Programming languages Java

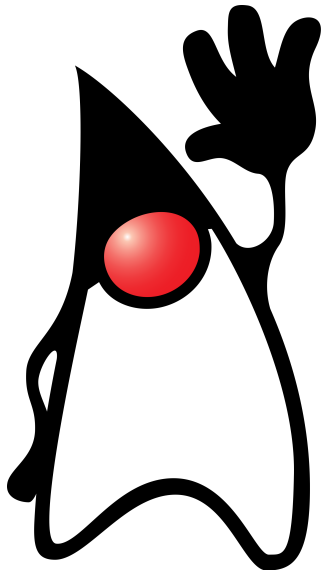
Basics

Kozsik Tamás

ELTE
IK

The Java language

- Syntax closely related to C
- Object-oriented
  - ◇ Class based
- Paradigm: mainly imperative
  - ◇ A bit also functional, too
- Compiles to bytecode (machine code of the Java VM)
- Strongly typed
- Static + dynamic type system
- Built-in language features for generics, concurrent programming



Java Language Specification

ELTE
IK

Properties

- Easy/cheap software development
- Rich infrastructure
  - ◇ Standard libraries
  - ◇ Third party libraries
  - ◇ Tools
  - ◇ Extensions
  - ◇ Documentation
- Platform independence (JVM)
  - ◇ Write once, run everywhere
  - ◇ **Compile** once, run everywhere
- Resource intensive

For fun: JavaZone video

ELTE
IK

History

James Gosling and others, 1991 (SUN Microsystems)

Oak → Green → **Java**

Java version history

- 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)
- LTS editions: Java SE 11 (2018), Java SE 17 (2021)

For fun: Game of Codes, Javazone 2014

Java Virtual Machine

- Machine code: bytecode
- Target of many languages (Ada, Closure, Eiffel, Jython, Kotlin, Scala...)
- Can be compiled further
  - ◇ Just In Time compilation
- Dynamic linking
- Code mobility

For details: Java Virtual Machine Specification

ELTE
IK

C vs Java: similarities

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

C vs Java: differences

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

C vs Java: more differences

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

Hello World!

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

Object-oriented programming

Object-oriented programming (OOP)

- Object
- Class
- Abstraction
  - ◇ Encapsulation
  - ◇ Information hiding
- Inheritance
- Subtyping, subtype polymorphism
- Overriding, dynamic binding

ELTE
IK

Encapsulation: object

Data and operations that can be performed on it (similar to `structs` in other languages)

- "Point" object
- "Rational number" object
- "List" object
- "Bank customer" object

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

ELTE
IK

Method

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

// in languages that do not have objects (like C)
p.x = 0;
p.y = 0;
move(p, 3, 5);
```

ELTE
IK

Class

A type in which similar objects belong

- "Point" class
- "Rational number" class
- "List" class
- "Bank customer" class

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

ELTE
IK

Instantiation

- The objects we create are instances of the class
  - ◇ The class is a blueprint of the object
- In Java, objects do not have pre-allocated storage
  - ◇ They are always dynamically created
  - ◇ They are stored on the heap

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

Example: texts and arrays

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

Basic structure of Java programs

(not all details are included)

- [module]
- package
- class
  - ◇ field
  - ◇ method
    - ▶ statement
      - expression

Java source file

- Most often has the name of the class
- `.java` extension
- Compilation unit
- In a folder appropriate to its package
- Character encoding

Compilation and execution

- The output of the compiler (object code) is JVM bytecode (`.class`)
- Loaded dynamically into JVM (not linked statically)
- Execution: interpreting bytecode + JIT

Using the command line

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

ELTE
IK

Executing Java programs

System memory is split in two distinct parts

- Execution stack
  - ◇ Activation records
  - ◇ Local variables
  - ◇ Parameter passing strategies
- Heap
  - ◇ Dynamic storage
  - ◇ Objects are stored here

Package

- Structures the program
- Groups related classes
- Program libraries
  - ◇ Standard library

The package declaration

```
package geometry;

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

- Fully-qualified name of the class: `geometry.Point`
- Short name of the class: `Point`

ELTE
IK

Namespace hierarchy

```
package geometry.basics;

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

- Standard libraries like `java.net.ServerSocket`
- `hu.elte.kto.teaching.javabsc.geometry.basics.Point`

ELTE
IK

Compilation and execution

- Working directory
- Package hierarchy → directory structure
- Compile from the working directory
  - ◇ Using filename with full/relative path
- Execution from the working directory
  - ◇ Using the fully-qualified class name

```
$ 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
$
```

Compilation: Java and 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
$
```

ELTE
IK

Recursive compilation

```
$ 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
$
```

Default package

Default/anonymous package

- When no `package` declaration is present
- Source file is in the working directory
- It is all right for small projects

Visibility categories

- `private`
  - ◇ accessible only from the class definition itself
  - ◇ hidden from everybody else
- no declaration (package-private)
  - ◇ accessible only from classes in the same package
- `public`
  - ◇ accessible from all classes

Public class with public and private members

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

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

Program with multiple classes

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

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

class Time { ... }
```

Main.java

```
// (in the default package)

public class Main {
  public static void main(String[] args) {
    hu.elte.kto.javabsc.theory.Time morning = new Time(6,10);
// compile error: no Time in the default pkg  ↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
  }
}
```

Within the same package

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

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

public class Time { ... }
```

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

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

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

Multiple type definitions in the same source file

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

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

class Time { ... }

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

The import declaration

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

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

public class Time { ... }
```

Main.java

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

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

Resolving class names

- Needed when the source code contains only the short class name
- `import hu.elte.kto.javabsc.theory.*;`
- Not transitive: works only within the same compilation unit
- The types from the package `java.lang` are automatically imported
- For clashing names: the full name is needed
  - ◇ `java.util.List`
  - ◇ `java.awt.List`

Structure of the compilation unit

- optional `package` declaration
- 0, 1 or more `import` declarations
- 1 or more type definitions

javac options

```
-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...
```

ELTE
IK

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
```

- If the `colors.RGB` class is needed:
  - `./colors/RGB.class`
  - `/usr/lib/java/colors/RGB.class`
  - in `/opt/java/myfiles.jar`: `colors/RGB.class`
- On Windows boxes: `.;C:\Users\kto\mylib;D:\myfiles.jar`
  - `CLASSPATH` environment variable

ELTE
IK

`jar` files

- Java Archive
- actually contains `.class` files in a `zip` archive
- Java SDK: `jar` tool
  - ◇ can also be created by any `zip` tool