Java Crash Course

Lecturer: Nataliia Stulova
Teaching assistant: Mohammadreza Hazirprasad

Software Composition Group
University of Bern
16 September 2020
Part 1: Java ecosystem
Java is...

- a programming language
- an environment to run applications written in this language
WORA: Write Once Run Anywhere

Java bytecode:
- intermediate representation interpreted by the Java Virtual Machine (JVM)
- does not depend on exact hardware architecture (= run anywhere)
Java lingo

- **Java SE/EE/ME:** Java Standard/Enterprise/Micro platform
  - collections of tools to develop Java programs and the environment to run Java programs
- **JDK:** Java Development Kit
  - an implementation of one of the platforms (differ by sets of tools)
  - we will use some in this course: `java`, `javac`, `javadoc`, `jar`
- **JRE:** Java Runtime Environment
- **JVM:** Java Virtual Machine
Part 2: Java syntax
Java programming language

- **object-oriented**
  - (almost) everything is an object of a class
  - classes describe how the data is represented (via attributes) and manipulated (via methods)

- **imperative**
  - programmer specifies computational steps
Hello, World!

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

The `main()` method is the starting point for JVM to start execution of a Java program. Without the `main()` method, JVM will not execute the program.
Hello, World! (anatomy)

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```
Primitive and reference types

- Primitive data types (hold values)
  - byte < short < char < int < long < float < double
  - boolean
- All other types are reference types (hold references to objects)
- null - special reference, does not refer to anything, “empty” reference
Operators

For primitive types:

- Assignment: =
- Arithmetic: +, -, *, /, % (integer division)
- Comparison: >, >=, <, <=, == (equality), != (inequality)
- Conditional: && (AND), || (OR), ! (NOT)

Different in reference types:

- Reference equality: ==
- Contents equality: equals(...)

Operators can be overloaded - given new meaning - in reference types.

For example, + is used for concatenation in Strings:

```java
String s1 = "Hel" + "lo";
String s2 = "Hello";
System.out.println(s1.equals(s2));
```
Conditionals

Executing different code depending on some logical (=boolean-valued) conditions:

```java
if (CONDITION1) {
    ...
} else if (CONDITION2) {
    ...
} else {
    ...
}
```

Executing different code depending on fixed values of a variable:

```java
switch (VARIABLE) {
    case VALUE1:
        ...
        break;
    case VALUE2:
        ...
        break;
    default:
        ...
        break;
}
```
Loops

Java has 3 kinds of loops...

- **for**
  
  ```java
  for (int i = 0; i < N; i++) {
      code
  }
  ```

- **while**
  
  ```java
  while (condition) {
      code
  }
  ```

- **do** while
  
  ```java
  do {
      code
  } while (condition)
  ```

...and two special loop statements

- **continue** - start next loop iteration
- **break** - exit the loop

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

prints numbers 0-5

```
for (int i = 0; i < 5; i++) {
    if (i % 2 == 0) continue;
    if (i == 3) break;
    System.out.println(i);
}
```

prints number 1

skip even numbers

stops reaching 3
Exceptions

public void myMethod() throws IOException{
    ... 
    throw new IOException();
}

If a method does not handle an exception, the method
must declare it using the throws keyword at the end of a
method's signature.

public void myMethod(double foo) throws IOException{
    ... 
    myOtherMethod();
}

Handling unexpected behavior:

try {
    ...code that might throw an exception
}
catch (ExceptionType1 e1) {
    ...process exception
}

} catch (ExceptionType2 e2) {
    ...
}

} finally {
    ...code that always executes.
}

this method actually throws an exception
IO: Input and Output

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

Program input:
- CLI arguments (here)
- System.in

Program output:
- System.out (here)
- System.err
Stream IO

Reading from input stream:

The code on the right:
1. creates a Scanner object
2. uses it to read a String and an int
3. prints to the output stream, and
4. closes the Scanner object because there is no more input to read

Hint: always close the input stream!

```java
Scanner scanner = new Scanner(System.in);
String myString = scanner.next();
int myInt = scanner.nextInt();
System.out.println("myString is: " + myString);
System.out.println("myInt is: " + myInt);
scanner.close();
```
Reading a line from a text file

```java
File myFile = new File("PATH/test.txt");
Scanner myReader = new Scanner(myFile);

String textLine = myReader.nextLine();
System.out.println(textLine);
myReader.close();
```

Writing a line to a text file

```java
FileWriter myWriter = new FileWriter("PATH/test.txt");

myWriter.write("Hello, world!");
myWriter.close();
```

You might need to close the file stream explicitly in many cases.
public class HelloWorld {

/** This is a method-level doc comment. This is free-text comment part. */
* @param args This is tagged comment part
*
public static void main(String[] args) {

    // this is an inline comment
    System.out.println("Hello, World");

} /* this is a multi-line block comment */
Part 3: Java applications
Compiling and running

Java code is usually organized as a **project**.

Project file hierarchy:
- project (collection of packages)
  - package (collection of classes)
    - class

3 options to produce an executable program:
- **CLI**: text editor + java, javac, jar
- **IDE**: Eclipse, NetBeans, IntelliJ, VisualStudio Code,...
- Build systems: Maven, Gradle
No-IDE compilation

Make a folder with the following structure:

- your-program-name
  - Main.java
  - other *.java files
  - MANIFEST.MF

The manifest file should specify main class:

Main-Class: Hello

Option 1:

$ javac *.java
$ java Main

Option 2:

$ javac *.java
$ jar cfm main.jar MANIFEST.MF *.class
$ java -jar hello.jar
Coding conventions: why?

80% of the lifetime cost of a piece of software goes to maintenance.

Hardly any software is maintained for its whole life by the original author.

Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

- Code Conventions for the Java™ Programming Language
Coding conventions: which?

- most IDEs have support for project-level style set up
- styles can differ between projects, so agree with collaborators
public class MyUnit {
    public String concatenate(String one, String two) {
        return one + two;
    }
}

class MyUnitTest {
    @Test
    public void testConcatenate() {
        MyUnit myUnit = new MyUnit();
        String result = myUnit.concatenate("one", "two");
        assertEquals("onetwo", result);
    }
}
Part 4: practice
Exercises

- The simplest program: Hello, world!
- The unit test example program: string concatenation
- A printer program that: reads a number N, if it is even prints N characters ‘-’ to the standard output stream, if it is odd - prints N characters ‘=’ to the standard error stream.
  - Stream IO, conditionals, operators, loops, comments, [exceptions]
- A program that copies text files: reads a line from one file and writes it to another file
  - File IO, conditionals, operators, loops, comments, exceptions, unit tests
Further resources* on Java

* section added on 18.09.2020
# General Java tutorials

## Online crash courses

<table>
<thead>
<tr>
<th>Provider</th>
<th>Course Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Udemy</td>
<td>Java Beginners Program - A crash course</td>
</tr>
<tr>
<td>University of California, Berkeley</td>
<td>A Java Crash Course</td>
</tr>
</tbody>
</table>

## Tutorial sites

<table>
<thead>
<tr>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.javatpoint.com/java-tutorial">https://www.javatpoint.com/java-tutorial</a></td>
</tr>
<tr>
<td><a href="https://www.tutorialspoint.com/java/index.htm">https://www.tutorialspoint.com/java/index.htm</a></td>
</tr>
<tr>
<td><a href="https://www.w3schools.com/java/default.asp">https://www.w3schools.com/java/default.asp</a></td>
</tr>
<tr>
<td><a href="https://howtodoinjava.com/">https://howtodoinjava.com/</a></td>
</tr>
</tbody>
</table>
Thematic resources

- **Code examples** collection of basic Java concepts
- **StackOverflow** - programming community Question/Answer website
- **Maven** in 5 minutes
- Unit testing with JUnit:
  - [https://www.tutorialspoint.com/junit/junit_test_framework.htm](https://www.tutorialspoint.com/junit/junit_test_framework.htm)
  - [https://www.vogella.com/tutorials/JUnit/article.html](https://www.vogella.com/tutorials/JUnit/article.html)
- **Codility** - a website with programming challenges