Creating Java Programs and Using Data

Pieter Joubert

September 27, 2023

Introduction to Object Oriented Programming

Table of Contents

Notes

Remark

Please note that the lectures will not cover ALL the work in the textbook but just the most important. It is the responsibility of the student to make sure they read up on the rest of the content.

Section 1

Creating Java Programs

Java Terminology

- We program in Java (a *High-Level Langauge*) by writing Java *Source Files*.
- The programs we write consist of *Statments* that need to conform to Java *Syntax*.
- The Java *Compiler* converts the *Source* code into *Byte* code which is executed by the *Java Virtual Machine (JVM)*.
- Executing code on the provides us with a number of advantages in terms of *WORA*, *Security* and being *Architecturally Neutral*.
- Our code needs to be free of *Syntax Errors* or else it won't compile.
- Even if your code is free of *Syntax Errors*, it might not run correctly if it contains *Logic Errors*.

Object Oriented Programming

- Java is an *Object Oriented* Programming Language.
- *Object Orientation* is the process of converting the Problem Space we want to solve into Classes and Objects in the programming language we are using.
- This makes it easier for us to link our code to the "Real World".
- This is especially useful for large projects with many developers working on it.

Classes and Objects

- *Classes* describe a "blueprint" or a definition of the various *Objects* we have in our Problem Space.
- Each of these Classes have Attributes (information about the Class) and Methods (Actions the class can perform)
- We can then instantiate concrete versions of these Classes called *Objects* that have thier own unique Attributes and Actions.

Classes and Objects - Example

- If we wanted to create program to keep track of Pets we might have a *Cat* class, with attributes suchs as *Name*, *Age*, *Breed*.
- We would then instantiate the *Objects* with the information of actual cats, e.g.
    - Cat1 → Salem, 4, Domestic Short Hair
    - Cat2 → Ginger, 2, Persian
    - Cat3 → Whiskers, 6, Maine Coon
- You will go into a lot more detail regarding OOP in Week 3, but we need to introduce it so that you can understand the basic Structure of a Java Program.

Structure of a basic program

```
public class Main {

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

    }

}
```

This piece of source code would need to be saved in a filed named Main.java to compile correctly.

Output

- There are a number of different ways a Java program can produce output, i.e. to the console, to a GUI applcation, writing to a file.
- In this module we will be looking at writing to the console and writing to files. Interacting with a GUI application using Java will be discussed in more detail in the Full Stack Application Development module.
- Technically we can also obtain output from a Java program by running Tests, though this kind of output is not what the user of the program would typically see.

Saving, compiling and executing a program

- We will be using IntelliJ which performs saving, compilation and executing for us via the IDE's interface.
- We can also compile and execute our code through the command line. This is in fact what IntelliJ is doing for us in the background.
- The commands below assume we have a file named *Main.java* in the current directory.

```
javac Main.java

java Main
```

Adding Comments to a program

- We can add comments, i.e. text that will not be executed by the compiler, to our code.
- We can use this to inform ourselves, and other programmers viewing the code, about various aspects of the code we have written.
- We can either create single line comments using //.
- Or we can create multi-line comments using /∗ to start the comment and ∗/ to end the comment.

Adding Comments to a program - Example

```java
/*
 * Author: Pieter Joubert
 * Date: 25 September 2022
 * Program: Hello World
 */
public class Main {

    //Our main method
    public static void main(String args[]) {
        // System.out.println("Good Bye World");
        System.out.println("Hello World");
    }
} // End of your Main class
```

Using comments

- While comments can be useful we should always try and rather make sure our code is readable without the need for additional comments if we can.
- Compare the two examples below:

```java
// this variable stores the number of students
int x = 100;

int numStudents = 100;
```

Section 2

Using Data - Part I

Using Data

- So far we have just used hardcoded values within our programs.
- Programming really only begins once we start manipulating data.
- We use *variables* to keep track of the data we want to store. A variable (generally) refers to a specific *memory* location, storing data of a specific *type*, with a *name* we can use to access it later and containg some kind of *value*.

Declaring and using constants and variables

```java
final String HEADER = "==================";
int numStudents = 100;
String moduleName = "OOP";

System.out.println(moduleName);
System.out.println(HEADER);
System.out.println("Number of students " +
    numStudents);

numStudents = 50;
moduleName = "SWW1";

System.out.println(moduleName);
System.out.println(HEADER);
System.out.println("Number of students " +
    numStudents);
```

Identifier naming rules and conventions

- We need to follow a number of rules to ensure that variables (and other identifiers) have valid names.
- Identifiers must start with a letter, an underscore or a dollar sign.
- Identifiers can only contain Letters, Digits, Underscores and Dollar Signs.
- We tend to use *Camel Case* for naming. This follows the convention that each word in a variable starts with an uppercase letter. We usually start the whole variable with a lower case unless it is a Class name.

Scope

- We when declare a variable we can only use it in certain areas of our program, known as *Scope*.
- If you try to access a variable that is not in scope you will get a syntax error. Essentially Java will let you know that it that you haven't declared the variable yet.

```java
public class Main {

    public static void main(String args[]) {
        String name = "Pieter Joubert";
        System.out.println("Hello " + name);
    }

    public void printName() {
        System.out.println("Hello " + name);
    }
}
```

Primitive Data Types

- There are a number of *primitive* data types we can use in Java.
- By primitive we mean that the data type is not a Class.
- Some examples of primitive data types are:
  - byte: usually the smallest single value you can store.
  - short, int, long: various lengths of numbers without a decimal value (both positive and negative)
  - float, double: numbers that can have a demical value stored in different ways
  - boolean: a value that can be either true or false
  - char: a single character value
- note that *String* is not a primitive type, as *String* is a built-in class in Java with additional funcationality added

Section 3

Arithmetic

Operations

- If we have data stored in our programming we can now perform various operations on this data, the most common type of operation being Arithmetic operations.
- We can perform all standard mathematical operators using a the appropriate *operator*.

OPERATOR	**DESCRIPTION**	**EXAMPLE**
$+$	Addition	$45 + 2$; the result is 47
$-$	Subtraction	$45 - 2$; the result is 43
$*$	Multiplication	$45 * 2$; the result is 90
$/$	Division	$45.0/2$; the result is 22.5
		$45/2$; the result is 22 not (22.5)
%	Modulus	Characters, Strings & Arrays

Associativity and Precedence

- Operators with the same precedence will be evaluated from left to right.
- The relative precedence of the operators is shown below.

OPERATORS	**DESCRIPTIONS**	**RELATIVE PRECEDENCE**
$*/\%$	Multiplication, Division, Modulus	Higher
$+-$	Additional, Subtraction	Lower

Section 4

Type Conversion

Type conversions

- Java performs *implicit* type conversion if you are performing an operation with different types. E.g. adding a float and an int.
- This is done by converting an answer to the unifying typpe, i.e. a type that can hold the answer to the evaluation without losing precision.
- You can also explicitly convert between types using the *cast* operator as shown below:

```java
float position = 10.45034;
int tilePosition = (int) position;
```

Section 5

Lecture summary

Lecture summary

- Creating Java Programs
- Using Data
- Arithmetic
- Type Conversions

Questions

Thank you! Questions?