

Topic 1: Getting Started

Programming Practice and Applications (4CCS1PPA)

Dr. Martin Chapman
Thursday 29th September

programming@kcl.ac.uk
martinchapman.co.uk/teaching

Before We Start

SYMBOLS TO LOOK OUT FOR



This content is particularly important, and **will** be tested in either the coursework, class test or exam.



This is for interest only, and will **not** be tested in either the coursework, class test or exam.



There is content in this slide that we will revisit at a later date.

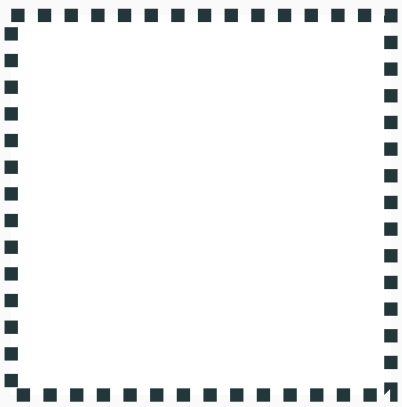


This is not my area of expertise, but may still be interesting or relevant to another module. This will **not** be tested.

SYMBOLS TO LOOK OUT FOR



This slide features animations, which were presented in the lecture, and may not appear well in print.



Slides with no symbol are all of equal importance, and **may or may not** be tested in the coursework, class test, or the exam.



You should be implementing **all the code** in these slides in your lab sessions (**line-by-line**), but this symbol tells you that this code is **particularly important**, or alerts you to the presence of **particular tasks**, which need to be completed.

GETTING STARTED: OBJECTIVES

To understand a **definition** of programming, and some open areas of **debate** around the subject.

To **write** and **run** one of the simplest Java programs.

Some Starter Questions...

What Is Programming?

A PROPOSED DEFINITION

Most people know **informally**.



Formally?

- "Writing a sequence of instructions that are designed to solve a particular problem in a formal, constructed language."
- "Writing stuff in a made up language on your computer so that other stuff happens on your computer."

PROGRAMMING: AN ART OR A SCIENCE?

This topic is **often debated**.

Certainly mostly scientific.

- A **set language**; the **logical** resolution of problems.

But definite artistic elements.

- Visual code **style**; **creative** solutions.

Because of the artistic element of programming, the code I show you, and, to some extent, the topics I share with you, will always be **influenced by my style**.

- Others, perhaps even those of you with programming experience, may **prefer** to do things in **different** ways.
- The **fundamentals** are **style-independent**.

Why Is Programming Important?

ENOUGH SAID



When Can You Call Yourself `A Programmer`?

WHEN CAN YOU CALL YOURSELF `A PROGRAMMER'? (1)

When you know the **syntax** (key words) and **semantics** (the meaning of these words) in every programming language?

- I don't think so.
- This is the `**how**' of programming.
- This can be obtained simply through **memory**.

WHEN CAN YOU CALL YOURSELF `A PROGRAMMER'? (2)

When you develop a **sense** for how to **solve** problems by **breaking them** down; you're able to **abstract** the world; and you know which **approaches** to apply in different circumstances

- I think so.
- This is the `**what**' of programming; even the pros use **web searches** for the `**how**' element of programming, but the `**what**' of programming cannot be obtained from Google.
- This can only be obtained by **practising**.

PROGRAMMING VS. WRITING USING A PROGRAMMING LANGUAGE

This emphasises the **difference** between learning a programming language (e.g. learning Java) and learning to program.

In PPA, we will focus on Java, but it is important to remember that this is just a **vehicle** for teaching you how to **program**.

- (I'll say it again) **Practise** (programming), don't just memorise (Java syntax).
- This will help you develop the **fundamental skills** associated with being a programmer.
- A lengthy process that may stretch **beyond** PPA.
 - Pedagogy necessitates **toy examples**. It may not be until you meet **real programming tasks** that skills develop.

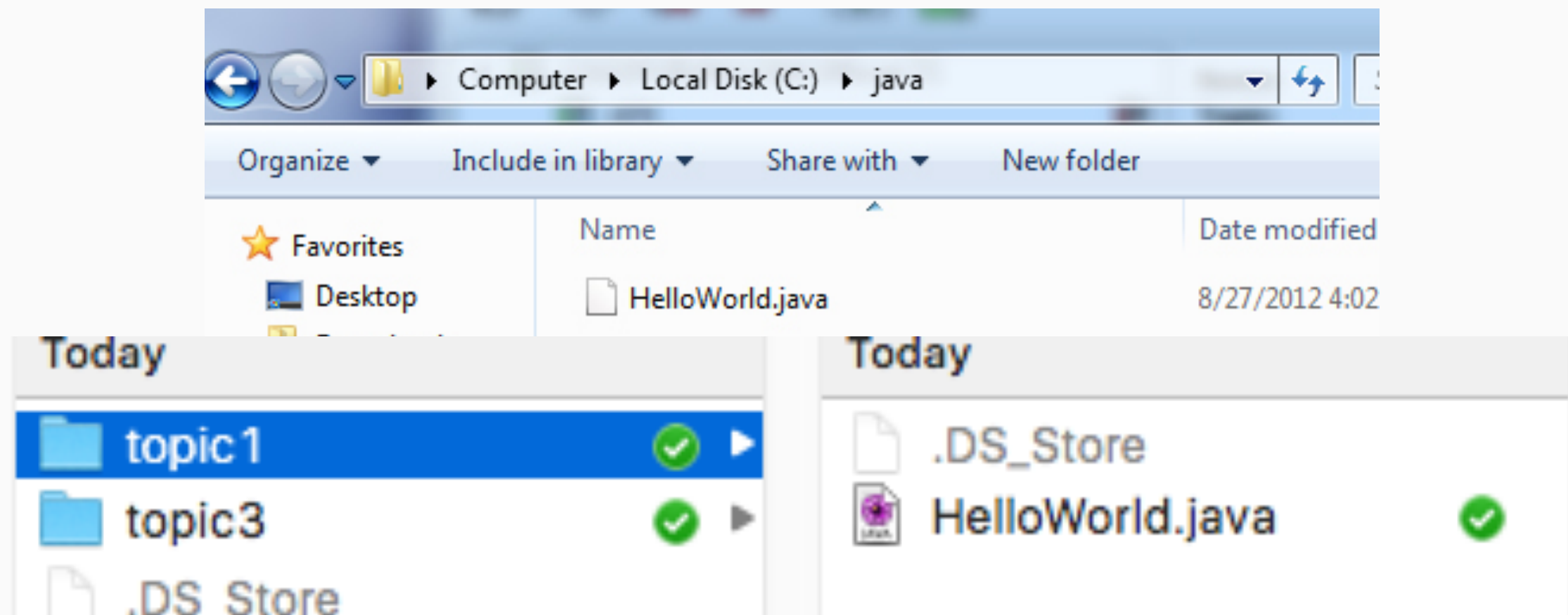
Diving In...

How Do You Program (In Java)?

START WITH A TEXT FILE

Most programs start their lives as simple **text files**, which contain our **source code**.

In Java, these files have the extension **.java** (which you typically alter manually from **.txt** after saving your text in a text editor).



UNDERSTANDING THE ROLE OF THE COMPILER (1)

We then need to **transform** this simple text file (and the source code it contains) into a **program**.

To do this, we need a specialist piece of **software** called a **compiler**.

Like other software, the compiler can be **downloaded** from the Internet (see **KEATS**).

Like most pieces of software, a compiler takes a file as **input** (our text file), and produces a different **type** of file as **output** (a program).

UNDERSTANDING THE ROLE OF THE COMPILER (2)



A music converter



A word processor

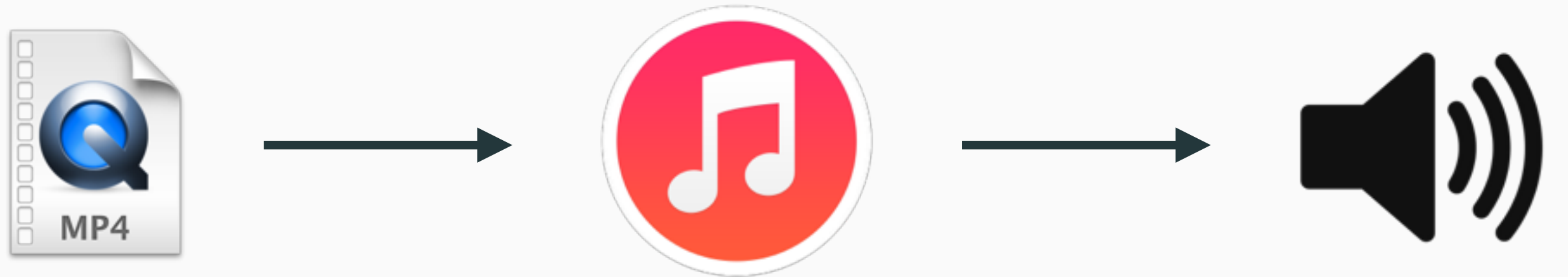


The Java compiler (**javac**)

UNDERSTANDING THE ROLE OF THE JAVA VIRTUAL MACHINE (1)

The compiler produces our program, a **.class file**, which is then read by the **Java virtual machine**, *another* piece of software (which can also be downloaded, see KEATS) designed to **run Java programs** and, typically, **produce output** that can be **viewed**.

UNDERSTANDING THE ROLE OF THE JAVA VIRTUAL MACHINE (2)



A music player

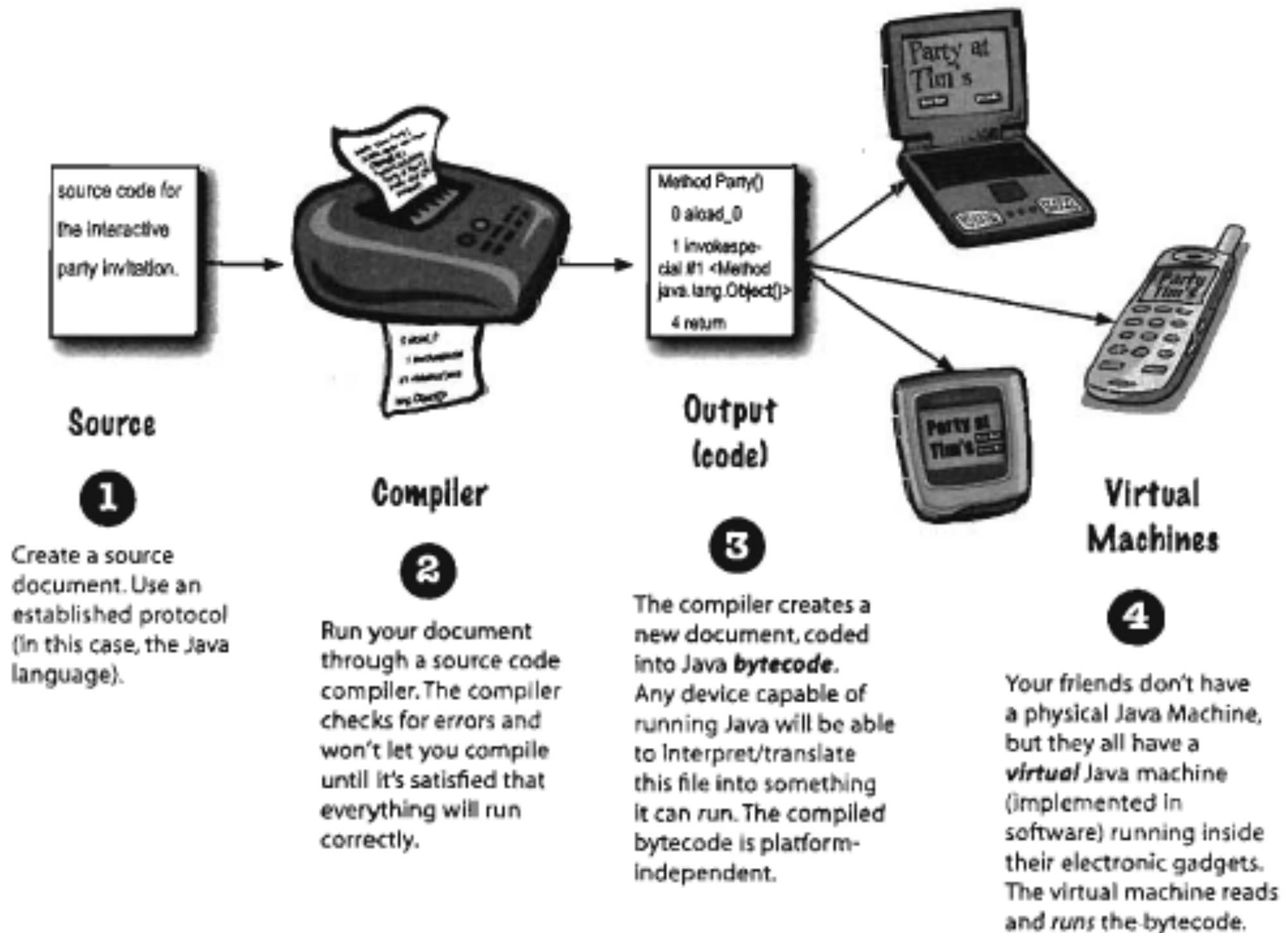


A document reader



The Java virtual machine (**java**)

THE FULL PROCESS



This compiled language (Java Bytecode) is close to assembly (4CCS1CS1).

USING THE COMPILER AND THE JAVA VIRTUAL MACHINE

When we use software such as a video converter to convert files, or we use a word processor, we **input our source files** (.mp3 file, .docx file) into the software using the **GUI**.

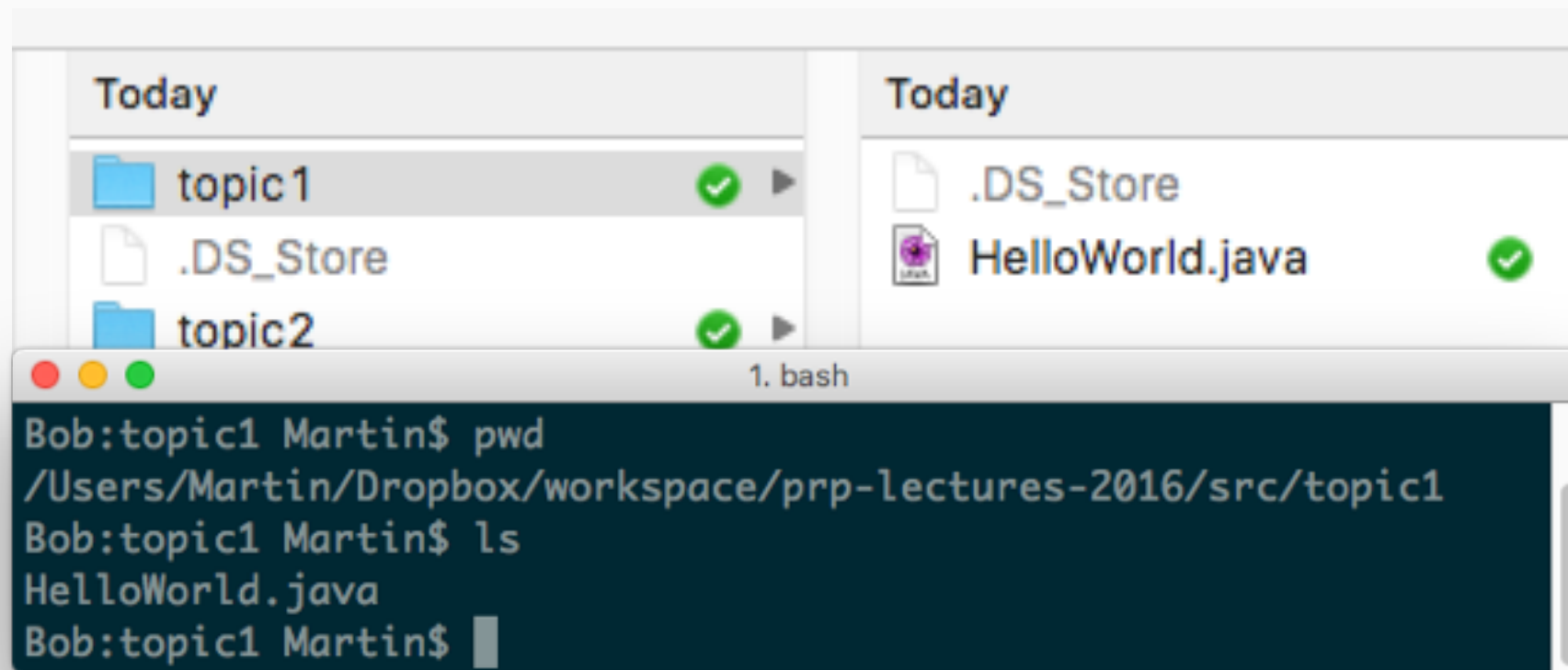
- We might right-click on the .docx file and select 'open with Microsoft Word'.

When we use software such as the Java compiler and the Java virtual machine to compile and run programs, we input our source files using a **terminal** (a **command line interface**).

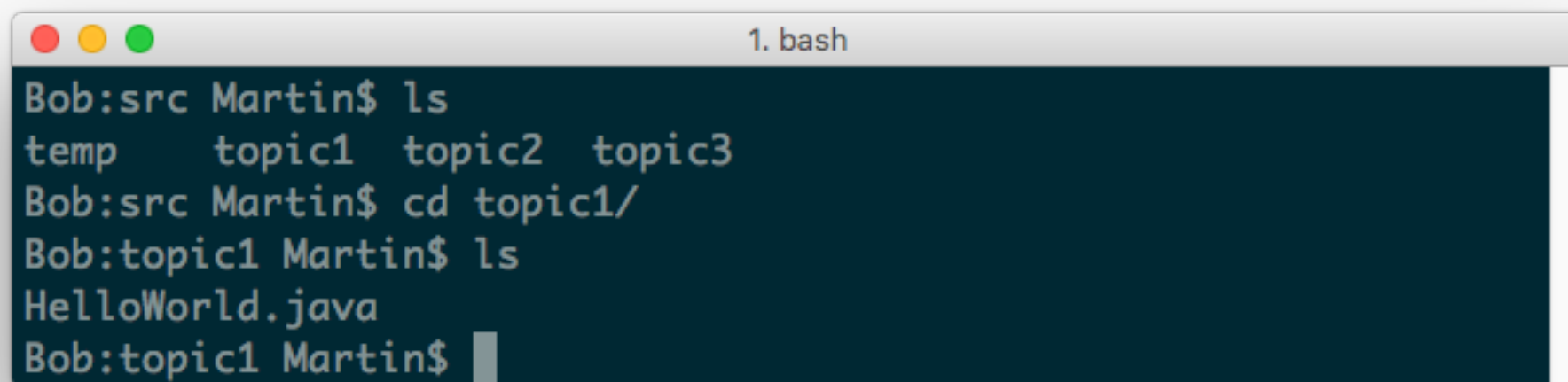
- It is possible to run compiled (Java) programs from the GUI, but it requires a more complex compilation configuration.

THE TERMINAL (1)

The terminal is a **text representation of a location in the file system**.



You issue **commands** in order to **move through the file system**.



Experiment with the terminal, and different commands, in your laboratory session. The commands you need will vary between **operating system**.

- How do you change directory?
- How do you list what is in the current directory?
- How do you find out your current location in the file system?

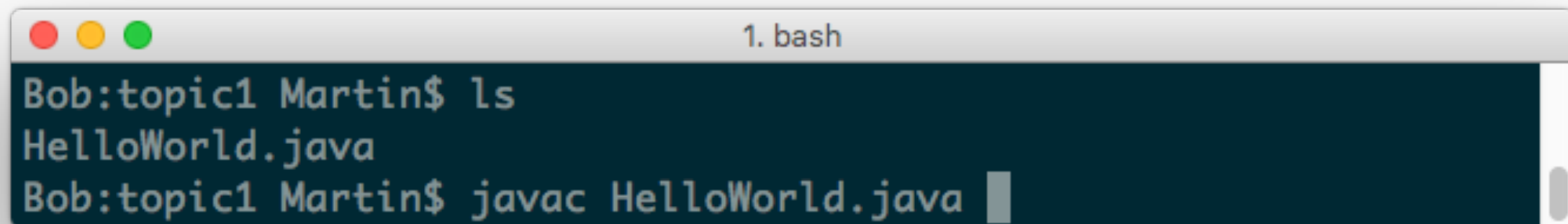
THE TERMINAL (3)

The equivalent of right-clicking on an application and selecting 'open with', on the terminal, is to write the **name of the application you wish to run**, followed by the **name of the file you wish to input**.

- This assumes that the application you wish to run is **installed** (i.e. exists) **somewhere** on your computer.
- The terminal does not necessarily know by default **where** the application you may wish to run is stored.
- So, if you wish to use the Java compiler and the Java virtual machine on your **own machine** you may have to tell the terminal where to find these two applications by **updating your terminal configuration** after you have downloaded and installed both piece of software (details on **KEATS**).

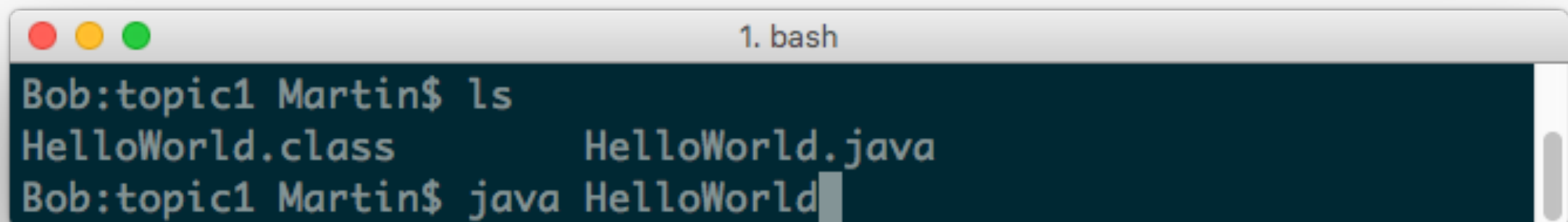
THE TERMINAL (4)

The name of the Java compiler software is **javac**.

A terminal window titled "1. bash" with a dark background. It shows the user "Bob:topic1 Martin" running the command "ls", which lists "HelloWorld.java". Then, the user runs "javac HelloWorld.java", and the cursor is at the end of the command.

```
Bob:topic1 Martin$ ls
HelloWorld.java
Bob:topic1 Martin$ javac HelloWorld.java
```

The name of the Java virtual machine software is just **java**.

A terminal window titled "1. bash" with a dark background. It shows the user "Bob:topic1 Martin" running the command "ls", which lists "HelloWorld.class" and "HelloWorld.java". Then, the user runs "java HelloWorld", and the cursor is at the end of the command.

```
Bob:topic1 Martin$ ls
HelloWorld.class      HelloWorld.java
Bob:topic1 Martin$ java HelloWorld
```

Note that we omit the **.class** extension when we are inputting a compiled Java program.

SUMMARY: FROM TEXT TO PROGRAM (1)

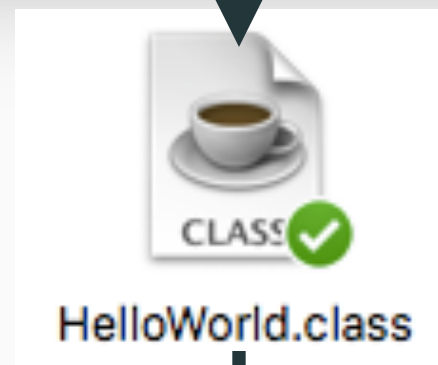
There should now be a clear process for compiling and running a Java program.

1. **Write** the program in a text file.
2. Change the **extension** of the program to .java.
3. In a terminal, **navigate** to the directory containing your .java file.
4. (Assuming the Java compiler and the Java virtual machine are installed and available to the terminal) **Run the compiler** with the .java file as input to produce a .class file.
5. **Run the virtual machine** with the .class file as input.

SUMMARY: FROM TEXT TO PROGRAM (2)



```
bash
Bob:topic1 Martin$ ls
HelloWorld.java
Bob:topic1 Martin$ javac HelloWorld.java
```



```
bash
Bob:topic1 Martin$ ls
HelloWorld.class      HelloWorld.java
Bob:topic1 Martin$ java HelloWorld
```

REMINDER: KEATS

On KEATS you will find links to download (and then install) the **Java Development Kit**, which includes the Java compiler and the Java virtual machine.

- Programming languages, like Java, come in **versions**, each with slightly different **keywords** and conventions. We may briefly touch on some of the latest Java features in this course (Java 1.8), so make sure you download the latest version. But in general we won't teach to a particular version.

Follow the instructions given on KEATS to **configure your terminal** in order to **find** the **javac** and **java** software (if needed).

- Confirm that the terminal can find this software by entering both commands into the terminal.

In our laboratories, everything is **already installed for you**.

Let's Try It Out...

A SIMPLE JAVA PROGRAM

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

Must be inside a file called HelloWorld.java

A SIMPLE JAVA PROGRAM: CODE BLOCKS (1)

*This code has a **clear structure**. It is organised into **nested blocks**, as indicated by the **curly braces**, **new lines** and **indentation**.*

```
public class HelloWorld {
```

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

```
        System.out.println("Hello World");
```

```
    }
```

```
}
```

Must be inside a file called HelloWorld.java



A SIMPLE JAVA PROGRAM: CODE BLOCKS (2)

*New lines and indentation are optional, but greatly increase **readability**.*

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

Must be inside a file called HelloWorld.java



A SIMPLE JAVA PROGRAM: THE MAIN METHOD

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

Must be inside a file called HelloWorld.java



ASIDE: BOILERPLATE CODE

A Java class requires something called a **main method** in order to be run by the Java virtual machine.

- Any code we run has to be either **in** the main method or be **called by it**.

Because the main method contains several reasonably advanced pieces of syntax, I am going to have to ask you to use this code as **boilerplate**.

- To use a piece of code without altering it (including its **case**), or really understanding its operation.
- This is considered an **pedagogical sin**, but is unavoidable.

If this does not work for you, consider **changing your learning path**

- Reminder: see the 'Support' section on KEATS.

There will be several points in the course at which I will (necessarily) ask you to accept boilerplate code, but we will always **come back to it**.

A SIMPLE JAVA PROGRAM: CLASSES AND CLASS NAMES (1)

Some more boilerplate, for now.

```
public class HelloWorld {
```

*Any code we write has to be placed inside the **block associated with** something called a **class**.*

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

```
        System.out.println("Hello World");
```

```
    }  
}
```

*The text following the class keyword is the **name of the class** and a **title** that summarises the functionality of the code inside.*

Must be inside a file called HelloWorld.java



A SIMPLE JAVA PROGRAM: CLASSES AND CLASS NAMES (2)

Each file can only contain **one** public class.

The name of this class **must match the name of the file** in which the class resides.

Therefore, at this stage, we will consider classes to be **roughly** equivalent to files.

- One file for each class.
- Later on in the course, we'll see that this isn't strictly true.

NAMING RULES AND CONVENTIONS IN JAVA (1)

Not all the words in a Java program have a special meaning. Some are **selected by us**.

```
public class IsThisLectureReallyStillGoingOn {
```

But there are rules for **human selected** text (intuitive)

- Cannot **begin with a number** (because we wouldn't be able to identify **values**).
- Cannot contain **spaces**.
- Cannot contain symbols **other** than \$ and _.
- Cannot be **reserved Java keywords** (such as class).

NAMING RULES AND CONVENTIONS IN JAVA (2)

camelCase notation is **typical**, but not enforced

- **Class names** capitalise the first letter (**HelloWorld**)

Making intelligible and presentable class name choices is part of the **artistic** element of coding.

```
public class isTHISLectur3reallystill_goingon {
```

Using the same class name twice will result in a **name conflict**.

- But any name selected will differ from the same name written in a **different case**.

A SIMPLE JAVA PROGRAM: SEMI-COLONS

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

All inside a file called HelloWorld.java

A SIMPLE JAVA PROGRAM: PRINTING OUTPUT

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

All inside a file called HelloWorld.java



TRYING IT OUT: PROGRAM OUTPUT



```
1. bash
Bob:topic1 Martin$ ls
HelloWorld.class      HelloWorld.java
Bob:topic1 Martin$ java HelloWorld
Hello World
Bob:topic1 Martin$
```

Text output from a program is printed **in the terminal**, after the program has been passed to the Java virtual machine.

Please **avoid using IDEs for the first few weeks of term (e.g. Eclipse, Netbeans, IntelliJ etc.)** or at least **disable features like code completion and code structuring.**

- Using an IDE at this stage will be like learning to ride a bike **with stabilisers**, and then **never taking them off** (you won't learn to ride properly).

You can start thinking about and exploring the potential uses of **version control.**

HOW TO PRACTISE GOING FORWARD

You actually now have all the **tools** you need to write and run Java programs.

How can you practise going forward?

- **Attend labs**, completing the lab slides and coursework tasks.
- Watch the KEATS '**Challenge Forum**' for additional challenges and problems
- **Text books** often contain hundreds of problems; specialist **coding challenge** books exist; or **search online**.
 - But try not to **jump ahead**, or approach things in different ways, unless you find yourself struggling.
- **Make something up**
 - If you want to write a program that changes the lights behind your television to match the colour of the content being played, **do it**.

Topic 1: Getting Started

Programming Practice and Applications (4CCS1PPA)

Dr. Martin Chapman
Thursday 29th September

programming@kcl.ac.uk
martinchapman.co.uk/teaching

These slides will be available on KEATS, but will be subject to ongoing amendments. Therefore, please always download a new version of these slides when approaching an assessed piece of work, or when preparing for a written assessment.