Java: short history and basic features

Chapter 1 – “Big Java” book
Chapter 1 – “Head First Java” book
Chapter 1 – “Introduction to Java Programming” book
Chapter 1 – “Java in a Nutshell” book

Past, Present and Future

- History:
  - Sun Microsystems began working on a programming language for “information appliances” – mobile phones, PDAs.
  - Requirements → safe language, small footprint, efficient development
- However too soon …
  - As PDAs failed, emphasis focussed on embedded software systems such as Set-top boxes; at this point the language was called Oak.
  - In 1993, the World Wide Web (WWW) started to take off and Mosaic (first graphical browser) was released!
- As the WWW started to become more used …
  - Oak was renamed to Java
  - Sun redevelops Mosaic (“Hot Java”) as a web browser.
  - The browser had the ability to dynamically download Java code → these are called applets

PDA = Personal Digital Assistant

What Java does for the WWW

- Java allows users to interact with a web page. It allows users to:
  - play games
  - calculate spreadsheets
  - chat in real time
  - get continuously updated data
  - listen to inline sounds that play in real time whenever a user loads a page
  - cartoon style animations
  - real-time video

  and much, much more …

Java is not just a ‘web thing’

- Does almost anything other traditional languages (e.g. C++, Pascal) do.
  - But has learnt from other languages’ mistakes; it is cleaner and easier to use.

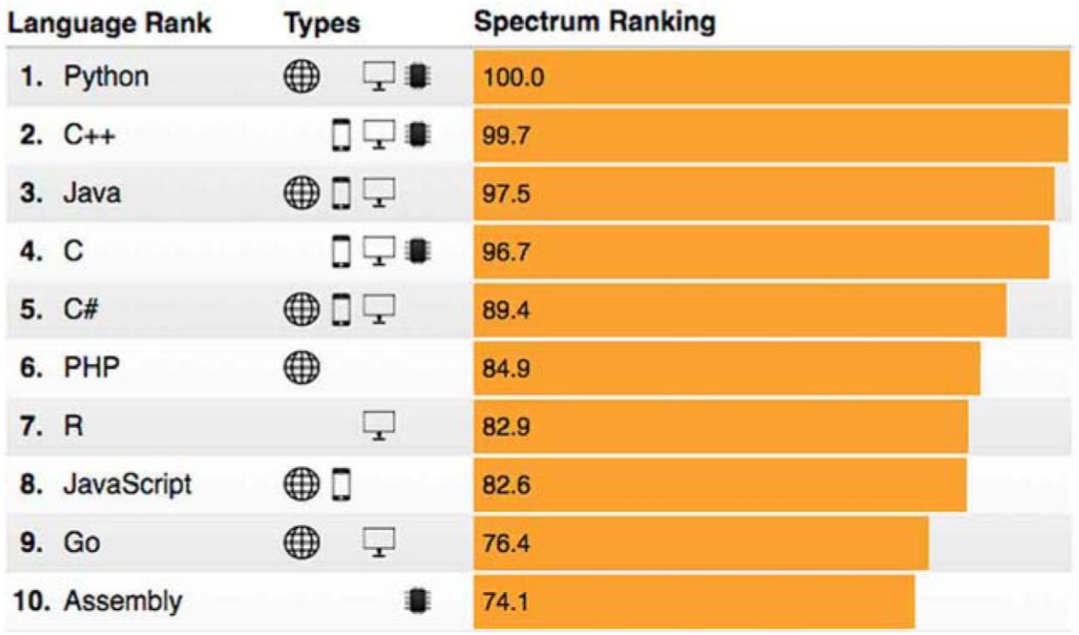
- Similar to C++, however Java has:
  - no automatic type conversion, it is strongly typed
  - no pointer operations
  - no `GOTO` statement, no global variables, no header file
  - no `struct` and `union` blocks, no templates
  - automatic garbage collection, so no need to “free” memory
  - no multiple inheritance (more about this later)
  - no backward compatibility with C, like C++

What is Java: General Characteristics

- Java is:
  - A simple language
  - Object Oriented (OO) – rich, well-designed class library
  - Platform independent – “Write once, Run anywhere”
  - Robust – error checking
  - Secure – security features
  - Multi-threaded – for efficient operation
  - Dynamic – extensible

What is Java: Present

- Software to compile and run Java is released free for all major operating systems
- Huge backing from industry
- Mass market OO language
- Rich and still growing language with a vast collection of classes supporting, e.g. Graphical User Interface (GUI) and multi-media developments

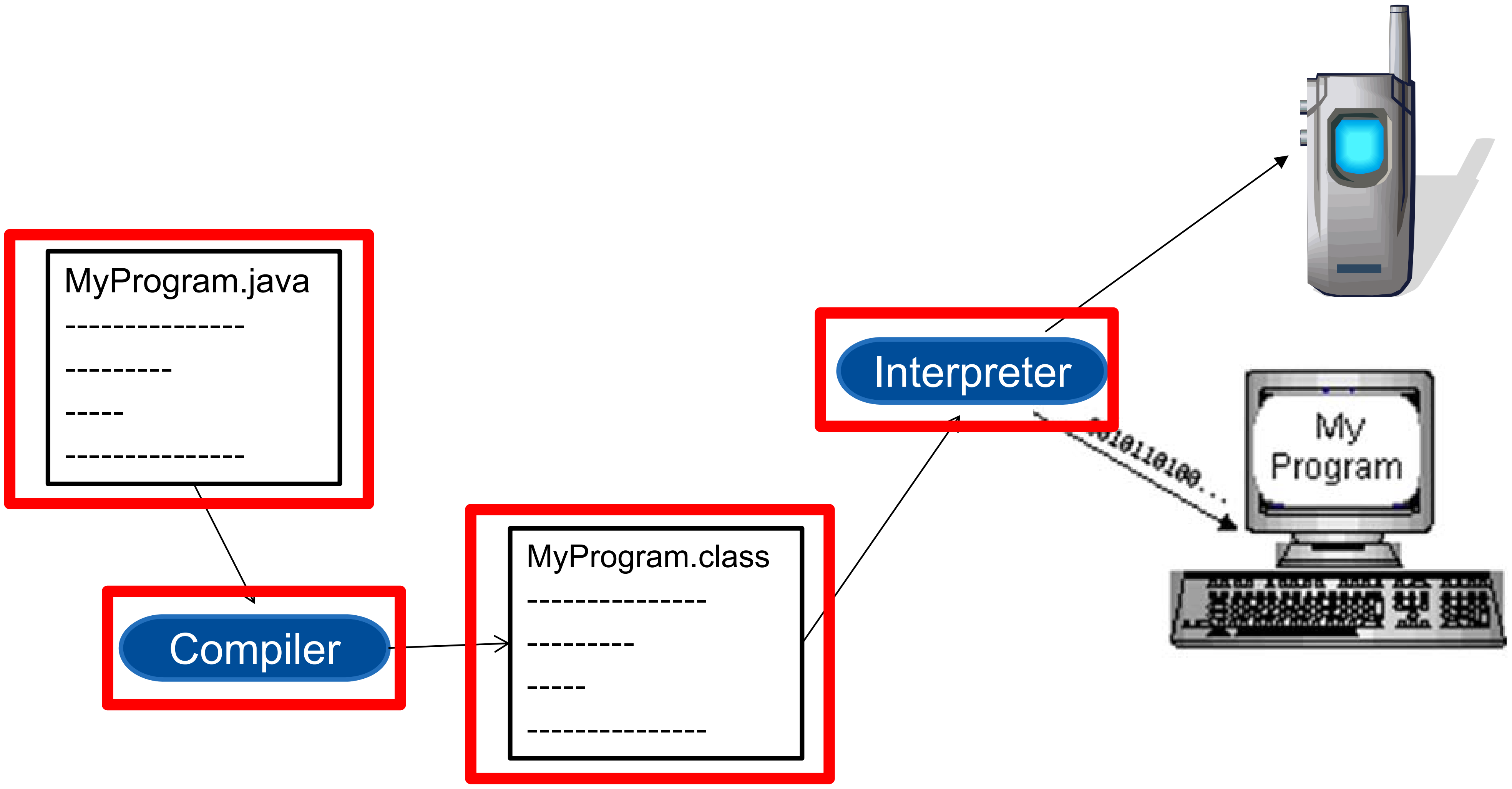


Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥 ▪	100.0
2. C++	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	97.5
4. C	📱 🖥 ▪	96.7
5. C#	🌐 📱 🖥	89.4
6. PHP	🌐	84.9
7. R	🖥	82.9
8. JavaScript	🌐 📱	82.6
9. Go	🌐 🖥	76.4
10. Assembly	▪	74.1

- Following Oracle Corporation's acquisition of Sun Microsystems in 2009-2010, Oracle has described itself as the:

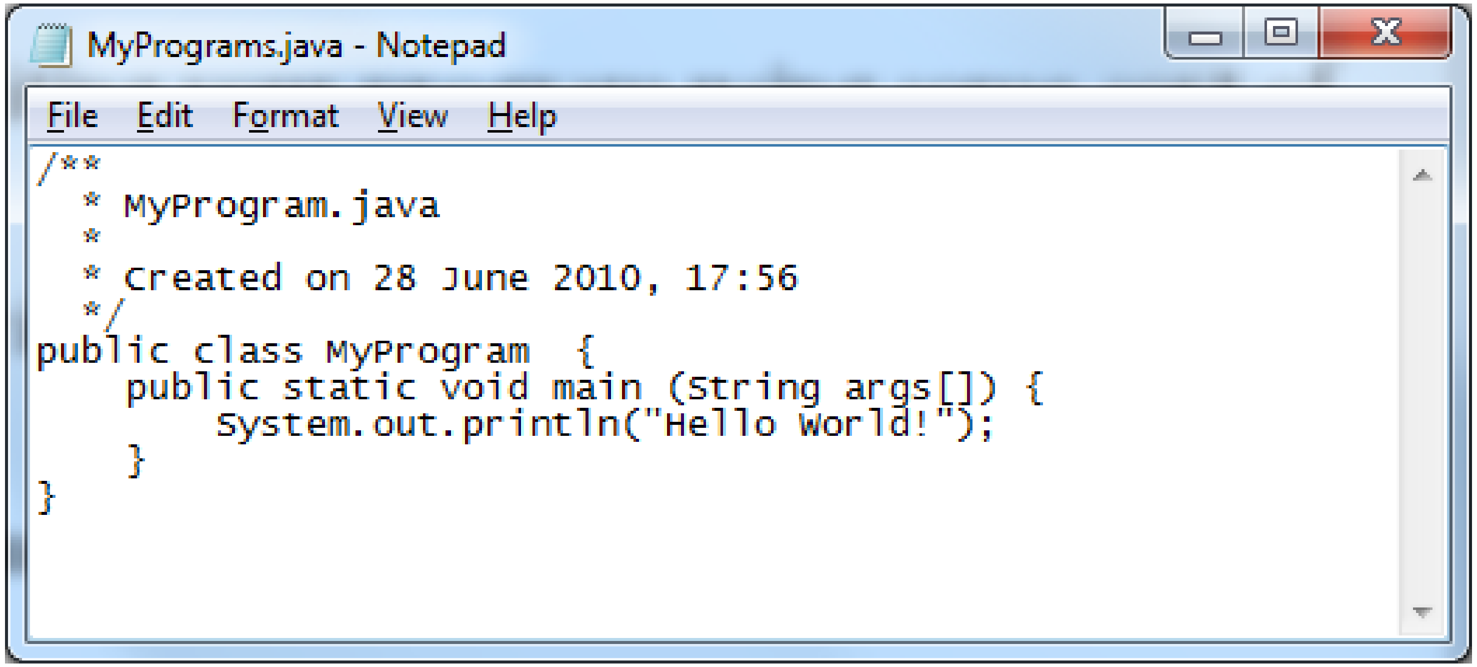
  "steward of Java technology with a relentless commitment fostering a community of participation and transparency"

Phases of a Java Program



Phase 1: Edit

- This phase consists of writing your program with some sort of text editor (e.g. Notepad).
- Every Java program is written in a text file with the extension **`.java`**
  - In this example, we save our program as **`MyProgram.java`**



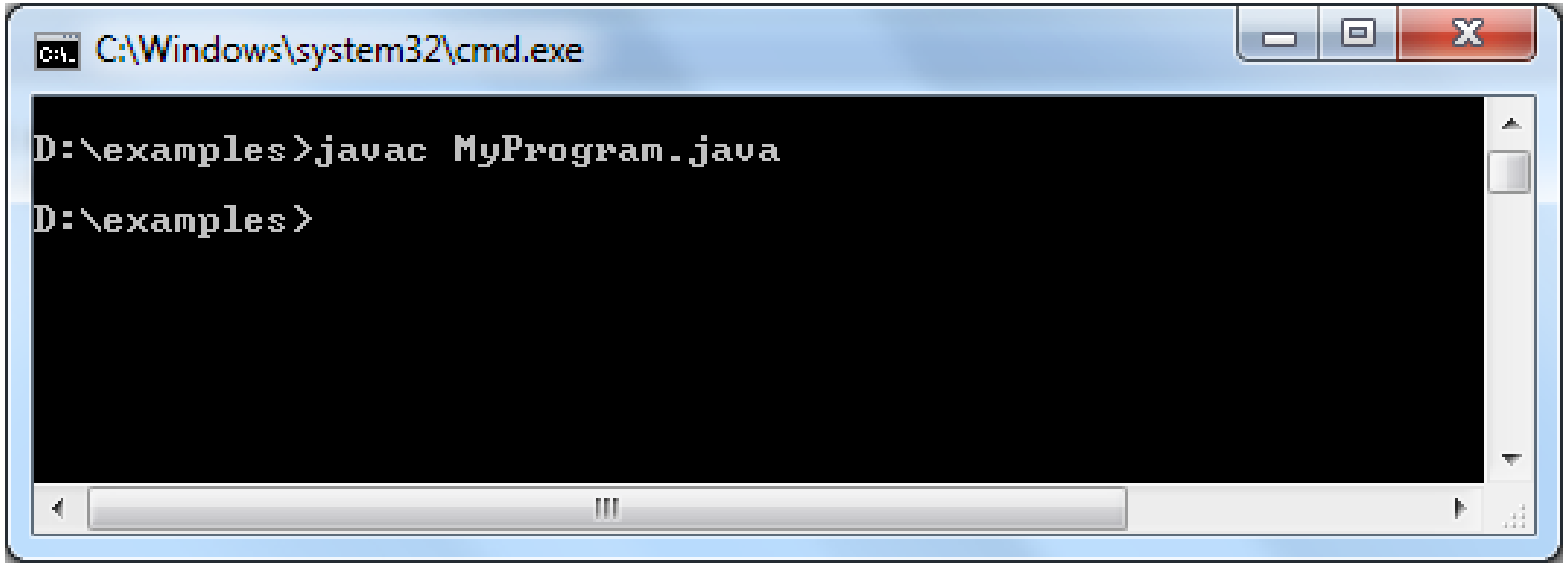
```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram  {
    public static void main (String args[]) {
        System.out.println("Hello World!");
    }
}
```

Phase 2: Compile

- The program is now compiled using the Java compiler using the command **`javac`**

  `javac MyProgram.java`

- A Java compiler translates your program into bytecode so that the Java interpreter can read the program → **`MyProgram.class`**



```
D:\examples>javac MyProgram.java

D:\examples>
```

Phase 3: Load and Phase 4: Verify

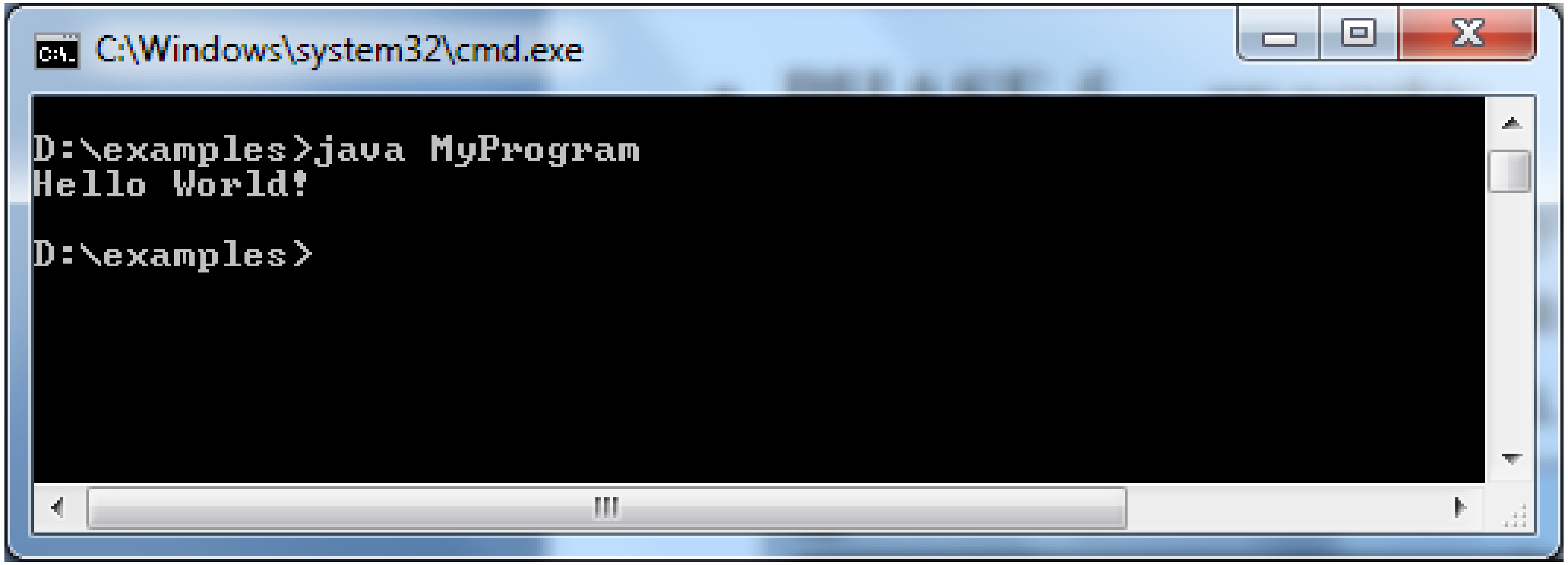
- Load: the application must be loaded into memory before it can be executed.
  - A class loader takes the `.class` file (bytecode) and loads it into memory.
  - The Java interpreter does this via the command `java`:

```
java MyProgram
```

- Verify: Consists of running a bytecode verifier; it ensures:
  - All bytes are valid
  - Does not violate Java security restrictions

Phase 5: Execute

- Computer interprets bytecode via a virtual machine (one byte at a time), and performs the actions in the program.



```
D:\examples>java MyProgram
Hello World!

D:\examples>
```

Notes: Common Problems

- The file name and the class name must be the same.

```
MyProgram.java
```

- To compile the program, type (with file extension):

```
javac MyProgram.java
```

- To run the program, type (without file extension):

```
java MyProgram
```

- Java is Case sensitive!
- Compiler catches syntax and (some) semantic errors.
- Output should be:

```
Hello World!
```

Basics of Java

- We are now going to show how a simple Java program runs on a PC
  - It is a simple console application to print some text back to the screen; this demonstrates the basic layout of a Java application.
  - We will use the JDK available for free from Oracle at http://www.oracle.com/technetwork/java/javase/downloads/index.html

JDK = Java Development Kit

But first … Basic application in C

- Do you remember? …
  - A C program is made up of a set of “functions” (cf. “procedures”)
  - Program execution starts in a function called **`main`**
  - Here is the “Hello world” program written in C:

```
#include <stdio.h>
int main() {
  printf(“Hello World!”);
  return 0;
}
```

The first Java program (1/7)

```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- Prints `Hello World` to the console.

The first Java program (2/7)

```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- Comments are ignored by the compiler.

The first Java program (3/7)

```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- The curly brackets `{}` are used to delimit blocks of code, and must be in pairs.
- Indentation highlights the structure of the code.

The first Java program (4/7)

```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- Define a class: a Java application is made up of a collection of classes.

The first Java program (5/7)

```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- Each application must have a **`main()`** method.
- The interpreter executes **`main`** first.

`main()` method header

```
public static void main(String[] args)
```

method name: `main` means the entry point for the program

`static`: means there's a single instance of the method and it's associated with the class rather than individual objects (more later)

access restrictions: `public` means it's globally accessible

type of value returned by method: `void` means it returns nothing

command line arguments: accessed as an array of `String`s (more later)

The first Java program (6/7)

```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

or **`print`**   or a variable

- Statement: console output can be achieved using
  `System.out.println(`*`“a string”`*`);`

The first Java program (7/7)

```
/**
  * MyProgram.java
  *
  * Created on 28 June 2010, 17:56
  */
public class MyProgram {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- Each statement ends with a semicolon ; and strings (or text) must be within “”.

Java Comments

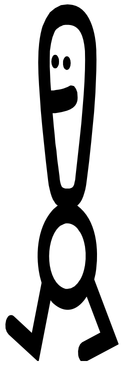
- There are 3 formats of Java comments:

```
// single line comments

/* many line comments,
   end with */

/**
  * comments here are put in a javadoc file
  */
```

… and things for you to try out!

General Notes: Java applications

- Programs should always follow the KISS principle
  - This means that if there are, e.g. two ways to do something and one is shorter or simpler than the other one, then you should always choose the simplest way!
- ALWAYS test your programs on every platform that you intend them to be run on!
  - Although it is easier to write portable programs in Java, it is not a foolproof process.



NOW
To compile and run Java programs, all you need is:
1. JDK
2. Text Editor (e.g. Notepad)

LATER IN THE COURSE
- We will also be using (and demonstrating examples with) the Eclipse IDE.
- Please avoid using any IDEs for now.

KISS = Keep It Simple Stupid
JDK = Java Development Kit
IDE = Integrated Development Environment

Installing the JDK

- The Java Virtual Machine
- Standard Java tools:

  `javac`: compiler

  `java`: launcher for Java applications

  `javadoc`: API documentation generator

  `jar`: manages JAR files

  `jdb`: Java debugger

  `javap`: class file disassembler

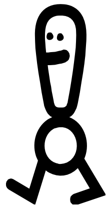
API = Application Programming Interface
JAR = Java ARchive

- The JDK is installed in the Computing Lab computers – you do not need to do anything.
  - If you want to do Java exercises using your own computer … you need to install the JDK and configure it
  - Please follow the instructions in the document in QMplus, under *Teaching Week 1 > Installing JDK & Essential Command Line Guide*

Exercises: things for you to try out

1. Go to the lab or install JDK on your own computer
2. Write, compile and run the first Java program (see previous slides)

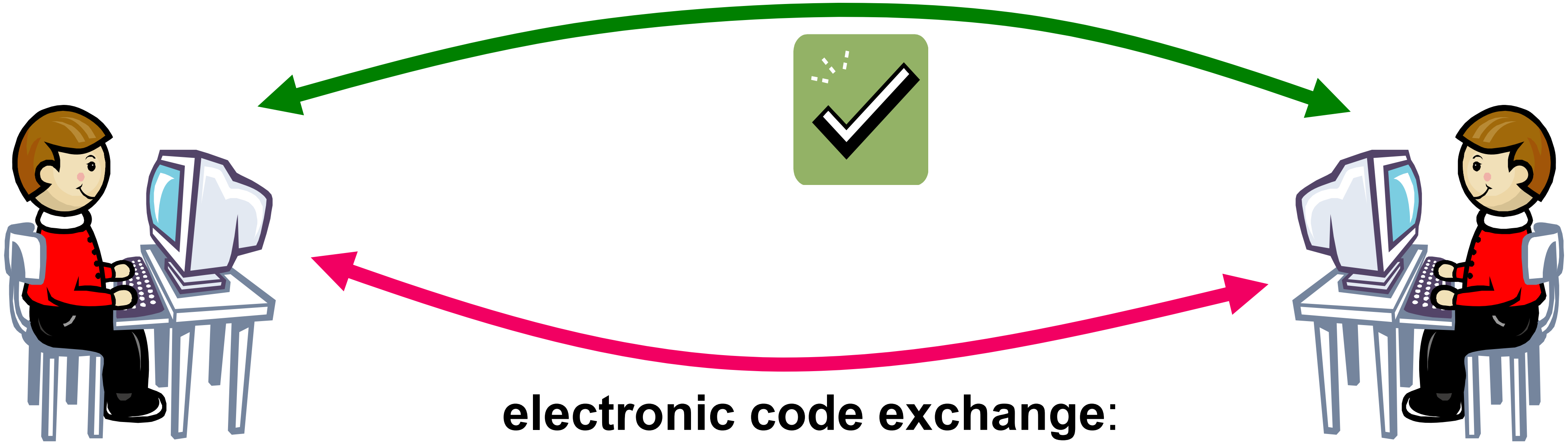
   You may experience some errors …
3. Modify this program to display your name on the screen
4. Modify this program to display your name and your email in the same line
5. Modify this program to display your name and your email in different lines



- From now on, get in the habit of trying (by yourself) to write, compile and run ALL the examples and exercises shown in the lecture slides.
- The only way to become a good programmer is to practice, practice, practice …

Plagiarism

talk: discuss course content and alternative approaches



electronic code exchange:
don't share code

Plagiarism is strictly forbidden!

- What is it?
  - The reproduction (or copying) of ideas, words or statements of another person without appropriate acknowledgement

- Examples:
  - A student knowingly permits another to submit his/her work
  - Presenting someone else's work as your own, without giving due credit
  - Using programs or program fragments of others without giving due credit
  - Writing or using programs based on the ideas, theories or algorithms of others without giving due credit

even if you have permission from the author

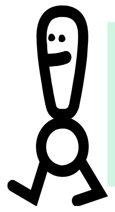
Copyright

Who owns the copyright?

- Normally, the individual or collective who authored the work will exclusively own the copyright.
- However, if a work is produced as part of employment, then the copyright belongs to the person/company who hired the individual.
  - QMUL holds the copyright of your coursework and your project!

Penalties for breaking copyright law

- This can have serious consequences for you and the University, threatening not only your own reputation and prosperity, but also that of the University
- Also, the University may find it more difficult to negotiate agreements that would make software more widely (and less expensively available) to students



The department will pursue any suspected cases of plagiarism!

How do we give credit?

- Always make clear what is written by yourself and what was written by others
- Indicate clearly in the source code:
  - The author of the code or algorithm used
  - The source where the algorithm or code was obtained from
  - A description of any alterations made by yourself
- If you are re-using code written by yourself for a previous purpose this should also be made clear, so that the extent of the new work can be determined.

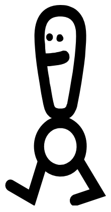
Academic Honesty: examples

DISHONESTY HAS OCCURRED:	DISHONESTY HAS NOT OCCURRED:
• When a student turns in the work of another student and represents it as his or her own work. • When a student knowingly permits another to turn in his work. • When a student copies code from the work of another student. • When a student deliberately transforms borrowed sections of code in order to disguise their origin. • When several students collaborate on a project and fail to inform the instructor of this. • When a student steals or obtains examinations, answer keys, or program samples from the instructors files or computer directories. • When a student modifies or deletes another student's or an instructor's computer files.	• when students have permission to collaborate on a project, and list all collaborators. • When students receive advice from instructors, teaching assistants, or staff members involved in the course. • When students share knowledge about syntax errors, coding tricks, or other language-specific information that makes programming easier. • When students engage in a general discussion of the nature of an assignment, the requirements for an assignment, or general implementation strategies. • When students compare independent solutions to an assignment in order to better understand the nature of the assignment. • When students engage in discussion of course concepts or programming strategies in preparation for an assignment or examination. • When students copy code and cite its source on assignments for which the instructor allows inclusion of code other than the student's own.

How to correctly reference your code

You are asked to write a java address book program. Much of the address book implementation comes from a chapter in the Barnes and Kölling textbook “Objects First with Java”. Here is what the citation would look like:

```
/**
 * A class to maintain an arbitrary number of contact
 * details. Details are indexed by both name and phone
 * number.
 * Class taken from: Barnes, and Kölling; “Objects
 * first with Java”, Prentice Hall, 2005;
 *
 * Changes: Added the ability to search by email
 * and address.
 * @author David J. Barnes and Michael Kölling
 * @author Your Name
 * @version 05/03/2017
 */

public class AddressBook {
```



If the code you are referencing is from a Textbook or from the Web.

The **red** text are your additions. However, when commenting your code it does not have to be in a different format.

How to reference code given by the lecturer

Some code is given to you by your class instructor.

```
/**
 * This class creates a pack of tiles that can be used
 * for playing games. It demos the deck by allowing the
 * user of the program to select a tile and also to mix
 * up the tiles in the pack. This class provides a
 * graphical interface for showing how the classes Tile
 * and TilePack work.
 *
 * This code was given out in class on 29 September 2003
 * by Instructor Name. The no arguments constructor and
 * main methods were provided.
 * Changes:
 * 1. Added the method doLayout(), which moves
 * GUI painting to a separate method in order to
 * reuse the code in refreshing the GUI.
 * 2. Continue documenting large changes
 *    (added methods, moved code etc.)
 *
 * @author Instructor Name
 * @author Your Name
 * @version 1.1
*/

public class TilingDemo {
```

The **red** text is your additions. However, when commenting your code it does not have to be in a different format.

Small sections of code (including use of algorithms)

You are developing a program yourself. In this program you need a search function. You find a suitable implementation that you subsequently adapt to work with your code (in either a book or on the web).

```
/**
 * Title: MyClass.java
 * Javadoc description of the class.
 * @author Your Name
 * @version 1.1
*/

public class TilingDemo {

/* The rest of your class */

/**
 * This method performs a search on the data
 * structure. It has been adapted from the code/
 * algorithm presented {in textbook "name" by
 * author/ at www.WhereYouFoundIt.com/index.hmtl".
 * @param name The name of person whose record
 *             should be found.
 * @return The record ID.
 */
public int search(String name){
 /* Code for search */
}
```

Other reuse of code

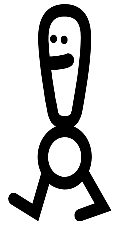
- You should indicate all the sources you used, in as much detail as possible.
  - Highlight all changes you have made; always stay on the side of caution.
  - Remember that if a class or method is found to have been copied without acknowledgement → you get 0
  - If you "lend" your code to someone else, it doesn't matter who copied and who wrote it → you both get 0

- If you are uncertain about how to comment/reference something, the surest procedure is to be scrupulously honest and complete in your acknowledgements.
  - If you are still unsure about how to document something, ASK

Plagiarism is not tolerated!

- Respect for the intellectual work of others is part of the ethos of universities.
  - Proper attribution encourages free exchange of ideas and promotes progress.

- All students must complete their own work <u>and</u> are expected to behave with integrity at all times.



Plagiarism is strictly forbidden; there are severe penalties when detected! More information about this is in:

- http://www.arcs.qmul.ac.uk/media/arcs/policyzone/academic/Academic-Regulations-2018-19-FINAL.pdf
- your Student Handbook.

Questions?