2025 — 2026

1TI – 1ACS



UCLL HOGESCHOOL

Back-End Development

Java Introduction

N. Jacobs, J. Pieck, E. Steegmans, B. Van Impe, S. Van Peborgh, V. Witters

CONTENTS

- Situating Java
- First Java program
- Java building blocks
- Exercises

Situating Java

First Java program

Java building blocks

Exercises

Situating Java

WHAT IS JAVA

- A platform-independent programming language
  - Java code can be executed on many different machines and systems without modification
- An object-oriented programming language
  - Code is structured according to the concept of *objects*
  - We will discuss objects in detail in later classes
- A language suitable for large software applications
  - The language has concepts that allow structuring large applications
  - The Java ecosystem has mature tools and frameworks

WHAT IS JAVA

- An open-source language with a complex ecosystem
  - The functionality of each version is described in a set of specifications
    - Anyone can make a proposal to include a feature
    - Every 6 months, a check is made to see which features are ready to be included in the next release
  - A reference implementation of each version is provided by the OpenJDK community
  - Anyone is free to distribute their own (commercial or non-commercial) implementation



Situating Java

First Java program

Java building blocks

Exercises

First Java program

- Structure
- Life cycle
- Execute & Debug

FIRST JAVA PROGRAM

```
public class Application {

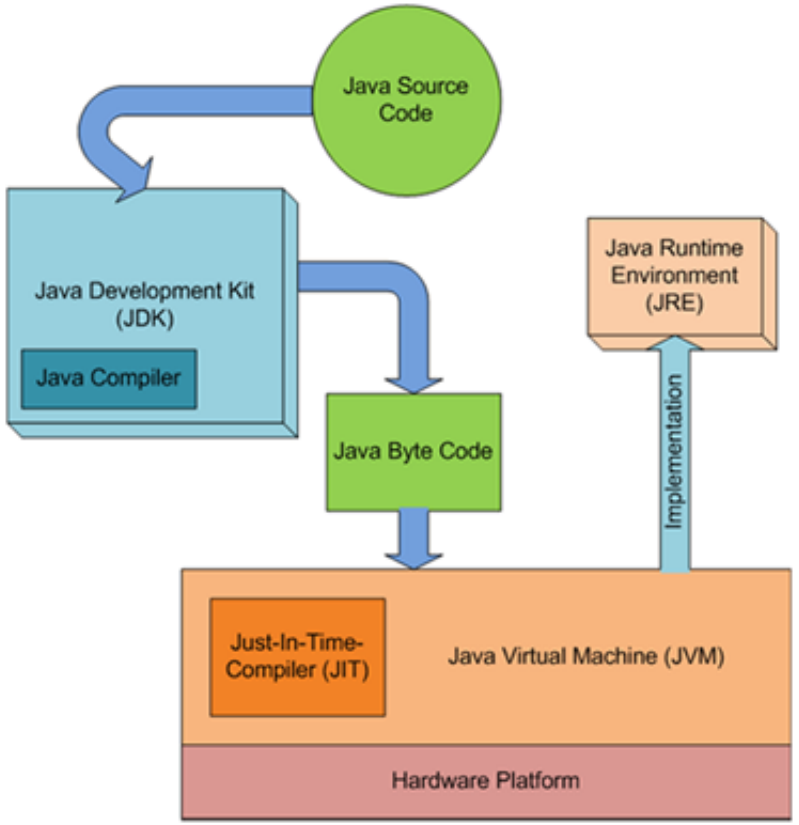
    public static void main(String [] args) {
        System.out.println("Hello!");
    }
}
```

- Minimal Java program that prints text to console
- Every Java program has (at least) one main *method*
  - This is the entry point of the program
  - The execution starts from the main method

FIRST JAVA PROGRAM

- Running Java code consists of two steps
  - First, the code is translated (more precisely, compiled) into platform-independent bytecode
  - Next, the bytecode is read by the **J**ava **V**irtual **M**achine (JVM) which translates it into specific instructions for the machine on which the program will run

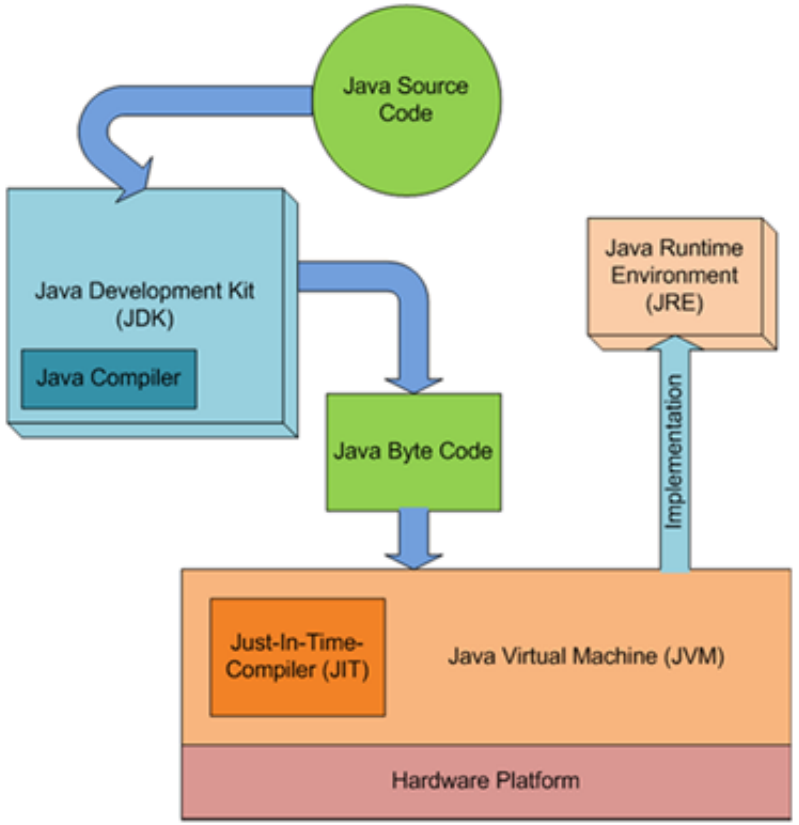
- Both steps can lead to errors
  - Compilation errors
  - Runtime errors



FIRST JAVA PROGRAM

- Java code is distributed as a bundle of bytecode files
- To run these files, you need a JVM for your system
- You obtain a JVM by installing a **J**ava **R**untime **E**ngine (JRE)

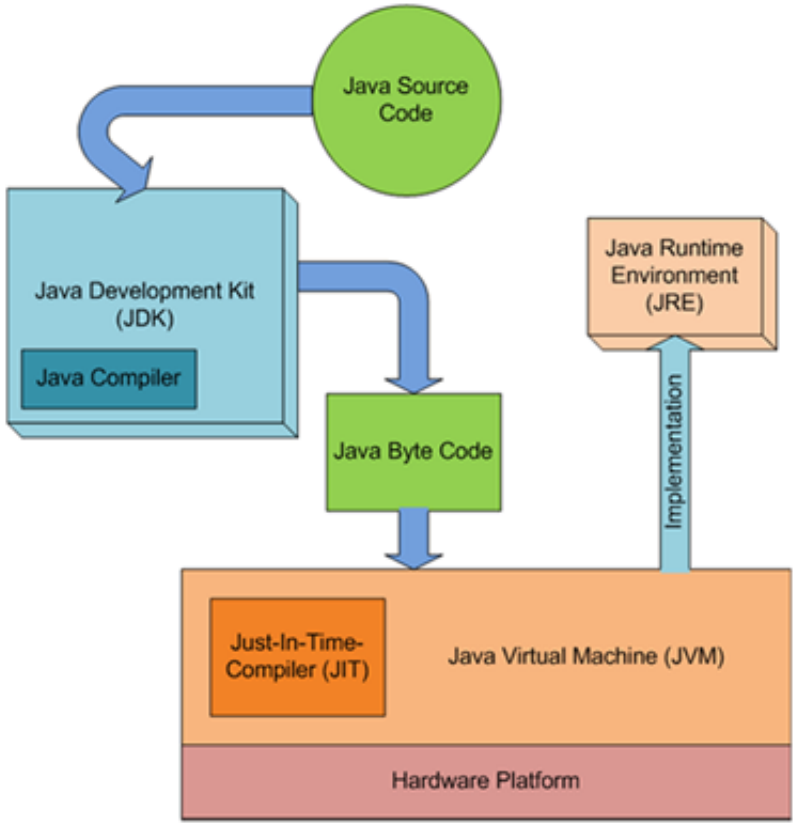
- You can start the JVM directly via the "java" command on the command-line



FIRST JAVA PROGRAM

- To develop Java programs, you need a **J**ava **D**evelopment **K**it (JDK)
- The JDK contains everything the JRE contains plus extra tools (compiler, debugger, ...)

- You can control the compiler directly via the "javac" command on the command-line



FIRST JAVA PROGRAM

```
public class Application {

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

- In summary, to run this program the following commands must be ran:
  - javac Application.java
    - Compiles code with the Java compiler to bytecode (JDK)
  - java Application
    - Starts the JVM and executes bytecode (JRE)

FIRST JAVA PROGRAM

- Command-line tools are not always easy to use (especially for larger projects)
- We will develop, run and debug code in an **IDE**
- An **I**ntegrated **D**evelopment **E**nvironment is a program that provides a single interface to control all aspects of development
  - Behind the scenes, this is done by controlling low-level tools
- As an IDE, we are going to use *IntelliJ*

EXERCISES @CLASS

- exercise1



Situating Java

First Java program

Java building blocks

Exercises

Java building blocks

- Statements & Comments
- Variables
- Control Structures
- Methods

STATEMENTS & COMMENTS

```
int a = 2; // Valid
int a = 2  // Invalid

/*
This is a comment
    across multiple lines
*/
```

- Each *statement* must end with a “;”
- A *comment* is indicated by “//” or “/* */”

VARIABLES

```
String text = "text is defined between double quotes";
int wholeNumber = 123;
double decimalNumber = 3.141592653;
boolean trueOrFalse = true;
```

- A variable is a container in which you can store information
- The name of a variable must be unique
  - The name is written in "camelCase"
- Every variable has a type
  - The type describes the nature of the data in the variable
  - Only values matching the type may be stored in the variable
- On creation, name and type must be specified

VARIABLES

```
int number = 123;
System.out.println(number);

number = 42;
System.out.println(number);
```

- The value of a variable is retained until a new value is assigned
- The new value must be of the same type as the variable
  - The type cannot be changed after creation

VARIABLES

```
public static void main(String[] args) {
    int number1 = 5;
    int number2 = number1;
    number1 = 10;

    System.out.println(number1);
    System.out.println(number2);
}
```

- If you assign a variable to another variable, the value is copied
- After the assign statement, the two variables are independent!

VARIABLES — TYPES

- Java has 8 *primitive* types

	Description
byte	A number between -128 and 127
short	A number between -32768 and 32767
int	A number between -2.147.483.648 and 2.147.483.647
long	A number between 9.223.372.036.854.775.808 --> 9.223.372.036.854.775.807
float	A large positive or negative decimal number
double	A gigantic positive or negative decimal number
boolean	The value “true” or “false”
char	A single text character

VARIABLES — TYPES

- Al other types are *classes*
  - An example is String
  - We will return to classes in detail later

- The type of a variable is either a primitive type or a class
  - This distinction is important when working with the variable
  - We will get back to this later

VARIABLES — NUMBERS

```
int calculation = (2 + 5) * ((1 + 4) - 3);
System.out.println(calculation);
```

- You can do calculations with the operators “+”, “-”, “*” and “/”
- “*” and “/” take precedence over “+” and “-”
- Parentheses allow you to change precedence rules

VARIABLES — NUMBERS

```
int number1 = 3;
int number2 = 2;
double number3 = 2.0;

System.out.println(number1 / number2);
System.out.println(number1 / number3);
System.out.println(number1 % number2);
```

- The types of the variables you use in a calculation determine the type of the result
  - All int -> result is also int
  - Int and double mixed -> result is double
- Take care when dividing!
  - Using “/” with integers -> integer division
  - % calculates remainder of the integer division

VARIABLES — NUMBERS

```
int number1 = (int) 3.2;
double number2 = (double) 2;

System.out.println(number1);
System.out.println(number2);
System.out.println((double) number1 + number2);
```

- You can change the type of a variable via *casting*
- Depending on the direction of the cast, this leads to loss of information
- You can cast anywhere in your program

VARIABLES — STRINGS

```
String startOfText = "you can paste text ";
String endOfText = "together with the + operator";
String fullText = startOfText + endOfText;
System.out.println(fullText);

String youCanConcatOtherTypes = "this is number " + 3;
System.out.println(youCanConcatOtherTypes);
```

- Strings can be concatenated using "+"
- You can also attach other types to a string with "+"
  - Behind the scenes, they are casted to a String

VARIABLES — STRINGS

```
int number1 = Integer.parseInt("3");
double number2 = Double.parseDouble("3.0");
boolean valueTrue = Boolean.parseBoolean("true");
int number3 = Integer.parseInt("this won't work");
```

- Casting Strings to other types requires the use of built-in methods

CONTROL STRUCTURES — IF

```
if (true) {
    System.out.println("This will be printed :-)");
}

if (false) {
    System.out.println("This won't be printed :-(");
}
```

- An if statement executes the code if the condition is met

CONTROL STRUCTURES — IF

```
int a = 5;
if (a < 4) {}
if (a == 5) {}
if (a != 5) {}
```

- In the condition, you can use comparison operators
  - *>, >=, <, <=*
  - *==*
  - *!=*
- The equality operator is to be used for primitive types only
  - For classes (such as Strings), use ".equals"

CONTROL STRUCTURES — IF ELSE

```
int a = 5;

// Statements using a ...

if (a > 4) {
    System.out.println("Will be printed if a > 4");
}  else {
    System.out.println("Will be printed if a <= 4");
}
```

- With *else*, you specify what should happen if the condition is not met

CONTROL STRUCTURES — IF ELSE

```
int a = 4;

// Statements using a ...

if (a < 4) {
    // ...
} else if (a == 4) {
    // ...
} else {
    // ...
}
```

- If-statements can be chained and closed with an else-statement
- The code of the first condition that is true is executed
- If all conditions are false than the else-statement is executed

CONTROL STRUCTURES — IF ELSE

```
int a = 4;
int b = 3;

// Statements using a & b ...

if (a < 4 && b > 4 || !((a - b) < 3)) {
    // ...
}
```

- You can write complex conditions with the help of logical operators
  - AND: &&
  - OR: ||
  - NOT: !

CONTROL STRUCTURES — LOOPS

```
for (int i = 0; i < 10; i++) {
    // Do stuff
}
```

```
boolean condition;
do {
    // Do stuff
} while (condition == true);
```

```
boolean condition = true;
while(condition) {
    // Do stuff

    if (...) {
        condition = false;
    }
}
```

- You can use loops to repeat similar code
- The different loop forms are functionally similar
  - You can achieve the same result with any loop form

METHODS

- A method is a named grouping of statements
- This group of statements can be called at other places in the code by using the name of the method

- Methods are used to structure programs

- Java contains pre-built methods (e.g. Integer.parseInt())
- You can also write your own methods

METHODS

```
public static void main(String [] args) {
    sayHello();
}

public static void sayHello() {
    System.out.println("Hello!");
}
```

- A method has a name and an implementation
- The name of the method should clearly indicate what the method does
- Calling a method can be done by writing the name followed by parentheses

METHODS — PARAMETERS

```
public static void main(String [] args) {
    sayAWord("Pancake");
}

public static void sayAWord(String word) {
    System.out.println(word + "!");
}
```

- A method may define parameters
  - A parameter is a value you can pass to a method
  - The parameter can then be used in the implementation of the method
- The value of the parameter is filled-in by copying the value when calling the method
  - We call this **pass-by-value**

METHODS — PARAMETERS

```
public static void main(String [] args) {
    sayTwoWords("Pancake", "Waffle");
}

public static void sayTwoWords(String word1, String word2) {
    System.out.println(word1 + "!");
    System.out.println(word2 + "!");
}
```

- A method can have multiple parameters
- Parameters are also referred to as (*method-)arguments*

METHODS — RETURN VALUE

```
public static void main(String [] args) {
    sayTwoWords("Pancake", "Waffle");
}

public static void sayTwoWords(String word1, String word2) {
    System.out.println(word1 + "!");
    System.out.println(word2 + "!");
}
```

- A method *may* return a value
  - If a method returns nothing, the keyword *void* is used

METHODS — RETURN VALUE

```
public static void main(String[] args) {
    System.out.println(count(1, 2));
}

public static int count(int n1, int n2) {
    return n1 + n2;
}
```

- A method *may* return a value
  - If a method returns something, the type of the return value is used
- To return something from the implementation, use the keyword *return*

METHODS — RETURN VALUE

```
public static int calcDifference(int n1, int n2) {
    if (n1 > n2) {
        return n1 - n2;
    } else {
        return n2 - n1;
    }

    int i = 2; // Error!
}
```

- A method may have multiple return statements
- Other statements are not allowed after a return statement

PARAMETERS AND VARIABLES

```
public static void main(String[] args) {
    int number = 1;
    System.out.println("The value of the variable 'number' in the main program: " + number);
    incrementByThree(number);
    System.out.println("The value of the variable 'number' in the main program: " + number);
}

public static void incrementByThree(int number) {
    number = number + 3;
}
```

- A parameter is a variable that exists only within the method
- When calling a method, the value of a given variable is copied into the parameter
- Afterwards both are independent!

SCOPE OF A VARIABLE

```
private static int incrementByThree(int number) {
    int result = number + 3;
    return result;
}
```

- You may also define variables in a method
- A variable defined in a method is only visible within that method
- The space in which a variable exists is called the *scope* of that variable

Situating Java

First Java program

Java building blocks

Exercises

Exercises

- @class
- @home

EXERCISES @CLASS

- exercise2
- exercise3
- exercise6
- exercise12
- exercise13
- exercise14



EXERCISES @HOME

- exercise4
- exercise5
- exercise7
- exercise8
- exercise9
- exercise10
- exercise11
- exercise15
- exercise17

