Inf1-OP
Course Overview

Volker Seeker, adapting earlier version by Perdita Stevens and Ewan Klein

School of Informatics

February 26, 2018

Who to contact for help?

**Lecturer:** Volker Seeker (office hours)

**TA:** Naums Mogers

**Course Page:** [http://www.inf.ed.ac.uk/teaching/courses/inf1/op/](http://www.inf.ed.ac.uk/teaching/courses/inf1/op/)

**Piazza:** see sign-up link on web page

**Tutors and lab demonstrators:** see web page

**ITO:** office at Appleton Tower level 6; source of all admin knowledge

Additional help?

**Fellow Students:** Feel free to work in groups

**InfBase:** Drop in helpdesk (Link)

**InfPals:** student-to-student support (Link)

**Programming Club:** For more programming practice (Link)

**Societies:** CompSoc or Hoppers

**Better Informatics:** [https://betterinformatics.com/](https://betterinformatics.com/)
**Textbooks**

[Image of textbooks]

**Recommended textbook**


- Contains a lot more than you need for this course.
- Available from library as ebook, see course web page.

Recommended if you expect to go beyond the basic syllabus: advanced lab material may assume you have it.

**Online Resources**

To get you started:

- Oracle Java tutorials
- Java Language Spec
- API Spec
- Tutorials Point
- Lynda
- stackoverflow

- but there are many many sources: feel free to browse and find things that suit your own style.

**Weekly course events**

1. **One** lecture per week
2. **One** 2-hour scheduled lab session per week
3. **One** tutorial per week, from Week 2
4. Some bonus video lectures

- Lectures will be videoed – access via Learn
- Anyone is welcome to audio record for revision purposes
- Lecture slides are all on web page now. They are subject to changes before delivery.
Lectures

Target audience for these lectures:
▶ You have taken Informatics 1 Functional Programming.¹
▶ You have not already learned to program in an imperative, object-oriented language.²

¹I’ll allude to this from time to time, but don’t worry if you haven’t, just ignore my allusions.
²If you have, I will understand if you skip lectures. But use this semester to get additional programming practice!

Lab Exercises

▶ Weekly lab exercises—similar to weekly exercises in Haskell course.
▶ Can be carried out in the labs; you will be assigned to a two-hour lab session per week, starting from this week.
▶ If necessary, go to more than one session; can also swap.
▶ You may access the lab exercises from anywhere: physically attending the lab sessions is optional after the first week.
▶ Exercises are divided into warm-up, core and optional; the core exercises are obligatory.
▶ Feedback on lab exercises:
   ▶ principally via automated tests which you must be able to use for the exam
   ▶ plus help from demonstrators in scheduled labs
   ▶ plus discussions with your tutor (please send them your solutions).

Scheduled Labs

▶ You will be allocated to one of these – see course web page for which and when.
▶ In 5.05 or 6.06, Appleton Tower.
▶ Allocation is just to manage space: feel free to turn up to a slot you’re not allocated for, but...
▶ ... if you have a clash that means you can’t attend your allocated slot ever, use the ITO student portal, to get it changed; If that does not work, use the ITO web form
▶ ... it’s possible there might be contention for seats: if so, those not allocated to the lab must leave.

Tutorials

▶ starting in week 2
▶ Mainly a chance for you to ask questions and get answers about course material and lab exercises
▶ A few designed exercises to make sure you have the essential development skills
▶ Last few tutorials to practice old exams questions
▶ Your tutor is your best source of feedback on your progress
▶ To get the most out of your tutorials, prepare questions or issues you want your tutor to address and send it to them up front
▶ You must attend a tutorial every week.
▶ If you’re allocated to a time you can’t make, contact the ITO (via student portal or web form).
I already have programming experience

Great - Keep Practising!

- additional material on course page
- advanced lab exercises
- programming club
- make use of tutorials to discuss advanced content

Assessment

Mostly formative – labs and tutorials to help you learn and give you feedback on how you’re doing. The only summative assessment is the final programming exam – this determines your mark. It is:

- Scheduled as part of normal exam diet.
- Done in the labs on DICE machines in “exam mode”
- 2 hours long (not 3 as before). We do not aim to put you under time pressure.
- “Open book”: some online documentation provided, and you may take in anything you like on paper (books, lecture notes, solutions to lab exercises...) or on a USB stick.
- All code you submit must compile and pass some basic JUnit tests you’re given, or it will get 0. Make sure you know how to use JUnit!
- A mock programming exam is scheduled for Week 11. Be there!
- There is an experimental automarking service: see web page.

Motivation

Why learn another programming language after Haskell?

- Haskell is not the only language you will ever need!
- Object orientation is the dominant paradigm, so you need to know an OO language.
- More importantly you need to get good at learning languages.
Why Java?

- Decently designed OO language
- Strong static typing
- Very popular – > 9 million developers worldwide – so:
  - huge ecology of libraries, frameworks, tools...
  - useful for you to have on your CV!

Why Java?

▶ Decently designed OO language
▶ Strong static typing
▶ Very popular – > 9 million developers worldwide – so:
  ▶ huge ecology of libraries, frameworks, tools...
  ▶ useful for you to have on your CV!

How to approach this course

▶ The lectures, Fundamental labs and exam cover only a small subset of Java.
▶ To pass the course, you will need to know that subset well.
▶ But further: if you plan to do more Informatics courses, that is not enough – for example, if that’s all you know, you will have to learn more Java as you go on in second year. When you will be busier than you are now...
▶ The better you learn Java this semester, the better off you will be.
▶ So if you have a head start: lucky you, but still work!
▶ Conversely if you don’t, take full advantage of the support offered this semester: it’ll never happen again.

What is object orientation?

It means: your program is structured like the domain (real world). Objects (organised into classes of similar objects) typically represent things (organised into types of similar things).

Objects have

▶ state: they can store data
▶ behaviour: they can do things, in response to messages
▶ identity: two objects with the same state can still be different objects.

Any of state, behaviour, identity can be trivial for a particular object, though. Our first objects will be just little bits of wrapped up behaviour.

Lets Get Started Already
**A First Example**

**HelloWorld.java**

```java
/*************************
* Prints "Hello, World!"
*************************/

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

**Creating a New Class**

1. All Java code sits inside a class.
2. By important convention, class names are capitalized and in 'CamelCase'.
3. Each class goes into a file of its own (usually; and always in this course).
4. So, use a text editor (e.g., gedit) to create a file called `<HelloWorld>.java`.
5. The name of the file has to be the same as the name of the class, and suffixed with `.java`.

**At the terminal**

```bash
gedit HelloWorld.java
```

**A First Example**

**Declare a class**

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

- Basic form of a class definition.
- Class definition enclosed by curly braces.

**Declare the main() method**

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

- We need a `main()` method to actually get our program started.
- All our other code is invoked from inside `main()`.
- `void` means the method doesn’t return a value.
- The argument of the method is an array of `Strings`; this array is called `args`.
- Definition of a method enclosed by curly braces.
A First Example

Print a string to standard output

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

▶ `System.out` is an `object` (a rather special one).
▶ `println("Hello World!")` is a `message` being sent to that object: `println` is the `method name`, "Hello World!" is the `argument`.
▶ The whole line is a `statement`: `must` be terminated with a semi-colon (`;`).
▶ Strings `must` be demarcated by `double quotes`.
▶ Strings cannot be broken across a line in the file.

Running the Program

▶ Now that we have compiled code, we can run it.
▶ Use the `java` command in a terminal.

```java
java HelloWorld
Hello World!
```

▶ Note that we omit the `.class` suffix in the run command. The `java` command wants a `classname` as argument, not a filename.

Compiling

▶ The program needs to be compiled before it can be executed.
▶ Use the `javac` command in a terminal.

```java
javadoc HelloWorld.java
```

▶ If there's a problem, the compiler will complain.
▶ If not, compiler creates a Java bytecode file called `HelloWorld.class`.

Edit-Compile-Run Cycle

Type in the program using an editor and save the program to a file. Use the name of the main class and the suffix `.java` for the file. This is called a `source file`.

The process of compiling a source file generates the bytecode file. The bytecode will have a `.class` suffix; the prefix will be the same.

A java interpreter will read the bytecode file and execute the instructions in it. If an error occurs while running, the interpreter will stop its execution.
Edit-Compile-Run Cycle

- The program needs to be compiled before it can be executed.
- If you edit a program, you need to compile it again before running the new version.
- Eclipse will compile your code automatically.

Basic Functionality

Arithmetic

Addition and Division

```java
public class Calc {
    public static void main(String[] args) {
        System.out.print("The sum of 6 and 2 is ");
        System.out.println(6 + 2);
        System.out.print("The quotient of 6 and 2 is ");
        System.out.println(6 / 2);
    }
}
```

Output

The sum of 6 and 2 is 8
The quotient of 6 and 2 is 3

String Concatenation, 1

```java
public class Concat {
    public static void main(String[] args) {
        System.out.println("The name is " + "Bond, ");
        System.out.println("James Bond");
    }
}
```

Output

The name is Bond, James Bond
String Concatenation

```java
public class Concat {
    public static void main(String[] args) {
        System.out.println("Is that you, 00" + 7 + "?");
    }
}
```

Output

Is that you, 007?

Assignment: Basic Definitions

Variable: A name that refers to a value
Assignment Statement: Associates a value with a variable

```
int a, b;
a = 1234;  // literal
b = 99;    // literal
```

Important: = is the operator in an imperative statement, not a logical assertion.

Assignment: Combining Declaration and Initialisation

Variables that have been declared, but not assigned to, are a potential source of error. (Exercise for the keen: understand what happens to them in Java.) It’s often best to declare a variable and initialise it at the same time.

```
int a, b;
a = 1234;
b = 99;
int c = a + b;  // combined declaration and assignment statement
```

Hello World with Added Variables

Storing a String in a variable

```java
public class HelloWorld {
    public static void main ( String [] args ) {
        String msg = "Hello World!";
        System.out.println( msg );
    }
}
```
## Built-in Data Types

<table>
<thead>
<tr>
<th>type</th>
<th>value set</th>
<th>literal values</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>characters</td>
<td>'A', '$'</td>
<td>compare</td>
</tr>
<tr>
<td>String</td>
<td>sequences of characters</td>
<td>&quot;Hello World!&quot;, &quot;Java is fun&quot;</td>
<td>concatenate</td>
</tr>
<tr>
<td>int</td>
<td>integers</td>
<td>17, 1234</td>
<td>add, subtract, multiply, divide</td>
</tr>
<tr>
<td>double</td>
<td>floating-point numbers</td>
<td>3.1415, 6.022e23</td>
<td>add, subtract, multiply, divide</td>
</tr>
<tr>
<td>boolean</td>
<td>truth values</td>
<td>true, false</td>
<td>and, or, not</td>
</tr>
</tbody>
</table>

### Integer operations

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 + 3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5 - 3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5 * 3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5 / 2</td>
<td>2</td>
<td>no fractional part</td>
</tr>
<tr>
<td>5 % 2</td>
<td>1</td>
<td>remainder</td>
</tr>
<tr>
<td>1 / 0</td>
<td></td>
<td>run-time error</td>
</tr>
<tr>
<td>3 * 5 - 2</td>
<td>13</td>
<td>* has precedence</td>
</tr>
<tr>
<td>3 + 5 / 2</td>
<td>5</td>
<td>/ has precedence</td>
</tr>
<tr>
<td>3 - 5 - 2</td>
<td>-4</td>
<td>left associative</td>
</tr>
<tr>
<td>(3 - 5) - 2</td>
<td>-4</td>
<td>better style</td>
</tr>
<tr>
<td>3 - (5 - 2)</td>
<td>0</td>
<td>unambiguous</td>
</tr>
</tbody>
</table>

### Floating-Point Numbers

The default floating-point type in Java is **double**.

### Floating-Point Operations

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.141 + .03</td>
<td>3.171</td>
</tr>
<tr>
<td>3.141 - .03</td>
<td>3.111</td>
</tr>
<tr>
<td>6.02e23 / 2.0</td>
<td>3.01e23</td>
</tr>
<tr>
<td>5.0 / 3.0</td>
<td>1.6666666666666667</td>
</tr>
<tr>
<td>10.0 % 3.141</td>
<td>0.577</td>
</tr>
<tr>
<td>1.0 / 0.0</td>
<td>Infinity</td>
</tr>
<tr>
<td>Math.sqrt(2.0)</td>
<td>1.4142135623730951</td>
</tr>
<tr>
<td>Math.sqrt(-1.0)</td>
<td>NaN</td>
</tr>
</tbody>
</table>
Type Conversion

Sometimes we can convert one type to another.

- **Automatic:** OK if no loss of precision, or converts to string
- **Explicit:** use a cast or method like `parseInt()`

<table>
<thead>
<tr>
<th>expression</th>
<th>result type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1234&quot; + 99</td>
<td>String</td>
<td>&quot;123499&quot;</td>
</tr>
<tr>
<td><code>Integer.parseInt(&quot;123&quot;)</code></td>
<td>int</td>
<td>123</td>
</tr>
<tr>
<td><code>(int) 2.71828</code></td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td><code>Math.round(2.71828)</code></td>
<td>long</td>
<td>3</td>
</tr>
<tr>
<td><code>(int) Math.round(2.71828)</code></td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td><code>(int) Math.round(3.14159)</code></td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>11 * 0.3</td>
<td>double</td>
<td>3.3</td>
</tr>
<tr>
<td><code>(int) 11 * 0.3</code></td>
<td>double</td>
<td>3.3</td>
</tr>
<tr>
<td><code>(int) (11 * 0.3)</code></td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td><code>(int) (11 * 0.3)</code></td>
<td>int</td>
<td>3</td>
</tr>
</tbody>
</table>

Type Conversion: Division

<table>
<thead>
<tr>
<th>expression</th>
<th>result type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 / 2</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td><code>(double)(5 / 2)</code></td>
<td>double</td>
<td>2.0</td>
</tr>
<tr>
<td>5 / 2.0</td>
<td>double</td>
<td>2.5</td>
</tr>
<tr>
<td>5.0 / 2</td>
<td>double</td>
<td>2.5</td>
</tr>
<tr>
<td>5.0 / 2.0</td>
<td>double</td>
<td>2.5</td>
</tr>
</tbody>
</table>

*Moral:* if you want a floating-point result from division, make at least one of the operands a `double`.

Command-line Arguments

Unix commands

`mkdir` MyJavaCode

`mkdir` is a command and MyJavaCode is an argument

Using Java to carry out commands

```java
public class Add {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        System.out.println(a + b);
    }
}
```

3 and 6 are command-line arguments for the program Add

```
% java Add 3 6
9
```

int a = Integer.parseInt(args[0]);

- This reads in a string (e.g., "3") from the command line,
- parses it as an int, and
- assigns this as the value of variable a.
Command-line Arguments

Missing an argument

% java Add 3
java.lang.ArrayIndexOutOfBoundsException: 1

This a run-time error — we didn’t provide anything as a value for args[1]:

int b = Integer.parseInt(args[1]);

Recap: Learning Outcomes for this week

▶ Use a text editor to create and modify simple Java programs which print strings to a terminal window.
▶ Use the command-line to compile and run Java programs.
▶ Declare int, double and String variables and assign values to them.
▶ Use Java’s main() method to consume command-line arguments.
▶ Parse strings into values of type int and double.
▶ Carry out simple operations on int, double and String data values.
▶ Compute fractional results from division with integer values, using casting if necessary.

Summary

This Week's Reading

Java Tutorial
pp1-68, i.e. Chapters 1 Getting Started, 2 Object-Oriented Programming Concepts, and Chapter 3 Language Basics, up to Expressions, Statements and Blocks
– except note:
  ▶ We use Eclipse, not NetBeans, as our IDE.
  ▶ We’ll come to the Chapter 2 material later.
  ▶ We’ll talk about Arrays later.
I suggest skimming Ch 2 and the Arrays section, and rereading them later.