
    return less(left,right) ? left : right;
}
```

OK

```
CharSequence cs = min( "cool", "hot" );
```

Fordítási hiba

```
String str = min( "cool", "hot" );
```

Parametrikus polimorfizmus

```
static boolean less( CharSequence left, CharSequence right ){ ... }

static <T> T min( T left, T right ){
    return less(left,right) ? left : right;
}
```

Fordítási hiba: `less`

Korlátozott univerzális kvantálás

```
static boolean less( CharSequence left, CharSequence right ){ ... }

static <T extends CharSequence> T min( T left, T right ){
    return less(left,right) ? left : right;
}
```

Korlátozott univerzális kvantálás

```
static boolean less( CharSequence left, CharSequence right ){ ... }

static <T extends CharSequence> T min( T left, T right ){
    return less(left,right) ? left : right;
}
```

```
String str = min( "cool", "hot" );
StringBuilder sb = min( new StringBuilder(), new StringBuilder() );
CharSequence cs = min( "cool", new StringBuilder() );
```

Korlátozott univerzális kvantálás

```
static boolean less( CharSequence left, CharSequence right ){ ... }

static <T extends CharSequence> T min( T left, T right ){
    return less(left,right) ? left : right;
}
```

```
String str = min( "cool", "hot" );
StringBuilder sb = min( new StringBuilder(), new StringBuilder() );
CharSequence cs = min( "cool", new StringBuilder() );
```

- constrained genericity
- bounded universal quantification
- bounded parametric polymorphism
- $\forall T$-re, amely a `CharSequence`-ből származik, definiáljuk a `min` függvényt úgy, hogy...
- felső korlát (upper bound)

„Természetes rendezés” (natural ordering)

```
java.util.Arrays.sort(args)
```

„Természetes rendezés” (natural ordering)

```
java.util.Arrays.sort(args)
```

```
public final class String ... {
    ...
    public int compareTo( String that ){ ... }
}
```

```
public final class Integer ... {
    ...
    public int compareTo( Integer that ){ ... }
}
```

Természetes rendezés - interface

3-way comparison

```
package java.lang;
public interface Comparable<T> {    // negative: this < that
    int compareTo( T that );        //     zero: this = that
}                                   // positive: this > that
```

Természetes rendezés - interface

3-way comparison

```
package java.lang;
public interface Comparable<T> {    // negative: this < that
    int compareTo( T that );        //     zero: this = that
}                                   // positive: this > that
```

```
package java.lang;
public final class String implements Comparable<String> { ... }
// public int compareTo( String that ){ ... }
```

```
package java.lang;
public final class Integer implements Comparable<Integer> { ... }
// public int compareTo( Integer that ){ ... }
```

Saját természetes rendezés

3-way comparison

```
package java.lang;
public interface Comparable<T> {    // negative: this < that
    int compareTo( T that );        //     zero: this = that
}                                   // positive: this > that
```

```
public class Rational implements Comparable<Rational> {
    ...
    public int compareTo( Rational that ){
        /* class invariant: denominator > 0 */
        return numerator * that.denominator -
               that.numerator * denominator;
    }
}
```

Rendezhetőség öröklése

```
public class Rational implements Comparable<Rational> {
    ...
    public int compareTo( Rational that ){
        return numerator * that.denominator -
               that.numerator * denominator;
    }
}
```

```
public class CanonicalRational extends Rational { ... }
```

```
(new Rational(3,6)).compareTo(new Rational(5,9))
(new Rational(3,6)).compareTo(new CanonicalRational(5,9))
(new CanonicalRational(3,6)).compareTo(new CanonicalRational(5,9))
(new CanonicalRational(3,6)).compareTo(new Rational(5,9))
```

Probléma az öröklődéssel

Egy osztály nem implementálhatja ugyanazt a generikus interface-t többször, különböző típusparaméterekkel.

```
public class Time implements Comparable<Time> {
    ...
    public int compareTo( Time that ){ ... }
}
```

Fordítási hiba

```
public class ExactTime extends Time
                       implements Comparable<ExactTime> {
    ...
    public int compareTo( ExactTime that ){ ... }
}
```

Más összehasonlításhoz

```
@FunctionalInterface
public interface Comparator<T> {
    int compare( T left, T right );   // 3-way
}
```

Más összehasonlításhoz

```
@FunctionalInterface
public interface Comparator<T> {
    int compare( T left, T right );   // 3-way
}
```

```
class StringLengthComparator implements Comparator<String> {
    public int compare( String left, String right ){
        return left.length() - right.length();
    }
}
```

```
java.util.Arrays.sort( args, new StringLengthComparator() );
```

Más összehasonlításhoz

```
@FunctionalInterface
public interface Comparator<T> {
    int compare( T left, T right );   // 3-way
}
```

```
class StringLengthComparator implements Comparator<String> {
    public int compare( String left, String right ){
        return left.length() - right.length();
    }
}
```

```
java.util.Arrays.sort( args, new StringLengthComparator() );
```

```
java.util.Arrays.sort( args, (a,b) -> a.length()-b.length() );
```

Rendezés

Nyilvános műveletek a `java.util.Arrays` osztályban:

```
static <T> void parallelSort( T[] a, Comparator<? super T> cmp )
```

- `cmp`: létezik olyan S típus, amelynek altípusa a T, és ilyeneket tud összehasonlítani
  - egzisztenciális kvantálás (existential quantification)
  - alsó korlát (lower bound)

Rendezés

Nyilvános műveletek a `java.util.Arrays` osztályban:

```
static <T> void parallelSort( T[] a, Comparator<? super T> cmp )
```

- `cmp`: létezik olyan S típus, amelynek altípusa a T, és ilyeneket tud összehasonlítani
  - egzisztenciális kvantálás (existential quantification)
  - alsó korlát (lower bound)

```
static <T extends Comparable<? super T>> void parallelSort(T[] a)
```

- A T olyan típus legyen, amelynek van olyan bázistípusa, amely rendelkezik természetes rendezéssel

Altípus reláció paraméterezett típusokon

```
public class ArrayList<T> ... implements List<T> ...
```

∀ T-re: `ArrayList<T>` <: `List<T>`

- `ArrayList<String>` <: `List<String>`
- `ArrayList<Integer>` <: `List<Integer>`

Típusparaméter altípusossága?

Szabály

`List<Integer>` $\not<:$ `List<Object>`

Típusparaméter altípusossága?

Szabály

`List<Integer>` $\not<:$ `List<Object>`

Indirekt tegyük fel, hogy `List<Integer>` $<:$ `List<Object>`

```
List<Integer> nums = new ArrayList<Integer>();
nums.add( 42 );                  // Integer.valueOf(42)
List<Object> things = nums;      // indirekt feltevés
things.add( "forty-two" );       // String <: Object
Integer n = nums.get(1);         // hiba!
```

Tömbökre gyengébb szabály vonatkozik

A Java megengedi, hogy `Integer[]` $<:$ `Object[]` legyen!

Tömbökre gyengébb szabály vonatkozik

A Java megengedi, hogy `Integer[]` $<:$ `Object[]` legyen!

Paraméterezett típusokra

```
List<Integer> nums = new ArrayList<Integer>();
nums.add( 42 );                  // Integer.valueOf(42)
List<Object> things = nums;      // fordítási hiba
things.add( "forty-two" );       // String <: Object
Integer n = nums.get(1);         // hiba lenne
```

Tömbökre gyengébb szabály vonatkozik

A Java megengedi, hogy `Integer[]` $<:$ `Object[]` legyen!

Paraméterezett típusokra

```
List<Integer> nums = new ArrayList<Integer>();
nums.add( 42 );                  // Integer.valueOf(42)
List<Object> things = nums;      // fordítási hiba
things.add( "forty-two" );       // String <: Object
Integer n = nums.get(1);         // hiba lenne
```

Tömb típusokra

```
Integer[] nums = new Integer[2];
nums[0] =  42 ;                  // Integer.valueOf(42)
Object[] things = nums;          // szabályos
things[1] = "forty-two" ;        // ArrayStoreException
Integer n = nums[1];             // hiba lenne
```

Outline

Öröklődéssel definiált osztály

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

Példa: klónozás

```
package java.lang;
public interface Cloneable {}
```

```
package java.lang;
public class CloneNotSupportedException extends Exception { ... }
```

```
package java.lang;
public class Object {
    public boolean equals( Object that ){ return this == that; }
    ...
    protected Object clone() throws CloneNotSupportedException {
        if( this instanceof Cloneable ) return /* shallow copy */
        else throw new CloneNotSupportedException();
    }
}
```

Sekély másolat – első próbálkozás

```
public class Time implements Cloneable {
    private int hour, minute;
    public Time( int hour, int minute ){ ... }
    public void setHour( int hour ){ ... }
    ...
    @Override public String toString(){ ... }
    @Override public int hashCode(){ return 60*hour + minute; }
    @Override public boolean equals( Object that ){
        if( that != null && getClass().equals(that.getClass()) ){
            Time t = (Time)that;
            return hour == t.hour && minute == t.minute;
        } else return false;
    }
}
```

Kényelmetlen!

```
package java.lang;
public class Object {
    ...
    protected Object clone() throws CloneNotSupportedException ...
}
```

Nem hívható akárhol!

```
public class Time implements Cloneable {
    ...
    public static void main( String[] args ){
        Time t = new Time(12,30);
        try { Object o = t.clone(); }
        catch( CloneNotSupportedException e ){ assert false; }
    }
}
```

Újradeklarálva jó!

```
public class Time implements Cloneable {
    private int hour, minute;
    ...
    @Override public Time clone(){
        try { return (Time)super.clone(); }
        catch( CloneNotSupportedException e ){
            assert false; return null;
        }
    }
}
```

Szabályos felüldefiniálás

- Láthatóság bővíthető
- Visszatérési típus szűkíthető
- Bejelentett ellenőrzött kivételek szűkíthetők

Klónozás szabályai

`implements Cloneable`

- Tegyük nyilvánossá a `clone()`-t
  - felüldefiniálás
  - újradeklarálás

Klónozás szabályai

`implements Cloneable`

- Tegyük nyilvánossá a `clone()`-t
  - felüldefiniálás
  - újradeklarálás
- Mindig használjuk a `super.clone()`-t!
  - megőrzi a dinamikus típust
  - megörökölt `clone()` is jót ad vissza

Klónozás szabályai

`implements Cloneable`

- Tegyük nyilvánossá a `clone()`-t
  - felüldefiniálás
  - újradeklarálás
- Mindig használjuk a `super.clone()`-t!
  - megőrzi a dinamikus típust
  - megörökölt `clone()` is jót ad vissza
- Sekély másolat: maradhat ugyanaz az implementáció
  - mély(ebb) másolat: új implementáció
  - immutable tagot nem kell másolni

Mély másolás

```
public class Interval implements Cloneable {
    private Time from, to;
    public void setFrom( Time from ){
        this.from.setHour( from.getHour() );
        this.from.setMinute( from.getMinute() );
    }
    @Override public Interval clone(){
        try {   Interval that = (Interval)super.clone();
                that.from = that.from.clone();
                that.to = that.to.clone();
                return that;
        } catch( CloneNotSupportedException e ){
                assert false; return null;
        }
    }
    ...
```

Programozási nyelvek – Java

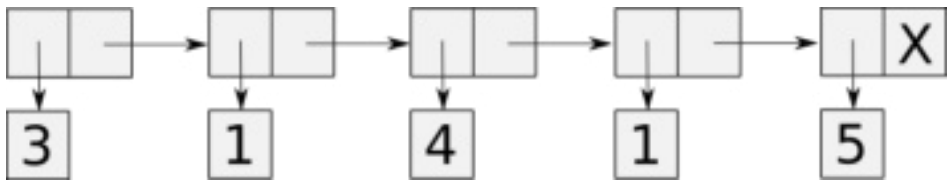
Típusbeágyazás

Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Saját fejlesztésű *sorozat* osztály

```
package datastructures;
public class Sequence<E> {
    public void insert( int index, E element ){ ... }
    public E get( int index ){ ... }
    public E remove( int index ){ ... }
    public int length(){ ... }
}
```



Megvalósítás láncolt listával

datastructures/Sequence.java

```
package datastructures;
public class Sequence<E> {
    private int size = 0;
    private Node<E> first = null;
    ...
}
```

datastructures/Node.java

```
package datastructures;
class Node<E> {
    E data;
    Node<E> next;
    Node( E data, Node<E> next ){ ... }
}
```

Egy fordítási egységben több típusdefiníció

- Még mindig szükségtelenül sokan hozzáférnek a segédosztályhoz

datastructures/Sequence.java

```
package datastructures;
public class Sequence<E> {
    private int size = 0;
    private Node<E> first = null;
    ...
}

class Node<E> {
    E data;
    Node<E> next;
    Node( E data, Node<E> next ){ ... }
}
```

Privát statikus tagosztály

datastructures/Sequence.java

```
package datastructures;

public class Sequence<E> {
    private int size = 0;
    private Node<E> first = null;
    ...

    private static class Node<E> {
        E data;
        Node<E> next;
        Node( E data, Node<E> next ){ ... }
    }
}
```

Statikus típusbeágyazás: java.util.Map.Entry

```
package java.util;

public interface Map<K,V> {

        public static interface Entry<K,V> {
                ...
        }

        ...

}
```

Iterátor statikus tagosztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator(){ return new SeqIt<>( this ); }

    private static class SeqIt<E> implements Iterator<E> {
        private Node<E> current;
        SeqIt( Sequence<E> seq ){ current = seq.first; }
        public boolean hasNext(){ return current != null; }
        public E next(){ ... }
    }
}
```

Példányszintű beágyazás

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator(){ return new SeqIt<>( this ); }

    private static class SeqIt<E> implements Iterator<E> {
        private Node<E> current;
        SeqIt( Sequence<E> seq ){ current = seq.first; }
        public boolean hasNext(){ return current != null; }
        public E next(){ ... }
    }
}
```

Iterátor példányszintű tagosztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator(){ return new SeqIt(); }

    private class SeqIt implements Iterator<E> {
        private Node<E> current = first;
        public boolean hasNext(){ return current != null; }
        public E next(){ ... }
    }
}
```

Iterátor példányszintű tagosztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator(){ return new SeqIt(); }

    private class SeqIt implements Iterator<E> {
        private Node<E> current = first;
        public boolean hasNext(){ return current != null; }
        public E next(){ ... }
    }
}
```

- Sequence.this.first

Iterátor lokális osztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator(){
        class SeqIt implements Iterator<E> {
            private Node<E> current = first;
            public boolean hasNext(){ return current != null; }
            public E next(){ ... }
        };
        return new SeqIt();
    }
}
```

Iterátor névtelen osztályként

```
...
public class Sequence<E> implements Iterable<E> {
    private static class Node<E> { ... }
    private Node<E> first = null;
    ...

    public Iterator<E> iterator(){
        return new Iterator<E>() {
            private Node<E> current = first;
            public boolean hasNext(){ return current != null; }
            public E next(){ ... }
        };
    }
}
```

Lambdák

```
@FunctionalInterface
public interface Comparator<T> {
    int compare( T left, T right );
}
```

```
java.util.Arrays.sort( args, (a,b) -> a.length()-b.length() );
```

Lambdák

```
@FunctionalInterface
public interface Comparator<T> {
    int compare( T left, T right );
}
```

```
java.util.Arrays.sort( args, (a,b) -> a.length()-b.length() );
```

```
java.util.Arrays.sort(
                args,
                new Comparator<String>() {
                    public int compare( String left, String right ){
                        return left.length() - right.length();
                    }
                }
);
```

Bájtkód

- `datastructures.Sequence.Node`:
  `datastructures/Sequence$Node.class`
- `datastructures.Sequence` első névtelen osztálya:
  `datastructures/Sequence$1.class`