Programozási nyelvek – Java
Első előadás

Kozsik Tamás

1 Tantárgyi követelmények

A tárgy célja

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

A tárgy célja, hogy programozási nyelvekkel kapcsolatos fogalmakkal ismertesse meg a hallgatókat (tudás), melyek alapján a hallgatók a programozás során képesek lesznek tudatosan választani a nyelvi eszközök közül (kompetencia). A tárgyalt ismeretkör az objektum-orientált és (kisebb részben, visszautalva) imperatív programozási paradigmát fedi le, valamint alapot teremt a későbbi, a konkurens programozási paradigmát tárgyaló kurzusnak. A tárgy szoros kapcsolatban áll a vele egy időben tartott *Objektumelvű programozás* kurzussal. Jelen tárgynak nem elsődleges célja, hogy a hallgatókat programozni tanítsa, de természetesen hozzájárul a hallgatók programozási készségeinek fejlődéséhez.

A gyakorlatokon Linux operációs rendszert használunk, így a tárgy hozzájárul ahhoz is, hogy a hallgatók felhasználói szinten megismerkedjenek ezzel az operációs rendszerrel, ezen belül is főleg a parancssor használatával. A nyelvi eszközök tudatos használatát azzal is erősíteni kívánja a tárgy, hogy nem integrált fejlesztői környezetben készítjük majd a programokat, hanem közönséges (programozói) szövegszerkesztőkben. Ennek köszönhetően a programírás nem az eszköz által felkínált lehetőségek közüli választás lesz, hanem egy sokkal tudatosabb folyamat.

Szervezés

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

Számonkérés

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

5. – [18, 20]

Információk

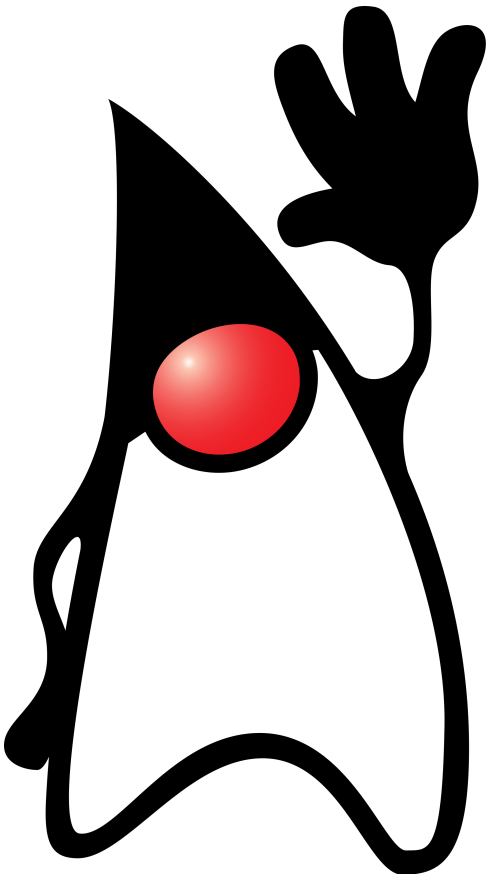
- Neptun
- Honlap
- Canvas
- BE-AD

2 Java – bevezetés

A Java nyelv

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

Java Language Specification



Jellemzői

- Könnyű/olcsó szoftverfejlesztés

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

JavaZone videó

Történelem

James Gosling és mások, 1991 (SUN Microsystems)

Oak → Green → **Java**

- Java 1.0 (1996)
- Java Community Process (1998)
- Java 1.2, J2SE (1998)
- J2EE (1999)
- J2SE 5.0 (2004)
- JVM GPL (2006)
- Oracle (2009)
- Java SE 8 (2014)
- Java SE 11 (2014)
- Game of Codes, Javazone 2014

Java Virtual Machine

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

Java Virtual Machine Specification

Egy pillantás a nyelvre

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

A szokásos „első program”így nézhet ki Javában. Kulcsszavak itt a `class`, a `public`, a `static` és a `void`. A Java osztályalapú objektumelvű nyelv, ezért a kód egy úgynevezett *osztálydefinícióba* van szervezve, amit a `class` kulcsszó vezet be. A programunk nevének az osztálynevet tekintjük, esetünkben ez a `HelloWorld`. A főprogram egy olyan metódus (így hívjuk az alprogramokat a Javában) lesz, amelynek a neve `main`, és lényegében pont úgy kell deklarálni, mint ahogy ebben a példában tettük: kell elé a `public static void`, egyetlen paramétere kell legyen, és ennek a paraméternek a típusa „sztringekből álló tömb” kell legyen. Természetesen ez felel majd meg a parancssori argumentumoknak.

A program tagolásában fontos szerephez jutnak a kapcsos zárójelek. Nem csak a metódusok törzsét és a blokk utasításokat, de az osztály törzsét is kapcsos zárójelekkel határoljuk. A kapcsos zárójelek elhelyezését szabályozó kódolási konvenciót jól megfigyelhetjük ezen a példán.

Egy másik kódolási konvenció az azonositók megválasztását szabályozza. Az osztályok, metódusok, változók nevét camel-case-ben írjuk. Az osztályok neve nagybetűvel, a metódusok és változók neve kisbetűvel szokott kezdődni.

A képernyőre történő kiíráshoz a `System.out.println` műveletet használjuk. Ez felel meg a Python `print`-jének, illetve a C `printf`-jének. A `println` végén a „-ln” a *line* szóra utal, arra, hogy egy soremelést is kiír.

A `void` kulcsszó ismerős lehet C-ből: ez jelzi azt, hogy a `main` nem ad vissza semmit. (Szemben a C-vel, ahol a `main`-nek egy egész számot kell visszaadnia.)

A `static` kulcsszót is használjuk C-ben, de itt picit más a jelentése. Egyelőre tekintsünk rá úgy, mint annak a jelzése, hogy a szóban forgó metódus nem „igazi” metódus, hanem inkább olyan, mint a C-ben egy (globálisan definiált) függvény.

A `public` kulcsszó értelme is kitalálható: az adott művelet láthatóságát terjeszti ki. Arra emlékeztethet bennünket, mint amikor a C-ben azt mondtuk egy függvényre, hogy külső szerkesztésű (external linkage), azaz nem `static`. Könnyű itt picit összezavarodni, mert a kulcsszavak és a fogalmak hasonlóak ugyan a két nyelvben, de ugyanaz a kulcsszó egész más fogalmat jelölhet.

Objektumelvű programozás

Object-oriented programming (OOP)

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

Egységbezárás: objektum

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

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

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

- “Pont” objektum “eltolás”, “tükrözés” stb. műveletekkel
- “Racionális szám” aritmetikai műveletekkel
- “Sorozat” objektum “beszúrás”, “törlés”, “összefűzés” stb. művelettel
- “Ügyfél” objektum adatkarbantartó, -validáló, adatbázisba mentő műveletekkel

Osztály

Objektumok típusa

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

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

Példányosítás (instantiation)

- Objektum létrehozása osztály alapján
- Javában: mindig a heapen

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

Java programok felépítése

(első blikkre)

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

Java forrásfájl

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

Fordítás, futtatás

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

Parancssorban

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

Java programok futása

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

3 Objektumok Javában

Osztály, objektum, példányosítás

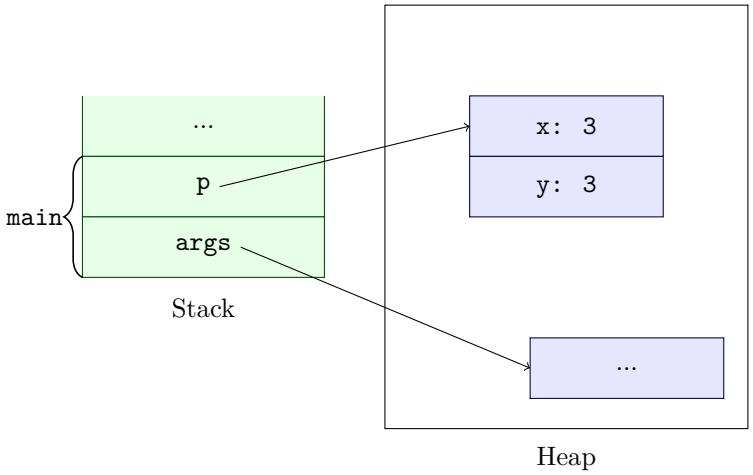
```
class Point {       // osztálydefiníció
    int x, y;       // mezők
}

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

Fordítás, futtatás

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

Stack és heap



Mezők inicializációja

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

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

Mező alapértelmezett inicializációja

Automatikusan egy nulla-szerű értékre!

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

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

3.1 Metódusok

Metódus

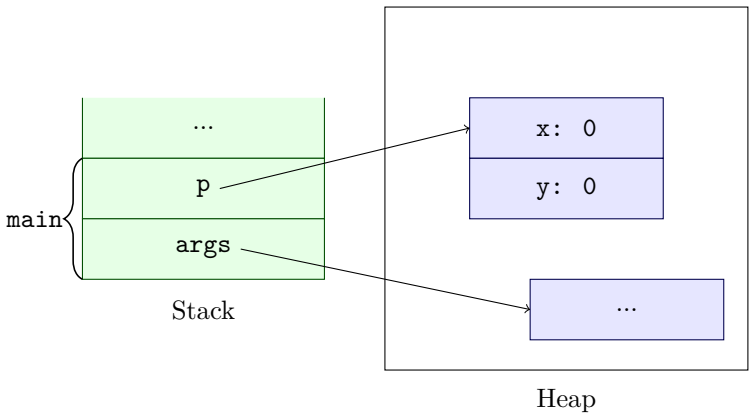
```
class Point {
    int x, y;   // 0, 0
    void move( int dx, int dy ){   // implicit paraméter: this
        this.x += dx;
        this.y += dy;
    }
```

```
}

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

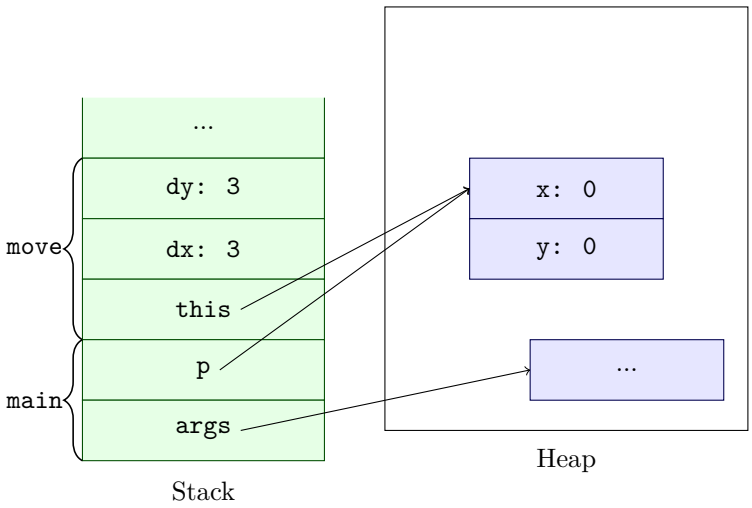
Metódus aktivációs rekordja – 1

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



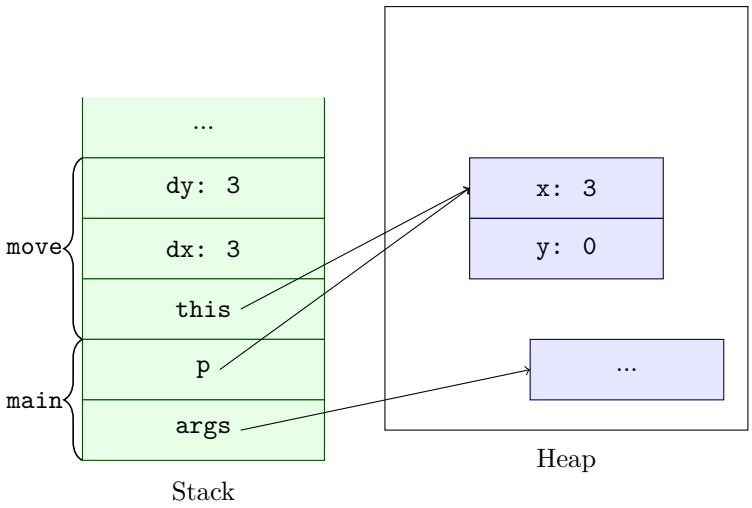
Metódus aktivációs rekordja – 2

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



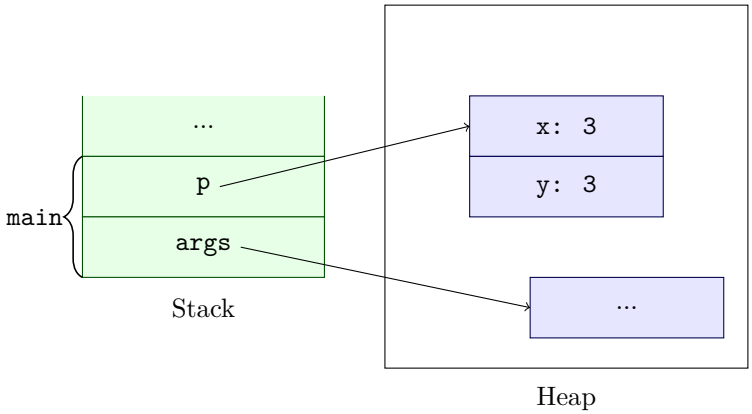
Metódus aktivációs rekordja – 3

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



Metódus aktivációs rekordja – 4

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



A `this` implicit lehet

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

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

3.2 Konstruktorok

Inicializálás konstruktorral

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

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

Inicializálás konstruktorral – a `this` elhagyható

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

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

Nevek újrahasznosítása

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

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

Paraméter nélküli konstruktor

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

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

Alapértelmezett (default) konstruktor

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

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

Generálodik egy paraméter nélküli, üres konstruktor

```
Pont(){}
```