List of Slides

1 Title
2 **Chapter 1:** Introduction
3 Chapter aims
4 **Section 2:** What is programming?
5 Aim
6 What is programming?
7 What is programming?
8 What is programming?
9 What is programming?
10 What is programming?
11 **Section 3:** What is a computer?
12 Aim
13 What is a computer?
14 Computer basics: hardware
15 Computer basics: software
16 Computer basics: data

17 Computer basics: hardware: processor
18 Computer basics: software: machine code
19 Computer basics: hardware: memory
20 Computer basics: hardware: persistent storage
21 Computer basics: hardware: input and output devices
22 What is a computer?
23 Computer basics: software: operating system
24 Computer basics: software: application program
25 Computer basics: data: files
26 Computer basics: data: files: text files
27 Computer basics: data: files: binary files
28 **Section 4:** What is a program?
29 Aim
30 What is a program?
31 Java tools: text editor
32 What is a program?
33 What is a program?
34 What is a program?

35 What is a program?
36 **Section 5:** Operating environment
37 Aim
38 Operating environment: programs are commands
39 Operating environment
40 Operating environment
41 Operating environment: standard output
42 Operating environment
43 Operating environment
44 Operating environment: command line arguments
45 Operating environment
46 **Section 6:** Example:Our first Java program
47 Aim
48 Our first Java program
49 Our first Java program
50 Our first Java program
51 Java tools: javac compiler
52 Java tools: java interpreter

53 Trying it
54 Trying it
55 Trying it
56 Running under Microsoft Windows
57 Using an integrated development environment
58 Using an integrated development environment
59 Coursework: Compile and run `HelloWorld`
60 **Section 7:** Example:Our second Java program
61 Aim
62 Our second Java program
63 Our second Java program
64 Trying it
65 Coursework: Compile and run `HelloSolarSystem`
66 Concepts covered in this chapter

THE UNIVERSITY of MANCHESTER

Title

Java Just in Time

John Latham

September 25, 2018

Chapter 1

Introduction

Chapter aims

- Introduce programming via an abstract scenario.
- Look at what a computer is.
- Look at what a program is and how it is produced.
- Look at the **operating environment** of a program.
- Look at two simple program examples.

Section 2

What is programming?

Aim

AIM: To explore what we mean by programming, by using a non-computing analogy.

What is programming?

- An analogy of computer programming:
  - Training a new puppy, Spike, to walk through a maze.
  - He´s not clever enough to solve mazes!
  - First train him to obey to simple commands:
    - “sit”, “stand”, “walk”, “stop”, “left”, “right”.
  - Direct him through maze shouting commands.
    - Record them on a tiny mp3 player!
    - One track per route through the maze.
  - Hang mp3 player round his neck – with canine headphones.
  - Put him at the start of a route, play the right track and... bingo!
  - He *seems* so clever!

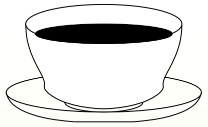
What is programming?

- You
  - created a machine that can follow simple instructions;
  - programmed it to solve complex tasks via programs.
- Just like computer programing!
  - Except
    - you don't design the language,
    - the machines are already 'trained'!

What is programming?

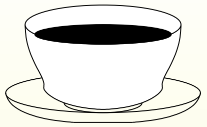
Programming is the process of **design**ing and expressing in an appropriate language, a set of instructions to solve a particular task, that a machine will obey sometime *later*, whenever that task is needed to be done.

What is programming?



Coffee time: What do you think would be the kind of problems that might make your dog machine unreliable and sometimes not work?

What is programming?



Coffee time: Apart from any previous computer programming experience you may have, which we are assuming is none, what previous experience of more general programming do you have? That is, in what situations have you done something which will cause some ‘programmed’ things to happen at a later stage?

Section 3

What is a computer?

Aim

AIM: To take a brief look at the components of a computer, including **hardware**, **software** and **data**.

What is a computer?

- Computers are dumb!
  - They blindly obey simple instructions.
    - Reliably (usually).
  - But they have no ‘understanding’ of the desired task
    - they cannot know whether the instructions are correct. . . .
- Sometime:
  - take the lid off your computer and peep inside.

Computer basics: hardware

- The physical parts of a computer is called **hardware**.
- You can see them, touch them.

Computer basics: software

- The instructions the computer will obey is called **software**.
- You cannot see it, touch it.
- It is stored on **computer media** (e.g. **DVD ROM**) and inside the computer.
  - As numbers.
- Try looking at a DVD ROM with a magnifying glass?

Computer basics: data

- You cannot see **data** either.
- It too is stored as numbers.
- E.g. digital camera image:
  - you can see the picture *represented* by the data
    - using an image viewing program.
    - It interprets the numbers and puts coloured dots on the screen.

Computer basics: hardware: processor

- The **central processing unit** (**CPU**) is **hardware**.
  - The component which obeys the instructions
    - dumbly!

Computer basics: software: machine code

- **CPU** instructions are expressed in **machine code**.
- A **low level language**.
  - Each instruction gets the CPU to do one simple thing.
    - E.g. add together two numbers,
    - send a **byte** to a printer,
    - etc..

Computer basics: hardware: memory

- The **computer memory** stores **data** for short term use.
  Including
  - **machine code** instructions currently being obeyed by **CPU**,
  - data the computer is currently using
    (e.g. digital camera image while being viewed).
- It is **volatile memory**
  - values are forgotten when power is switched off.
- Contents can be accessed and changed in any required order
  - known as **random access memory** or **RAM**.

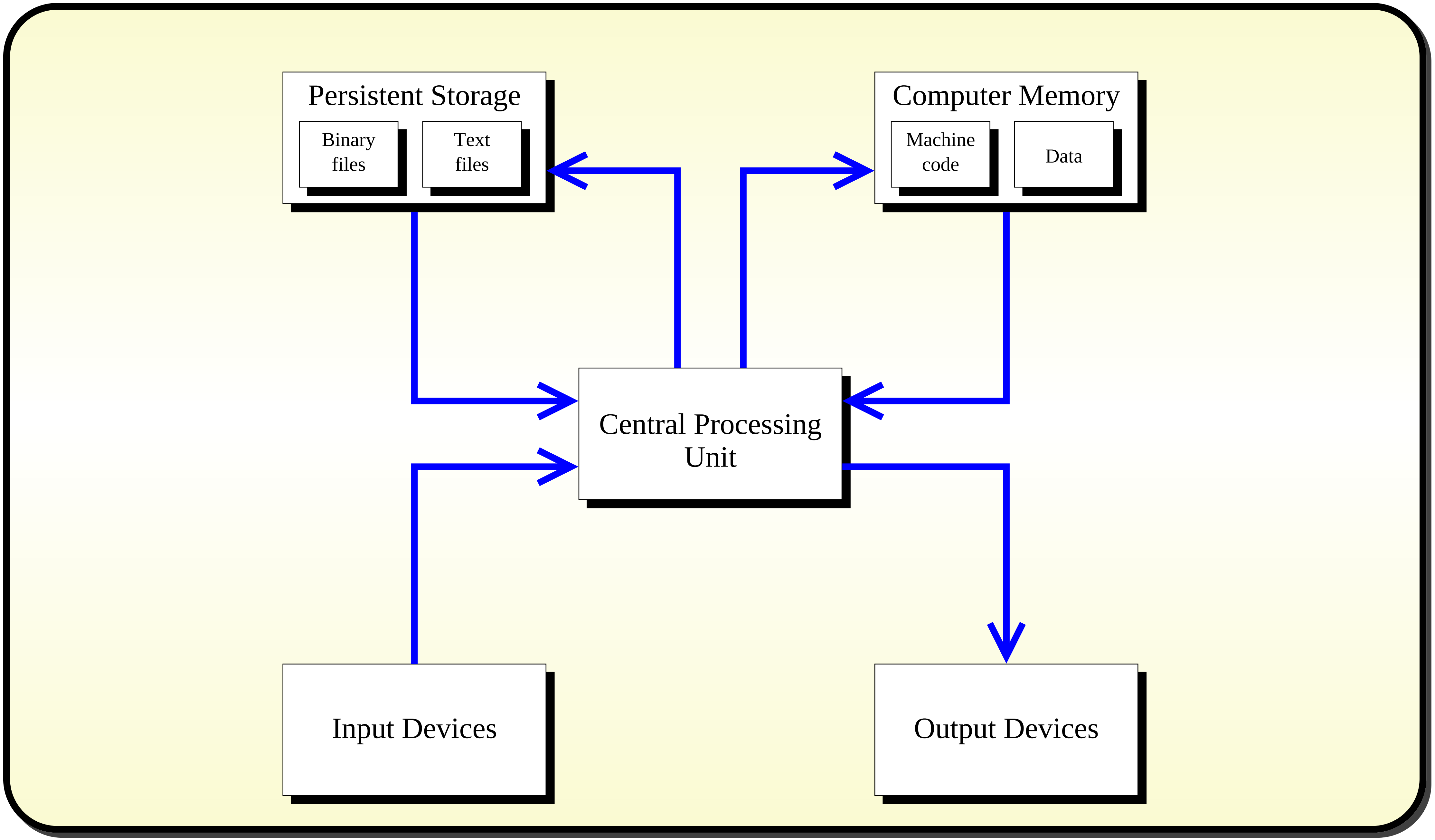
Computer basics: hardware: persistent storage

- For long term storage we have **persistent storage** devices.
  - E.g. **hard disc**s, **DVD ROM**s.
- Pros:
  - Much larger capacity than **computer memory**.
  - Persistent – remember values while power is off.
- Cons:
  - Much slower than computer memory.
  - Not efficient for random access.

Computer basics: hardware: input and output devices

- Keyboards and mice are examples of **input devices**.
- Displays and printers are examples of **output devices**.

What is a computer?

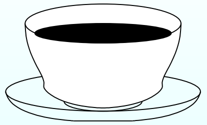


Computer basics: software: operating system

- An **operating system** – **software** for making the computer generally usable.
  - E.g. Microsoft Windows, Mac OS X and Linux.
  - Mac OS X and Linux are implementations of Unix.

Computer basics: software: application program

- An **application program** –
  software for a particular task or application.
  - E.g. Microsoft Word.



Coffee time:	Is an image editing program part of the **operating system** or an application program? What about a web browser?

Computer basics: data: files

- Stored **data** on **persistent storage** is organized into **file**s.
- These have a name to distinquish them from each other.
  - E.g. jpeg image stored in `my-mum.jpg`.
- Contents read/written via **operating system**.

Computer basics: data: files: text files

- A **text file** contains **data** stored as human readable **characters**.
  - E.g. `README.txt` that sometimes comes with **software**.
  - E.g. Source text for a document to be processed by the LaTeX document processing system (like this one!).
  - E.g. computer program **source code** files.

Computer basics: data: files: binary files

- A **binary file** contains **data** stored as **binary** (base 2) numbers.
- It is not human readable.
  - E.g. digital camera image stored as jpeg.
    - Look at it directly (not via an image viewer) and it looks like nonsense!
  - E.g. **machine code** instructions of a program.

Section 4

What is a program?

Aim

AIM: To look at what we mean by a computer program, particularly in Java, and how it is produced and processed.

What is a program?

- Simple view:
  - It comes on a **DVD ROM**.
  - Install it on your computer.
  - Then **run** or **execute** it.
    - It does something
    - and finally ends.
  - You can run it again and again.
- But first, a programmer must write the program
  - in a programming language, e.g. Java!
- S/he creates text expressing the program's meaning.
  - Stored in one or more **text files** – **source code files**.

Java tools: text editor

- A **text editor** allows us to type and edit **text files**.
  - E.g. `notepad` on Microsoft Windows,
  - but not `Microsoft Word` – that is a **word processor**.
- Text editor might be built in to an **integrated development environment**,
  - or be stand alone:
    - ∗ dozens of text editors suitable for Java programming.

What is a program?

- The **source code** contains instructions the programmer wants the computer to obey.
  - Essentially a sequence.
- These files are processed by another program – a **compiler**.
  - checks source code satisfies rules of the programming language.
  - produces version of program runnable by the computer.

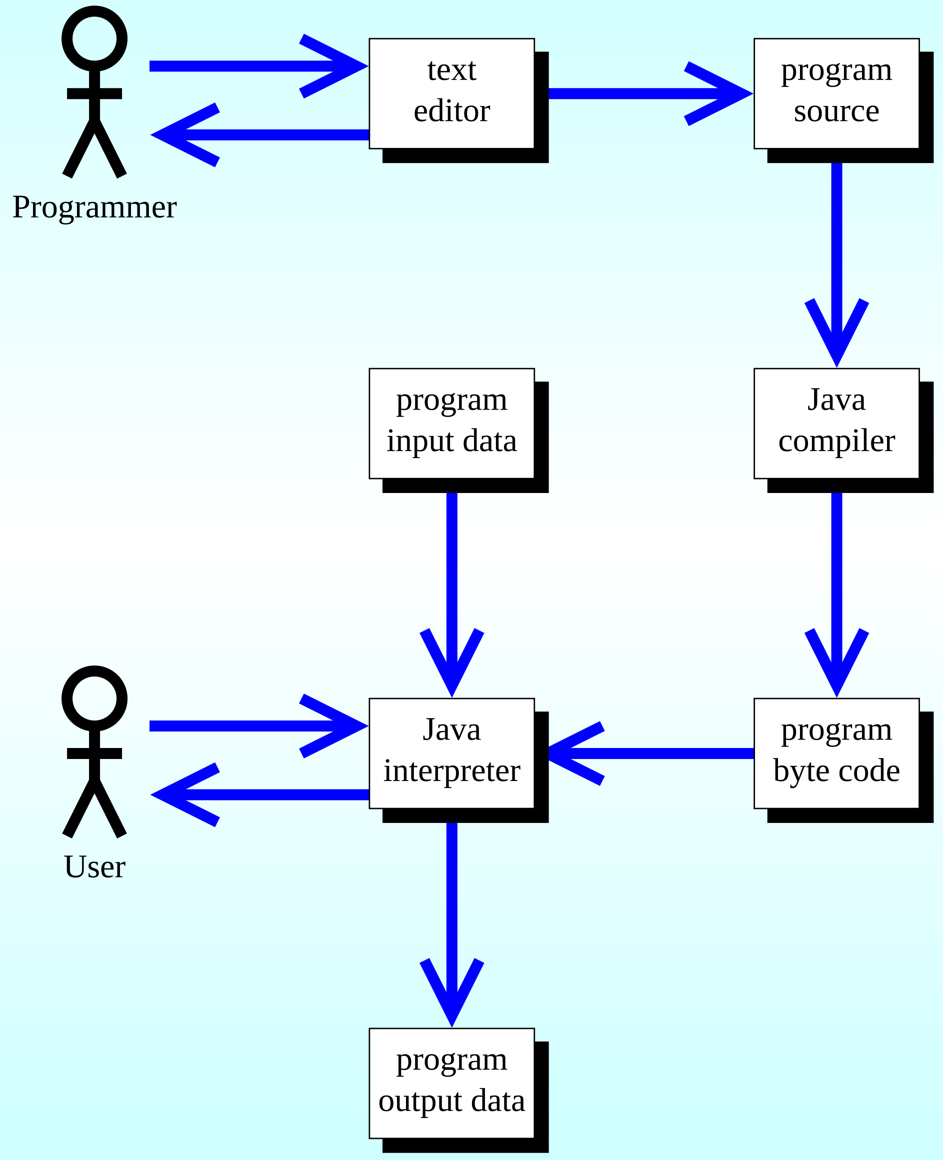
What is a program?

- Typically the **compile**d form is expressed in **machine code**.
  - the **low level language** for the CPU to execute directly.
- Modern programming languages are **high level language**s
  - each source program instruction is compiled (translated) into many machine code instructions.

What is a program?

- Different kinds of processors have different machine code languages.
  - E.g. a SUN UltraSPARC CPU cannot run code compiled for Intel X86.
  - A portability problem.
- Java gets around this using **byte code**
  - a CPU independent low level language.
- The compiler creates a byte code version from the Java source.
- This is run by another program – an **interpreter** or **virtual machine**.
  - It pretends to be a CPU with byte code as its `machine code' language.
- Interpreter is compiled for each kind of processor
  - every compiled Java program can be run everywhere!

What is a program?



Section 5

Operating environment

Aim

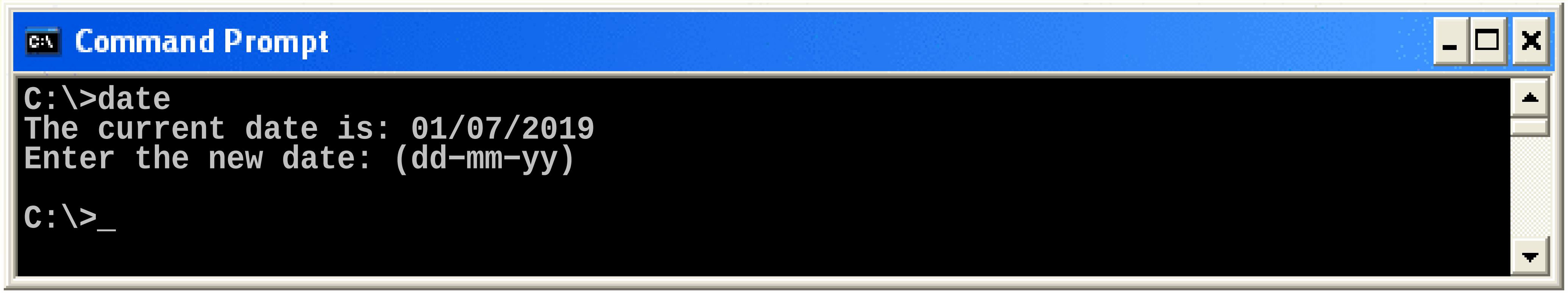
AIM: To explore the relationship between a program and the **operating system** it is **run** on. We see that programs are commands, with **command line arguments**, and meet **standard output**.

Operating environment: programs are commands

- When a program is **execute**d, its name is passed to the **operating system**
  - finds and loads that **file**
  - starts the program.
- May not be obvious with programs started from a menu or browser interface.

Operating environment

- E.g. on Microsoft Windows, running `date` in a **Command Prompt** window.



```
C:\>date
The current date is: 01/07/2019
Enter the new date: (dd-mm-yy)

C:\>_
```

- User has typed the command name `date` after the prompt.
- Windows then
  - finds the date program (actually `date.exe`)
  - loads it
  - runs it.

Operating environment

- The book uses a Unix (Linux) environment to run programs.
- It doesn't matter – programming concepts are independent of the **operating environment**.
- E.g. a similar program called `date` run in the Unix **shell**.

Console Input / Output

```
$ date
Mon Jul  1 19:12:00 BST 2019
$ _
```

Run

- Note:
  - assume prompt is a simple dollar ($)
  - what the user types is shown in **`bold face`**.

Operating environment: standard output

- Program **run**s can produce text results on their **standard output**.
- When run from **Command Prompt** or Unix **shell**:
  - output typically appears in that window.
- If run in some **integrated development environment**, browser, or menu:
  - output might appear in some pop-up box,
  - or special console window,
  - etc..

Operating environment

- Unix `date` program prints current date and time on its **standard output**.
- Various **command line arguments** can be used to alter its result.
- E.g. Make it print just date in day/month/year format
  or get output in custom form with our own words.

Console Input / Output

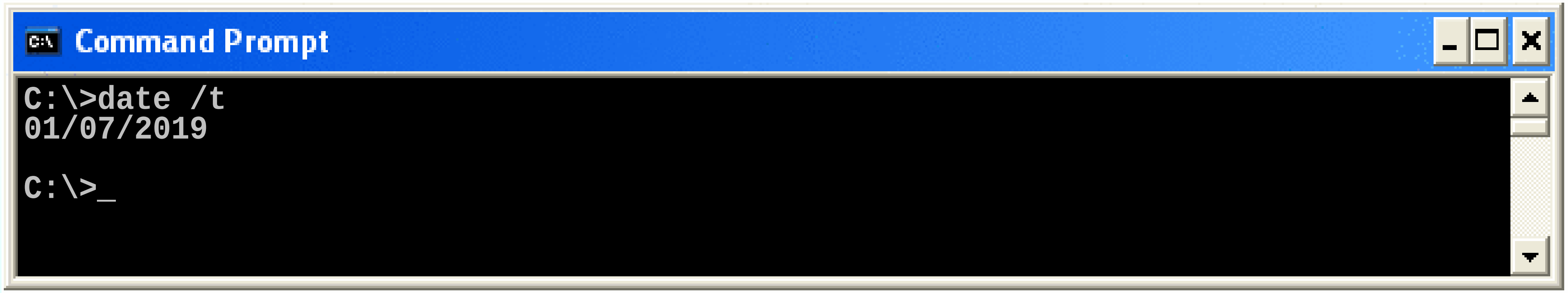
```
$ date "+%d/%m/%Y"
01/07/2019
$ date "+Today is %A, day %d of %B in the year %Y"
Today is Monday, day 01 of July in the year 2019
$ _
```

Run

- If using Unix
  - find out more about `date` via `man date`.

Operating environment

- Command line arguments are not specific to Unix.
  - E.g. Windows `date` with `/t` tells it not to prompt for new date.



```
C:\>date /t
01/07/2019

C:\>_
```

Operating environment: command line arguments

- Programs can be, and often are, given **command line argument**s to vary their behaviour.

Operating environment

- What happens when a user clicks on an icon representing a **text file** called `MyText.txt` in Windows?
  - A command line is created, consisting of (probably):
    `notepad "MyText.txt"`
  - This command line is then executed.
  - `notepad` knows which file to open from its argument.
- Verify this: look at `File Types` settings, via `Folder options...` on the `Tools` menu.

Section 6

Example:
Our first Java program

Aim

AIM: To show the mechanics of processing a finished Java source program so that it can be **run**, through to actually running it.

Our first Java program

- Our first example prints `Hello world!` on the **standard output**.
- Here is its **source code**.
  - Note: line numbers are not part of the **file**!

Our first Java program

Coffee time: We shall study this source code in detail in the next chapter, however for now you should try and guess which line of code is the instruction that makes the message appear on the standard output.

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

Our first Java program

- Source code is typed by programmer and saved in file `HelloWorld.java`.
- The book presents programs as though a stand alone text editor is being used
  - because the author believes that is best environment to get good grasp of things.
- We make the **compiler** process the saved file.
  - so is ready for **interpreter** to run it.

Java tools: javac compiler

- Java **compiler** is called `javac`.
- To **compile** program saved in `HelloWorld.java` we run

```
javac HelloWorld.java
```

- This produces **byte code** version in **file** `HelloWorld.class`
  - a **binary file** – not human readable.

Java tools: java interpreter

- The **interpreter** is called `java`.
- It runs the program named as its **command line argument**.
- To run `HelloWorld` program we type

```
java HelloWorld
```

- The **CPU** runs the interpreter or **virtual machine** `java`.
- `java` **execute**s the **byte code** program named as its first argument.
- Note: suffix `.java`
  - needed when compiling program,
  - not when **run**ning it.
- `java` finds byte code for `HelloWorld` in **file** `HelloWorld.class`
  - must have been previously produced by **compiler**.

Trying it

- Assume prompt from Unix **shell** is single $ followed by space.
- `ls` command with `-l` **command line argument** shows when a file was modified.

Console Input / Output

```
$ ls -l HelloWorld.java
-rw-------   1 jtl jtl 117 Jul 01 19:12 HelloWorld.java
$ _
```

Run

Trying it

- We **compile** the program.

Console Input / Output

```
$ javac HelloWorld.java
$ _
```

Run

- Now have extra **file**.

Console Input / Output

```
$ ls -l HelloWorld.*
-rw-------  1 jtl jtl 426 Jul 01 19:12 HelloWorld.class
-rw-------  1 jtl jtl 117 Jul 01 19:12 HelloWorld.java
$ _
```

Run

Trying it

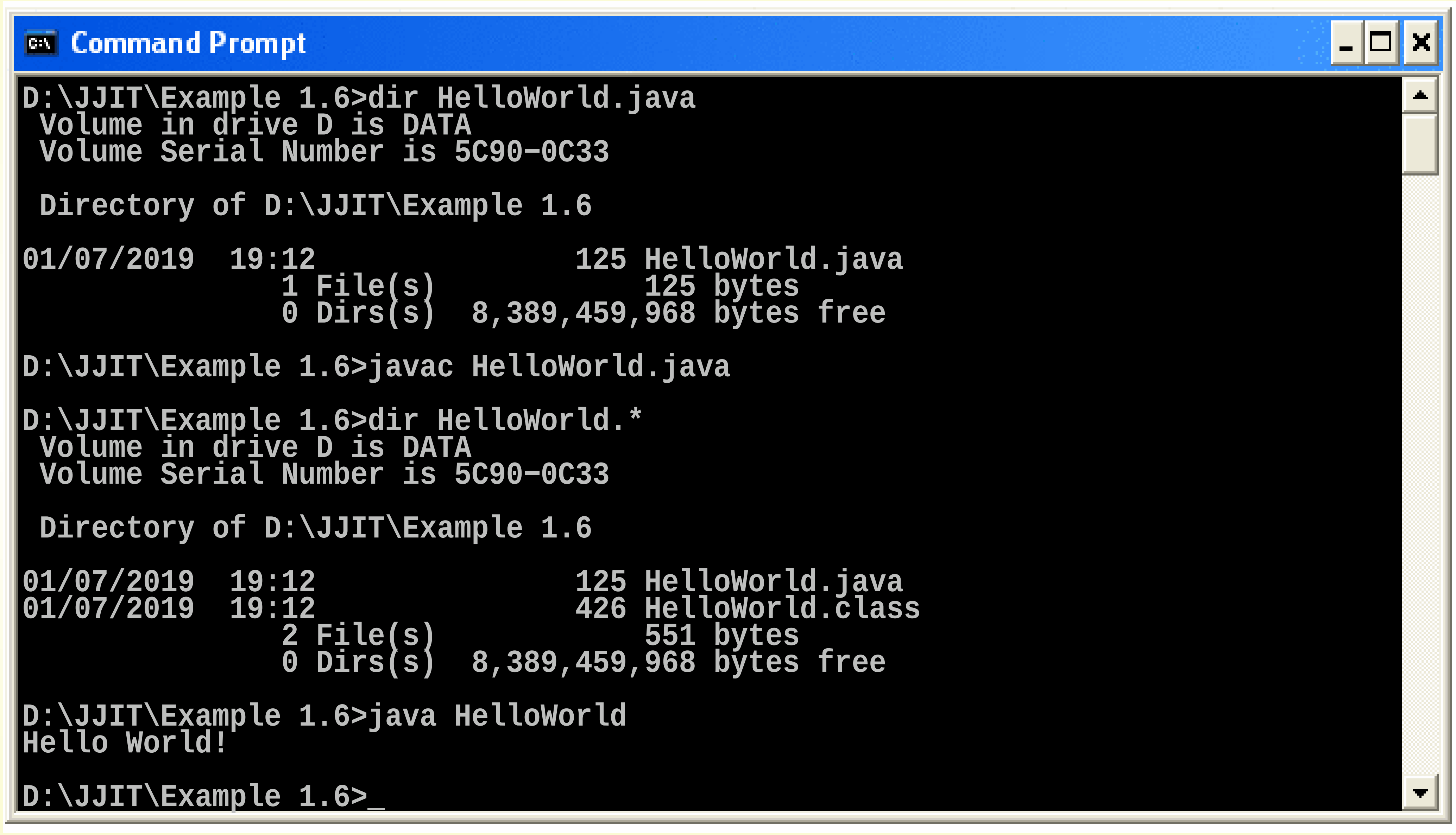
- Whenever run we see result on **standard output**.

Console Input / Output

```
$ java HelloWorld
Hello world!
$ _
```

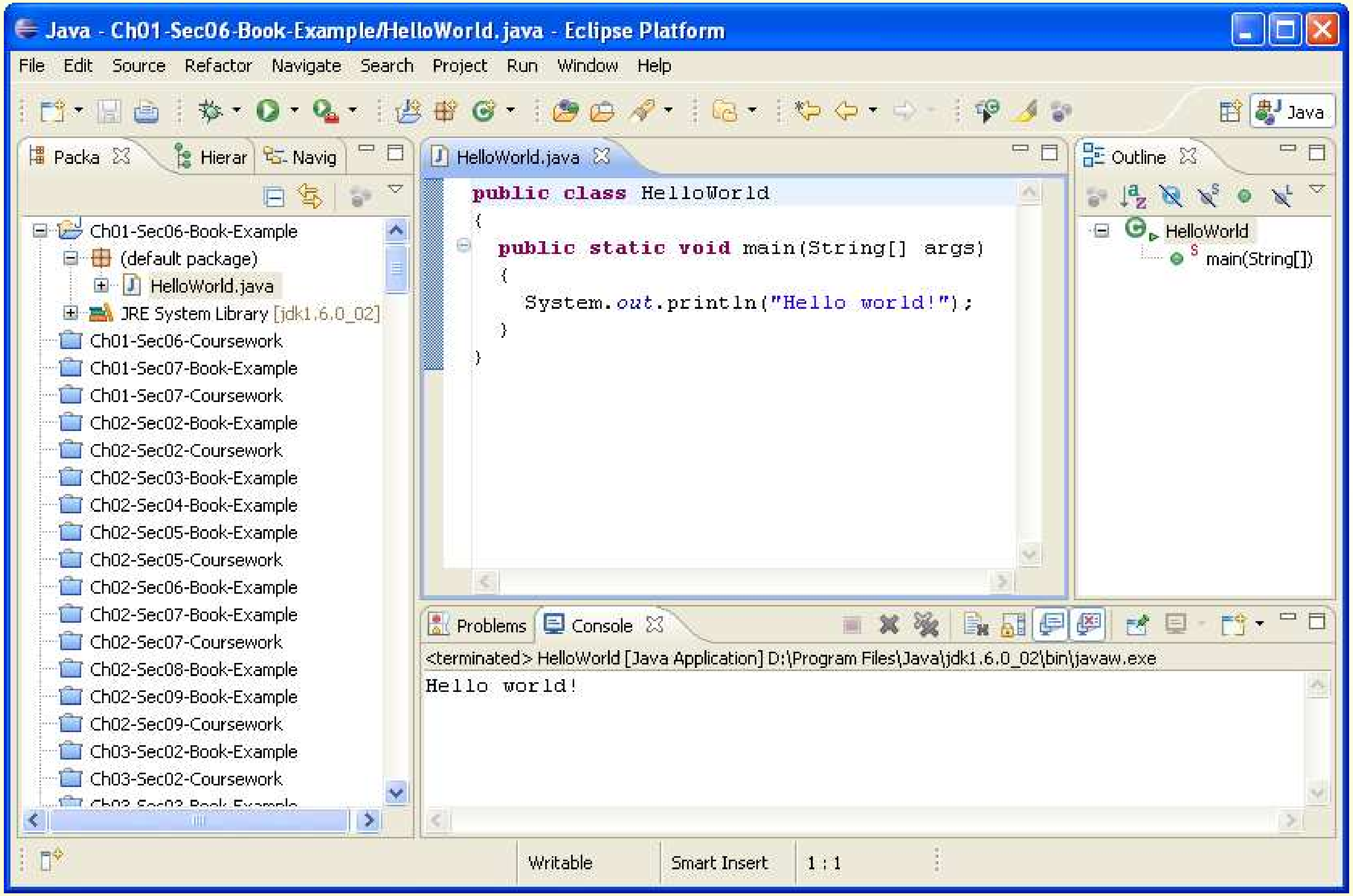
Run

Running under Microsoft Windows



```
D:\JJIT\Example 1.6>dir HelloWorld.java
 Volume in drive D is DATA
 Volume Serial Number is 5C90-0C33

 Directory of D:\JJIT\Example 1.6

01/07/2019  19:12               125 HelloWorld.java
               1 File(s)            125 bytes
               0 Dirs(s)  8,389,459,968 bytes free

D:\JJIT\Example 1.6>javac HelloWorld.java

D:\JJIT\Example 1.6>dir HelloWorld.*
 Volume in drive D is DATA
 Volume Serial Number is 5C90-0C33

 Directory of D:\JJIT\Example 1.6

01/07/2019  19:12               125 HelloWorld.java
01/07/2019  19:12               426 HelloWorld.class
               2 File(s)            551 bytes
               0 Dirs(s)  8,389,459,968 bytes free

D:\JJIT\Example 1.6>java HelloWorld
Hello World!

D:\JJIT\Example 1.6>_
```

Using an integrated development environment



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

```
<terminated> HelloWorld [Java Application] D:\Program Files\Java\jdk1.6.0_02\bin\javaw.exe
Hello world!
```

Using an integrated development environment

- If you use an **integrated development environment**
  - important to realize that beneath the convenience it is actually running `javac` and `java`.
  - The book keeps the fundamental steps of invoking them exposed.

Coursework: Compile and run `HelloWorld`

(Summary only)

To **compile** and **run** the `HelloWorld` program.

Section 7

Example:
Our second Java program

Aim

AIM: To reinforce the process of the **compile** and **run** cycle of a Java program.

Our second Java program

- Our second example says hello to all nine planets!
  - (Or goodbye. . . .)

Our second Java program

```java
public class HelloSolarSystem
{
  public static void main(String[] args)
  {
    System.out.println("Hello Mercury!");
    System.out.println("Hello Venus!");
    System.out.println("Hello Earth!");
    System.out.println("Hello Mars!");
    System.out.println("Hello Jupiter!");
    System.out.println("Hello Saturn!");
    System.out.println("Hello Uranus!");
    System.out.println("Hello Neptune!");
    System.out.println("Goodbye Pluto!");
  }
}
```

Trying it

Console Input / Output

```
$ ls -l HelloSolarSystem.java
-rw-------  1 jtl jtl 452 Jul 01 19:12 HelloSolarSystem.java
$ javac HelloSolarSystem.java
$ ls -l HelloSolarSystem.*
-rw-------  1 jtl jtl 687 Jul 01 19:12 HelloSolarSystem.class
-rw-------  1 jtl jtl 452 Jul 01 19:12 HelloSolarSystem.java
$ java HelloSolarSystem
Hello Mercury!
Hello Venus!
Hello Earth!
Hello Mars!
Hello Jupiter!
Hello Saturn!
Hello Uranus!
Hello Neptune!
Goodbye Pluto!
$ _
```

Run

Coursework: Compile and run `HelloSolarSystem`

(Summary only)

To **compile** and **run** the `HelloSolarSystem` program.

Concepts covered in this chapter

- Each book chapter ends with a list of concepts covered in it.
- Each concept has with it
  - a self-test question,
  - and a page reference to where it was covered.
- Please use these to check your understanding before we start the next chapter.