01: Introduction

Network Oriented Software

Stefan Huber Harry Schmuck Maximilian Tschuchnig Eduard Hirsch
ITS, FH Salzburg

Summer 2021

Literature on Java

- Horstmann, Core Java Volume I and II. [HC18; Hor18]
  Mostly good quality and easy to read. Very verbose with background information. Also useful for beginners and often compares Java to C++.
- Herbert Schildt, Java complete reference [Sch18].
  Well written, more compact than Core Java. A good book to lookup stuff.
- And there is more: [EF18; Blo17; Har13]
- There are a couple of Java books in our library; mostly older[2] editions at the moment.

[2] You do not require the latest editions of literature. Presumably the course will entirely rely on Java 9. Debian Buster ships Java 11.

More resources

Trust the specification! Question random code snippets on the web!

- Use the Java API Specification: `https://docs.oracle.com/en/java/javase/15/docs/api/index.html`.
- There is also The Java Language Specification: `https://docs.oracle.com/javase/specs/jls/se15/html/index.html`.

Web resources:

- `https://en.wikipedia.org/wiki/Java_(programming_language)`
- `https://www.w3schools.com/java/default.asp`

Section 2

Basics of the Java language and ecosystem

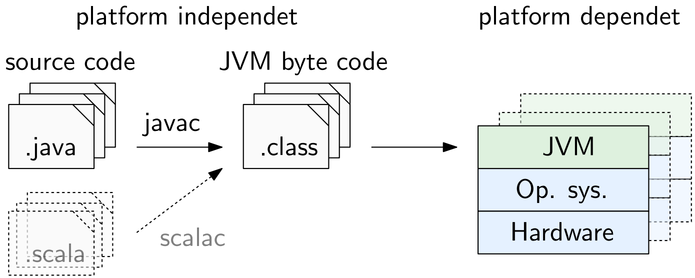
Java is platform independent

Java programs are interpreted by the Java Virtual Machine (JVM):

- Java originally targeted at embedded devices, like TV switch boxes. A variety of different platforms had to be supported.
- Java runs wherever you port the JVM to. Java is portable.

```
# Compile the Java code in HelloWorld.java to Java bytecode in HelloWorld.class
% javac HelloWorld.java
# Run HelloWorld.class by invoking the JVM
% java HelloWorld
Hello World
```

Java is platform independent



- The JVM is an abstract computing machine. It hides the details of the processor, like endianess, size of registers, details of the floating-point unit and so on. It is architecture neutral.
- The JVM comes at additional runtime costs. Just-In-Time (JIT) compilers reduce those costs significantly by translating the hotspots (often executed code snippets) on-the-fly and in-memory into native machine code.

Java is object oriented

- Java is purely object oriented.
- In Java, all functions are class members and they are called methods.
- Even `main()` is a method, as in C#, but unlike in C and C++.

```java
/**
 * The hello world class.
 */
public class HelloWorld {
    /** This is the program's entry point. */
    public static void main(String args[]) {
        // Purely object oriented: The System class contains an object out,
        // which provides a method println() to print-a-line.
        System.out.println("Hello world");
    }
}
```

A first step into the Java language

- Syntax close to the C-family, like C++, Objective-C or C#:
  Curly braces {} for blocks, whitespace is ignored, single-line comments // and multi-line comments /**/, double quotes delimit strings.

```java
public class HelloWorld {
    /** This is the program's entry point. */
    public static void main(String args[]) {
        // Purely object oriented: The System class contains an object out,
        // which provides a method println() to print-a-line.
        System.out.println("Hello world");
    }
}
```

- The class `HelloWorld` must reside in the file `HelloWorld.java`.
  - Each class has its own file. Case sensitive names, also for the filenames.
  - When the JRE looks for the bytecode of the class `A` then it looks for the class file `A.class`.
- If a class contains a `public static void main()` method then the class can be executed as a program by the Java interpreter.

Java is an environment

- Java comes with a whole environment, the Java Runtime Environment (JRE).
- The JRE contains the JVM but also the Java standard library with thousands[3] of classes.
- For development you need a Java Development Kit (JDK).
  - There are different editions, we use the Standard Edition (SE).
  - Most current version is Java SE 15. Debian Buster ships version 11.

```
# Debian Buster ships a Java 11 JRE (Java Runtime Environment):
% java -version
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode, sharing)
```

[3] Java 12 comes with 4433 classes, Java 5.0 with 3279 classes, Java 1.3.1 with 1840 classes.

Java ships a documentation generator

- Javadoc comments start with */*** and allow to generate source code documentation.
  - These are used to comment files, classes and its members and similar entities.
- Javadoc takes those comments and generates an HTML documentation of those entities.

```java
/** The obligatory hello world.
 *
 * This is a hello world demo that not only demonstrates the main() method,
 * println() and strings, but also javadoc comments.
 *
 * @author Stefan Huber <stefan.huber@fh-salzburg.ac.at>
 */

/**
 * The hello world class.
 */
public class HelloWorld {
```

Data types

Java is strongly typed:

- Every variable has a declared type.
- If a value is assigned to a variable, or an argument is passed to a function, the types are checked.

There are two categories of types:

- Primitive types
- Non-primitive types, e.g., class types, arrays, enums et cetera.

Primitive types

```java
/** The Java language knows exactly eight primitive types. */
class PrimitiveTypeDemo {
    public static void main(String args[]) {
        byte  a = -0x0a;       // A signed 8-bit integer. Hex literals as in C.
        short b = 0b1010;      // A signed 16-bit integer. Binary literals since Java 7.
        int   c = 1_000_000;   // A signed 32-bit integer. Underscores since Java 7.
        long  d = 42L;         // A signed 64-bit integer. Long literals by suffix L.

        float  e = 3.1416f;    // IEEE 754 single-precision floating-point number.
        double f = 1.256e-6;   // IEEE 754 double-precision floating-point number.

        char g     = '€';      // A character. (Actually, a UTF-16 code unit.)
        boolean h = true;      // There is only true or false, and no third!

        System.out.printf("%d %d %d %d %f %f %c %b\n", a, b, c, d, e, f, g, h);
    }
}
```

- `0x12`, `1.41f`, `3.14`, `'A'`, `false`, `"Hi"` and so on are called literals, and they have a type.
- Note that the C and C++ standards do not define the sizes of integer data types! This is why C99 introduced platform-independent types like `int32_t`.

Operators

The following operators work (almost) as in C:

- Arithmetic: `+ - * / %` and its assignment counterparts `+= -= *= /= %=`
- Incremental: `x++ ++x x-- --x`
- Relational: `== != <= >= < >` — Produce a `bool`
- Logical: `! && ||` — Take and produce `bool`
- Bitwise: `& | ^ ~ << >>` and its assignment counterparts — Java knows a zero-padding `>>>`

```java
class OperatorsDemo {
    public static void main(String args[]) {
        assert 13 + 5 == 18;                // Call java with -ea to enable assertions
        assert 5 / 13 == 0;                 // Integer division
        assert 1.0 / 2.0 == 0.5;            // Be careful: Floating-point equality!
        assert 1.0f + 0.00000001f == 1.0f;  // Numerical errors break exact equality

        //assert 3 >= 2 && 7;               // Syntax error: 7 is not boolean
        //assert 3 == true;                 // Syntax error: cannot compare int and bool
        assert false;                       // We expect to fail
    }
}
```

Operator precedence

If no parenthesis are used, like `(x && y) == z`, then the operator precedence defines which operators bind tighter. From highest to lowest:

Access and call	`[] . ()`	left to right
Unary	`! ~ ++ -- + - () (cast) new`	right to left
Binary	`* / \%`	left to right
	`+ -`	left to right
	`<< >> >>>`	left to right
	`< <= > >= instanceof`	left to right
	`== !=`	left to right
	`&`	left to right
	`^`	left to right
	`|`	left to right
	`&&`	left to right
	`||`	left to right
Ternary	`?:`	right to left
Assignment	`= += -= *= /= &= |= ^= <<= >>= >>>=`	right to left

Strings

- A string is a sequence of Unicode characters.
- A string is actually an instance of the class `String`.
  - This class provides more than 50 methods.
  - We expect that you consult the API specification when required.
- Unlike C++, Java has no operator overloading.
  - However, the Java language explicitly defines + on strings as concatenation.
- Strings are immutable; they cannot be changed.

```java
class StringDemo {
    public static void main(String args[]) {
        String str = "Alan Turing";
        System.out.println(str.length());               // 11
        System.out.println(str.substring(2, 7));    // "an Tu"
        System.out.println(str + " rocks!");           // "Alan Turing rocks!"
    }
}
```

Equal versus identical

- The operator == on objects tests whether they are the same instances.
- This is different from testing whether they are equivalent w.r.t. their data or state.

```java
class StringEqualityDemo {
    public static void main(String args[]) {
        String str1 = "Hello";
        String str2 = str1 + "";

        System.out.println(str1 != str2);          // Not identical, different instances
        System.out.println(str1.equals(str2));     // But two equal instances
    }
}
```

Empty versus null

- When a variable is `null` then it refers to no object.
- Of course, an empty string is not the same as a null object.

```java
class EmptyNullDemo {
    public static void main(String args[]) {
        String notmuch = "";
        String notatall = null;

        System.out.println(notmuch.length());     // 0
        System.out.println(notatall == null);     // true
        //System.out.println(notatall.length()); // Runtime error: NullPointerException

        // Lazy evaluation: As notatall != null is already false the second
        // operand of && is not evaluated. Similar with ||.
        System.out.println(notatall != null && notatall.length() > 0);   // false
        System.out.println(notatall == null || notatall.length() == 0);  // true
    }
}
```

Variable scopes and control flow

- Control flow structures are essentially those of the C-family.
- The scope (life range) of a variable is the surrounding block.

```
        System.out.println("Fibonacci series …");
        int a = 1, b = 0;
        while (b < 100) {                       // Each block {} defines a scope for variables
            System.out.print(b);
            System.out.print(" ");

            int c = a + b;                      // A variable lives only within is scope
            b = a;
            a = c;
        }                                       // c does not live outside this block
        System.out.println();

        int n = 1;
        int fac = 5;
        for (int i = 1; i <= fac; ++i)
            n *= i;
        System.out.printf("%d factorial is %d\n", fac, n);
```

Control flow

```java
// Execute as long as condition is true
while (bool_cond) {
    // bool_cond must be of type bool:
    // if (7) {} is illegal in Java!
    if (bool_cond) {
        continue;    // Goto start of inner loop
    } else {
        break;       // Exit inner loop
    }
}

for (init_expr; bool_cond; post_expr) {
    // Equivalent to {
    //     init_expr;
    //     while (bool_cond) {
    //         { block }
    //         post_expr;
    //     }
    // }  The scope of init_expr is the loop
}
```

```java
do {    // Do not repeat when
        // condtion becomes false.
} while (bool_cond);

switch (choice) {
    // byte, char, int, …
    case constant_integer:
        break;
    case enum_constant:
        break;
    // Since Java 7
    case "constant_string":
        break;
    default:
        break;
}

// There is actually a "tame goto":
// A break to a labled block.
```

Arrays

Arrays in Java are similar to dynamic C++ arrays.

- The class `java.util.Arrays` contains many helper methods like string conversion and comparison.

```java
import java.util.*;                                 // We need java.util.Arrays

class ArraysDemo {
    public static void main(String args[]) {
        int[] a = new int[6];                       // Uninitialized
        for (int i=0; i < a.length; ++i)            // a.length gives the array size
            a[i] = i * i;
        System.out.println(Arrays.toString(a));

        int[] b = {0, 1, 4, 9, 16, 25};
        System.out.println(Arrays.equals(a, b));

        int sum = 0;
        for (int elem : b)                          // Java 5 has a foreach loop, as C++11
            sum += elem;
        System.out.println(sum);
    }
}
```

Java is robust and secure

Java has been designed to be robust and secure:

- Java is memory-managed so there are no use-after-free or double-free memory errors because a garbage collector frees memory.
- Java has no concept of a pointer so there is no invalid pointer dereference error.
- Array access is checked at runtime to eliminate out-of-bound access. This eliminates buffer overflow attacks.
- Java comes with security policies.