

# MSc/ICY Software Workshop

## Introduction – Simple Computation, Variables, Types, static Methods

Manfred Kerber [www.cs.bham.ac.uk/~mmk](http://www.cs.bham.ac.uk/~mmk)

Tuesday 1 October 2019

12 ~ 15 h / wk

# Overview

1. Pocket calculator computations, base types, simple strings, variables, static methods, JavaDoc  
Wed/Thu/Fri: 1st Lab Lecture (login, editor, javac, javadoc)
2. Classes, objects, methods, JUnit tests  
Wed/Thu/Fri: 2nd Lab Lecture (Eclipse)
3. Conditionals, 'for' Loops, arrays, ArrayList
4. Exceptions, I/O (Input/Output)
5. Functions, interfaces
6. Sub-classes, inheritance, abstract classes
7. Inheritance (Cont'd), packages
8. Revision
9. Graphics
10. Graphical User Interfaces
11. Graphical User Interfaces (Cont'd)

Changes possible

Java is

- platform independent
- object-oriented
- strictly typed

We will learn in more depth what this means in the course of the next few weeks.

# A simple program

In a few simple steps to a HelloWorld program:

1. Open a file `HelloWorld.java` in an editor like jedit or gedit  
*camel*
2. Write the program starting with  
`public class HelloWorld`  
Note that the name of the class here must match the filename.
3. Save the file.
4. Compile the program on the command line by  
`javac HelloWorld.java`  
This generates a file `HelloWorld.class`.
5. Run the program on the command line by  
`java HelloWorld`
6. Extract documentation on the command line by  
`javadoc HelloWorld.java`


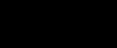
# The full HelloWorld program

```
/** → javadoc  
 * Java is an object-oriented programming language.  
 * Everything is put in classes. The file name  
 * HelloWorld.java has to match the name of the class.  
 *      public class HelloWorld  
 * We ignore for now the meaning of "public class" and  
 * the line "public static void main(String[] args)".  
 *  
 * System.out.println(...); prints the argument  
 * to standard output (the terminal here).  
 * Objects such as "Hello World!" are called strings.  
 */
```

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

*/print* *→ start a new line*

# Comments

- Java is to be understood by **computers and humans**.
- **Computers** translate programs into a level (by an interpreter or by a compiler) that can be executed command by command in the order given by the programmer.
- **Humans** translate programs into something that is meaningful to them.
- If somebody reads somebody else's code (or even their own code after a while) then they may have problems to understand it. What is done in the program? Why is it done? How is it done? When will it work and when not? Does it come with a Best-Before date? Who owns the copyright? Where may it be applied and where not? All relevant information should be written in the program as **comments**.
- In Java, everything between `/*` and `*/` is a **comment**. 
- Everything between an opening `/**` and `*/` is considered by **JavaDoc**. 

# Strings

- If a String contains " then it is represented by \".
- Strings can be put together (concatenated) by +.  
E.g. "Hello" + "World" is the string "HelloWorld".
- We can print Strings by something like  
`System.out.println("Hello World");`

*NaN*

*round(3.6) → 4*  
*ceil(4.1) → 5*  
*floor(4.7) → 4*

*Math.random() → 0~1*  
*Math.sqrt()*

*Math.PI*  
*Math.sin()*

*+ - \* / % sqrt*  
*exp*  
*Integer / float (double)*

# Variables

- All variables must be **declared** with a type such as `public int x;`.  
[Note, if Java can infer it, it may be left undetermined (keyword `var`).]
- It is possible to declare several variables of the same type in a single step `public int x, y, z, u, v;`
- Before a variable is used it must be **initialized**, that is, it must be given an initial value. E.g., by the **assignment** `x = 3;`.  
The value can be overwritten by a new assignment, e.g., by writing after the first assignment a second such as `x = 4;`.  
(In such a case the previous value is irrecoverably gone.)



# Variables (Cont'd)

Think of variables as names plus their values, represented in a table as follows:

name	x	y	z	u	v
val	-	-	-	-	-

After an assignment `variable = term;` the term on the right hand side is evaluated with respect to the current values in the table.

The assignment `x = 3;` means that the table is updated, any old value of `x` (or no value in this case) is overwritten by the new value. If the old value is still needed, it must be copied before the assignment, since after the assignment the value is lost.

name	x	y	z	u	v
val	3	-	-	-	-

## Variables (Cont'd)

name	x	y	z	u	v
val	3	-	-	-	-

After the assignment  $y = 2*(x+5)$ ; the table is computed by evaluating the expression  $2*(x+5)$  in the OLD TABLE and then overwriting the current value of y by the result.

# Variables (Cont'd)

name	x	y	z	u	v
val	3	-	-	-	-

After the assignment  $y = 2*(x+5)$ ; the table is computed by evaluating the expression  $2*(x+5)$  in the OLD TABLE and then overwriting the current value of y by the result.

name	x	y	z	u	v
val	3	16	-	-	-

## Variables (Cont'd)

name	x	y	z	u	v
val	3	-	-	-	-

After the assignment  $y = 2*(x+5)$ ; the table is computed by evaluating the expression  $2*(x+5)$  in the OLD TABLE and then overwriting the current value of y by the result.

name	x	y	z	u	v
val	3	16	-	-	-

After the assignment  $y = 2*(y+x*x)$ ;

## Variables (Cont'd)

name	x	y	z	u	v
val	3	-	-	-	-

After the assignment  $y = 2*(x+5)$ ; the table is computed by evaluating the expression  $2*(x+5)$  in the OLD TABLE and then overwriting the current value of y by the result.

name	x	y	z	u	v
val	3	16	-	-	-

After the assignment  $y = 2*(y+x*x)$ ;

name	x	y	z	u	v
val	3	50	-	-	-

# Variables (Cont'd)

name	x	y	z	u	v
val	3	-	-	-	-

After the assignment  $y = 2*(x+5)$ ; the table is computed by evaluating the expression  $2*(x+5)$  in the OLD TABLE and then overwriting the current value of y by the result.

name	x	y	z	u	v
val	3	16	-	-	-

After the assignment  $y = 2*(y+x*x)$ ;

name	x	y	z	u	v
val	3	50	-	-	-

After the assignments  $z = x+y$ ;  $u = 3*z$ ;  $x = 0$ ; the table is:

## Variables (Cont'd)

name	x	y	z	u	v
val	3	-	-	-	-

After the assignment  $y = 2*(x+5)$ ; the table is computed by evaluating the expression  $2*(x+5)$  in the OLD TABLE and then overwriting the current value of y by the result.

name	x	y	z	u	v
val	3	16	-	-	-

After the assignment  $y = 2*(y+x*x)$ ;

name	x	y	z	u	v
val	3	50	-	-	-

After the assignments  $z = x+y$ ;  $u = 3*z$ ;  $x = 0$ ; the table is:

name	x	y	z	u	v
val	0	50	53	159	-

## Variables (Cont'd)

name	x	y	z	u	v
val	0	50	53	159	-

If variables are needed for this evaluation which do not have a value yet – indicated by the hyphen in the table – then the assignment fails and an error occurs. E.g. `v = v+1;` would fail since `v` has currently no value. However, `x = x+1;` succeeds since the expression on the right hand can be evaluated.



# Types

Java has 8 different **base types**, 4 for the representation of integers, 2 for floating point numbers, 1 for boolean values, and 1 for characters.

# Types

Java has 8 different **base types**, 4 for the representation of integers, 2 for floating point numbers, 1 for boolean values, and 1 for characters.

Integers are implemented in a cyclic way (such as a clock):

# Types

Java has 8 different **base types**, 4 for the representation of integers, 2 for floating point numbers, 1 for boolean values, and 1 for characters.

Integers are implemented in a cyclic way (such as a clock):

**byte** range -128 to 127

**short** range -32,768 to 32,767

**int** range -2,147,483,648 to 2,147,483,647

**long** range -9,223,372,036,854,775,808 to  
9,223,372,036,854,775,807

Floating point numbers (reals) are approximated by 4 or 8 bytes

**float** 4 bytes, 6-7 significant decimal digits  
3.4028235E38

**double** 8 bytes

The type **boolean** contains two elements: `true` and `false`.

The type **char** is used to represent characters such as `'a'`, `'b'`, `'c'`, and `'\u00A3'` (for the pound symbol). They are enclosed in single quotes.

Assume the computation of the areas of squares:

- $7.932 * 7.932$
- $8.2 * 8.2$
- $13.87 * 13.87$
- $123.89 * 123.89$

In order to avoid the repetition of the values we would like to write a method that computes the values as

- `square(7.932)`
- `square(8.2)`
- `square(13.87)`
- `square(123.89)`

# Methods (Cont'd)

```
public static int square(int x) {  
    return x * x;  
}
```

- **public** means that the method can be called by anybody who can access the class.
- **static** we ignore for now.
- the first **int** is the return type.
- **square** is the name of the method with which it is “called.”
- the second **int** is the type of the parameter **x**.
- **x** is the name of the parameter.
- **return** specifies what the method gives back as value.

This method is called by something like **square(4)**, e.g.,  
`System.out.println(square(4))` will print 16.

- Lectures:
  - Tuesdays 15:00-17:00, Haworth 101
  - Wed/Thu/Fri Lab Lectures 9:00-11:00 (class split, see separate sheet) in the Grace Hopper Lab
- Tutorials:
  - Fridays, 1 hour (13:00-14:00 or 14:00-15:00), see separate sheet.  
Attendance will be taken.
  - Different groups streamed according to background knowledge and self-assessment.
- MSc/ICY Software Workshop support sessions
  - Mondays, Tuesdays, Wednesdays, Thursdays 14:00-17:00.
- Weeks 5 & 10 (28 Oct 2019 & 2 Dec 2019), class tests

This module is a 40cr module over two terms. 50% of the module mark are on the 20cr of this term.

The overall mark is computed as:

- 70% Examination in May/June 2020
- 20% Continuous Assessment Term 1 and Term 2
- 10% Team Project in Term 2

This term the Continuous Assessment is in form of 4 assessed Worksheets which must be submitted via Canvas, and in form of the second Class test.

MSc students need an overall mark of at least 50, ICY students an overall mark of at least 40 in order to pass this module.

[For the MSc students on the 20cr version of the module, the examination counts 75% and the continuous assessment 25% of the module mark.]

# Continuous Assessment

Assignment	Hand out	Hand in	%	Marking
<b>WS 1</b>	Wk1	Wk2 8 Oct 2019	0	
<b>WS 2</b>	Wk2	Wk4 22 Oct 2019	3	Tests
EW 1	Wk3			
<b>WS 3</b>	Wk4	Wk6 5 Nov 2019	3	Tests
EW 2	Wk5			
<b>InClass1</b>		Wk5 28 Oct 2019	0	Paper
<b>WS 4</b>	Wk6	Wk8 19 Nov 2019	3	Tests
EW 3	Wk7			
<b>WS 5</b>	Wk8	Wk10 3 Dec 2019	3	Viva
EW 4	Wk9			
EW 5	Wk10			
<b>InClass2</b>		Wk10 2 Dec 2019	3	Paper
EW 6	Wk11			

[Changes possible but not planned]. All Worksheet hand-ins are Tuesday at 12 noon sharp. Late submission invokes a penalty of 5 marks. No hand-in after 24 hours.

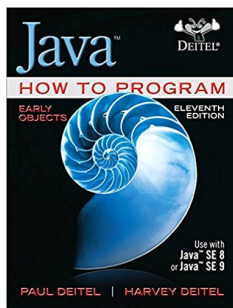
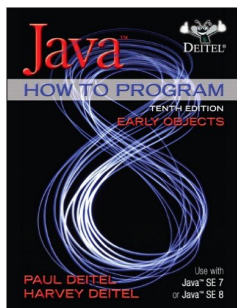
[For the MSc students on the 20cr version of the module, each piece of continuous assessment is worth 5 marks.]



- You can and are advised to cooperate in your revision, in the lab lectures, the non-assessed exercises, and in understanding the assessed exercises.
- You **must not collaborate** in writing the code for the assessed Worksheets and also **must not cheat** in the two in-class tests. If we find out we will take disciplinary measures.
- The University's Code of Practice on plagiarism is at <https://intranet.birmingham.ac.uk/as/registry/legislation/documents/public/Cohort-Legislation-2019-20/CoP-Academic-Integrity-19-20.pdf>, and the School guidance notes are at <http://www.cs.bham.ac.uk/internal/taught-students/plagiarism>

# Reading

In order for the lectures being most effective you are expected to have read the corresponding material **BEFORE** the lectures!



Paul and Harvey Deitel, Java - How to Program, Ed. 10 or 11

Week 1: Chapters 1 & 2

Week 2: Chapter 3

[Note: There are other books which cover the same material. You are free to choose a different book for your reading. However, I will indicate the reading according to the book above.]

# What Next?

- ① **Lab lecture** Grace Hopper Lab, Wed/Thu/Fri 9-11
- ② **Tutorials** Fri 13-14 or 14-15.
- ③ Follow the pages on  
<https://birmingham.instructure.com/courses/38428>
- ④ **More Examples**