

FIT ICT Software Development

Lecture 1 (Introduction)

Lecturer : Charles Tyner

Module Aim:

The purpose of this unit is to provide learners with an understanding of how object orientated programming technologies are used in programming and systems development. Learners will also be able to design, create, implement and test programming solutions.

Learning & Assessment 1

Learning outcome

The learner will:

1. Understand the principles of object oriented programming

Assessment criteria

The learner can:

- 1.1 Discuss the **principles, characteristics and features** of objected oriented programming

Learning & Assessment 2

Learning outcome

The learner will:

2. Be able to design object oriented programming solutions

Assessment criteria

The learner can:

- 2.1 Identify the **objects and data and file structures** required to implement a given design
- 2.2 Design an **object oriented programming solution** to a given problem

Learning & Assessment 3

Learning outcome

The learner will:

3. Be able to implement object oriented programming solutions

Assessment criteria

The learner can:

3.1 Implement an objected oriented solution based on a prepared design

3.2 Define **relationships between objects** to implement design requirements

3.3 Implement object behaviours using **control structures** to meet the design algorithms

3.4 Make effective use of an Integrated Development Environment (IDE), including code and screen templates

Learning & Assessment 4

Learning outcome

The learner will:

- 4. Be able to test and document object oriented programming solutions

Assessment criteria

The learner can:

- 4.1 Critically review and test an object orientated programming solution
- 4.2 Analyse actual test results against expected results to identify discrepancies
- 4.3 Evaluate independent feedback on a developed object oriented programme solution and make recommendations for improvements
- 4.4 create onscreen help to assist the users of a computer program
- 4.5 create documentation for the support and maintenance of a computer program

Having successfully completed the module you will have:

- Designed and developed graphical Windows based applications programs
- Designed and developed Object Oriented programs.
- A thorough understanding of programming within the Event-driven paradigm
- An understanding and practical ability with object orientation in Java.
- Familiarity with programming graphical interface components and class libraries.

Syllabus

Introduction to the Development of Software

- Algorithms
- Problem statement
- Pseudo code
- Flowchart
- Selection/sequence/iteration
- Variables: a name of a location in memory
- Arithmetic, Operators and Precedence
- Equality & relational operators
- Good programming practices
- Software Engineering Issues and the need for quality
- Approaches to testing software
- Installing the Java and the development environment

Control Structures

- Controlling the flow of a Java program
- Techniques to design and achieve this control
- Getting information into & out of a program
- IF and IF/ELSE
- Nested Ifs & ELSE statements
- Compound statements
- WHILE
- Counter control
- Sentinel
- FOR
- SWITCH
- DO_WHILE
- BREAK
- CONTINUE

Syllabus

Basic Graphical User Interface Components

- Understand the principles behind designing good graphical user interfaces.
- Build graphical user interfaces.
- Event Handlers and Event Listeners
- Create and manipulate text fields.
- Create and manipulate labels.
- Create and manipulate buttons.
- Create and manipulate panels.
- Create and manipulate lists.
- Understand mouse events and keyboard events.

Arrays

- Introduce the array data structure
- Use of arrays to store, sort, and search lists and tables of values
- Declare an array, initialise an array, and refer to individual array items
- Reference and reference parameters.
- To be able to pass arrays into methods
- Understand basic sorting techniques.
- Linear and binary search of an array
- Be able to declare and manipulate multiple subscript arrays.

Syllabus

Object-Oriented Programming

- Understand encapsulation and data hiding.
- Simple OO analysis and design
- Encapsulation, Inheritance and Polymorphism
- Understand data abstraction and Abstract Data Types.
- Object type or Class and its methods
- Using constructors
- Understand class variables and class methods.
- Understand class scope.
- Composition – objects as instance variables of other classes.
- The benefits of the object-oriented paradigm and software reusability.

Strings and Text Handling

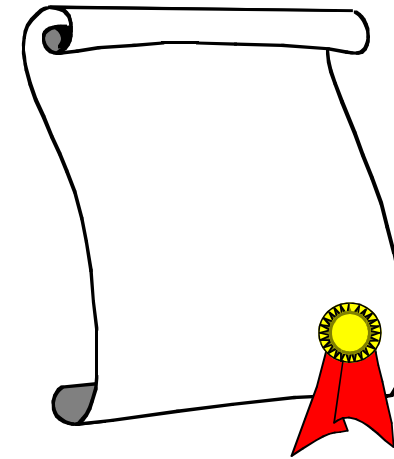
- Fundamentals of **Characters** and **Strings**.
- Create & manipulate character string objects
- Locating **Characters** and **Substrings** in **Strings**
- Extracting **Substrings** from **Strings**.
- Concatenating **Strings**.
- Miscellaneous **String** Methods

Syllabus

File Handling

- Creating files
- Input and output to sequential files
- Creating random access files
- Input and output to random access files
- Reading random access files sequentially

Your Success Depends on You



- Reading the notes and working through code on a weekly basis is **essential**
- Ensure you understand the content of the block of notes **every** week
- Be organised with notes and programs (type up code from lectures first and get it running)
- Be efficient - get through the practical work week-by-week

This Week

- History of Java
- Our 1st Java Program - Simple Output to the Keyboard
- Using Textpad to Write programs
- Compile/Execute a program
- Editor Techniques
- How a Program Runs
- Output to the screen
- Taking input from the keyboard
 - Declaring Variables
 - Assigning Values to Variables: Assignment Statement
 - A note on Reserved Words
 - Create **constant** values with the keyword **final**
 - Understand **arithmetic operators**
 - Understand **expressions**
- Introduce Program Design:
Algorithms, Pseudocode and Flowcharts

Introduction to Programming

In order to enable a computer to carry out a task, it must be told exactly what to do and given precise instructions as to how to do it.

The series of steps involved in carrying out a task to solve a particular problem is called an **algorithm**.

In order for the computer to carry out this series of steps the steps must be expressed in some form that the computer is able to understand..... i.e. in a language that the computer can read.

An **algorithm** expressed in such a form is called a **program**.

The computer is then said to **execute** this program.

A program is written in a programming language.

Programming Languages

A program is written by a developer in a special computer language. For example:

- C++;
- Java;
- Python

It is then translated by a special program (a **compiler**) into:

- machine code, or
- Java byte code.

History of Programming Languages

Assembly language (low-level language)

High-level languages (third generation languages or 3GLs).

Examples include

- C
- COBOL
- Pascal

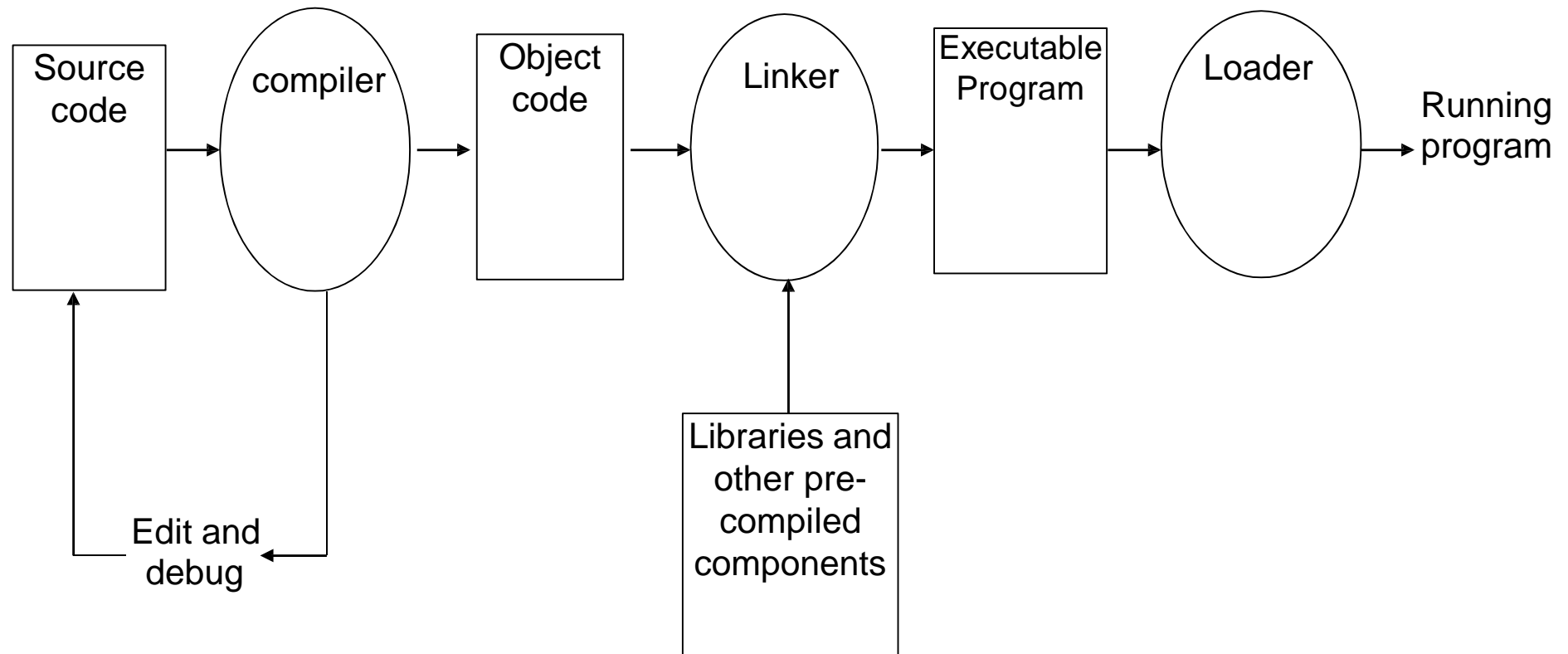
Object-oriented programming languages. Examples include

- C++;
- Java.

History of Java

- In 1991 Sun Microsystems funded an internal corporate research project code-named **Green** in 1991.
- Project resulted in the development of a C and C++ based language.
- This language became known as Java when a group of Sun people visited a local coffee shop.
- By sheer good fortune, the World Wide Web exploded in popularity in 1993 and Sun people saw the immediate potential of using Java to create Web Pages.
- Sun formally announced Java at conference in 1995.
- Java is now used to create web pages, to develop large-scale enterprise applications, enhance the functionality of WWW servers, to provide applications for consumer devices, such as cell phones.

The traditional way of compiling programs



Compilers and Interpreters

Compilers

- operate on the entire program;
- provide a permanent binary file which may be **executed** (or **run**).

Interpreters

- translate and execute the program one line at a time.

With Java

- the processes of compilation and interpretation are combined.

The Java Development Process

- Older compiled programs (Cobol) only suitable for a particular type of computer, eg PC. These programs will not run on an Apple Mac
- Advent of WWW needed to download a program and run it on our computer irrespective of whether our machine is a PC or Apple Mac.
- The need arose for a language that is **platform-independent** (runs on any platform or Operating System (OS) e.g. Microsoft Windows, Apple MacOSX etc....)

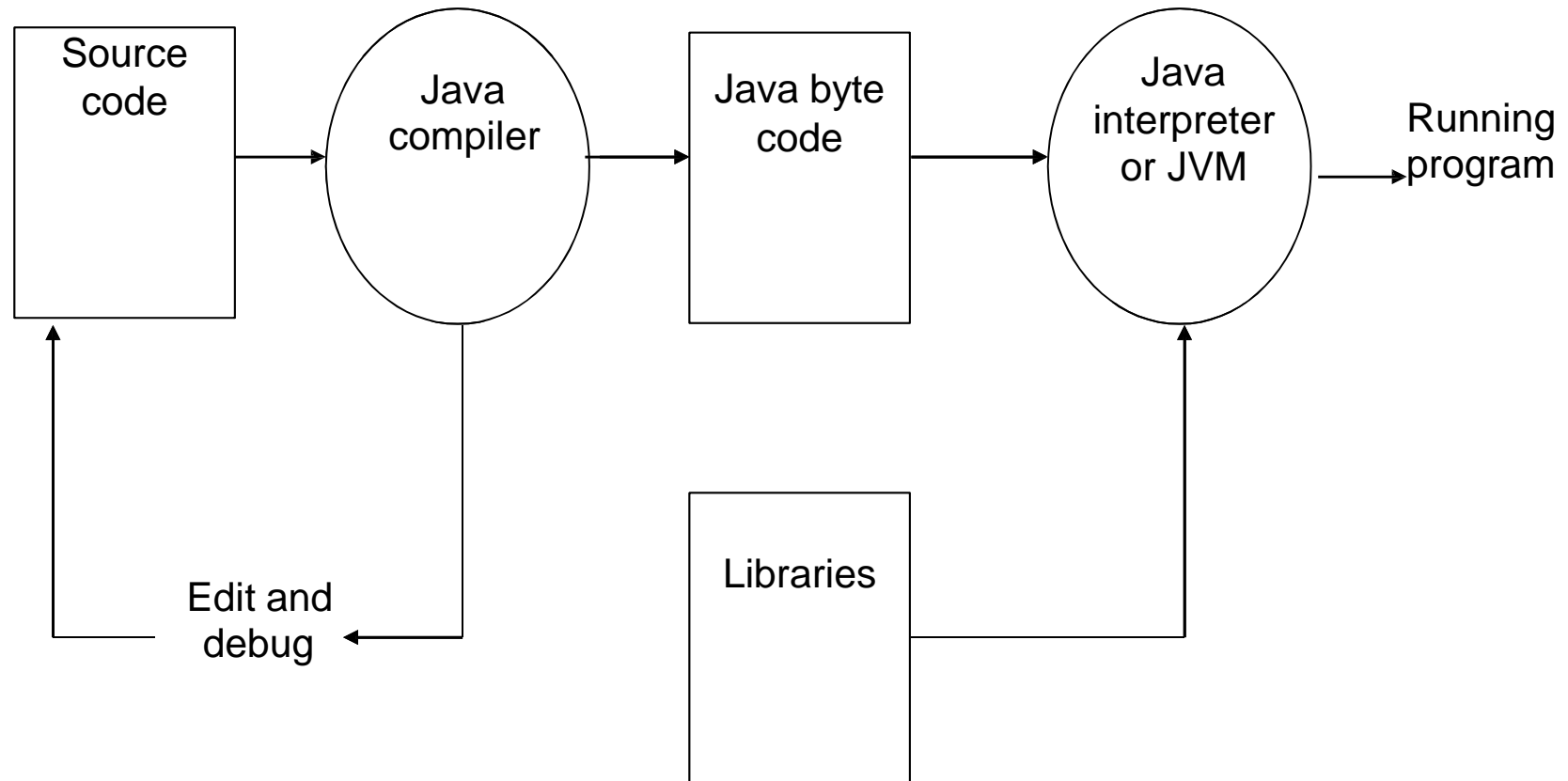
The Java Development Process

- JAVA is a **platform-independent language** (when we use Java to create software that software will run on any platform or operating system)
- JAVA was designed to work within the World Wide Web through a piece of software called browser, eg Safari, Firefox, Opera, Chrome or Internet Explorer.

The Java Development Process (Contd.)

- Java programs that run on the web are called **applets**.
- Inside modern browser, there is a special program called **Java Virtual Machine (JVM)**.
- JVM is able to run a Java program for the particular computer on which it is running.
- Java compilers translate the program into **Java byte code**.
- These instructions are then interpreted by the JVM for that particular machine.

Compiling, interpreting and running Java Programs



Program 1.1

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


Analysis of 'Hello World' Program

1st Line

class Hello

- Java is an object-oriented programming language
- This requires program to be written in separate unit called **classes**.
- In this example class **Hello** was created.
Hello is the **program name**.
- Everything in the class has to be contained between 2 curly brackets that look like { }.
This tells the compiler where the class begins and ends.

2nd Line

public static void main(String[] args)

- Applications (as opposed to applets) must always contain a class with a method called **main**.
- Program instructions are written in curly brackets that show us wher instructions begin and end.

Analysis of 'Hello World' Program (Contd.)

3rd Line

```
System.out.print("Hello world");
```

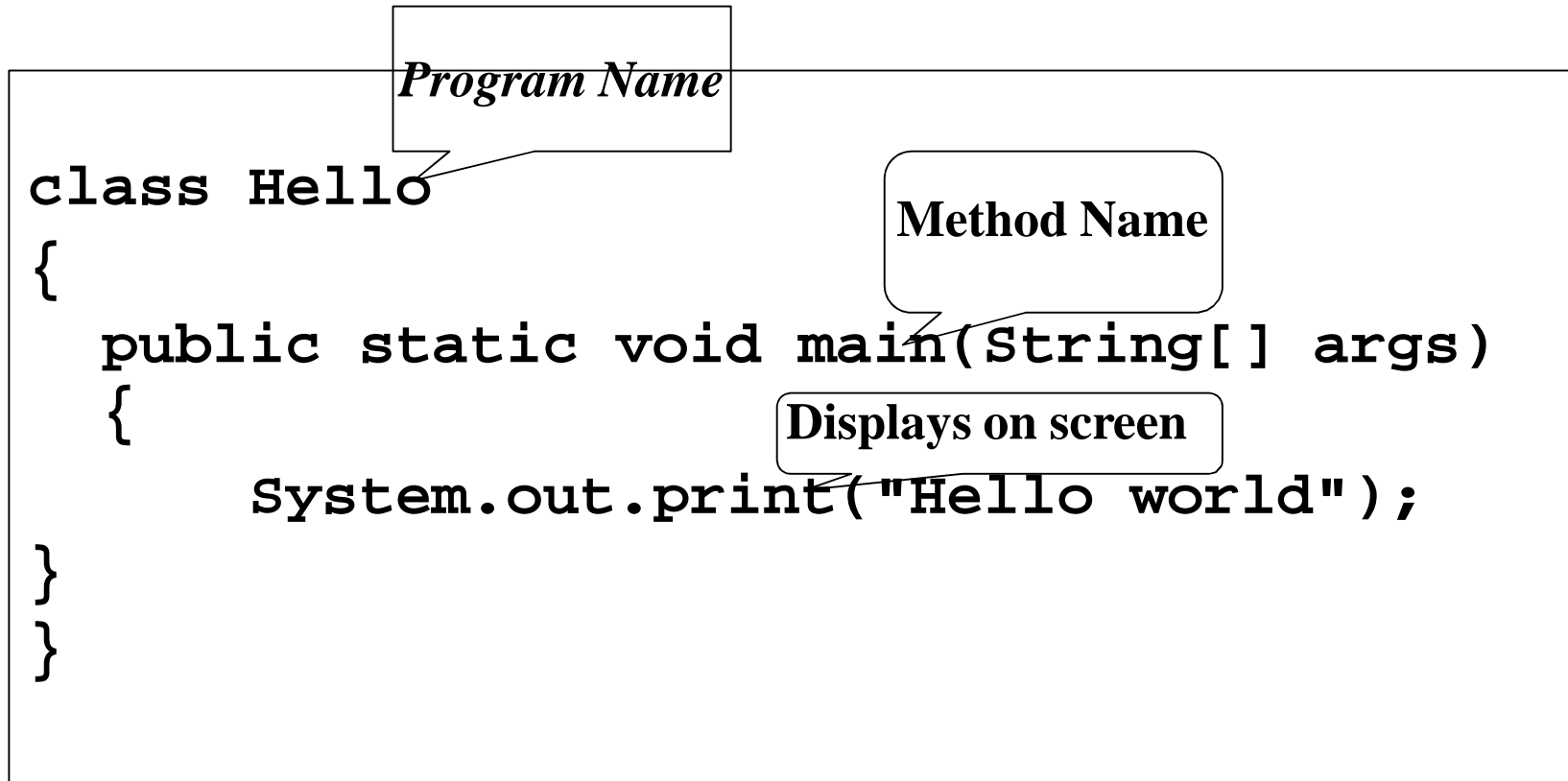
- We use System.out.print to get stuff printed to the screen.
- We put whatever we want to display in brackets.
- Notice semi-colon at end of statement.

Every Java instruction has to end with a semi-colon.

NOTE - JAVA IS CASE SENSITIVE

- This implies that **class** and **Class** are not the same
- If you use the word Class, you will get compile errors.

Summary - Program 1.1



To create a program – we use an Editor

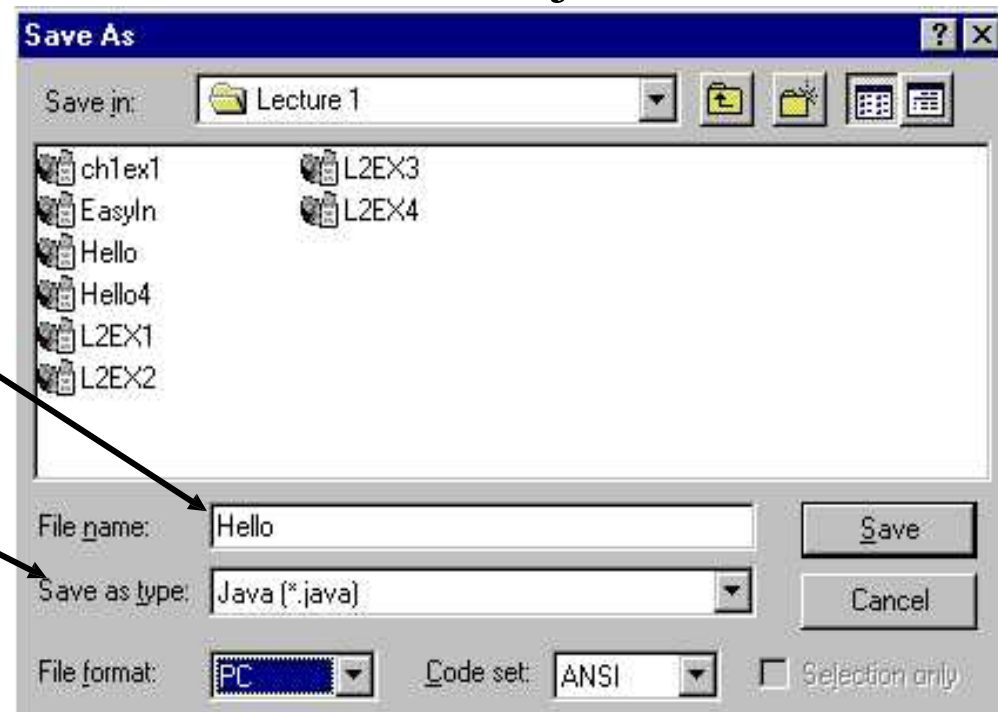
- **Textpad** to enter Java programs.
- You can execute this from **Start/Programs/Textpad**

To save the program

- Once you have created the program you should save it as
<classname>.java

In our last example program was saved as Hello.java

- Command is **File/Save As**
- Enter Program name
in File Name
- Ensure Save as type
is Java



Compile Program

- Command in Textpad is **Tools/Compile** or **Ctrl 1**.
- If the program compiles without error, a **.class** version will be created in the directory where your program is stored. For example if I compile the program Hello.java, Hello.class will be created in the directory.

Execute Program(Application)

- Command is **Tools/Run Java Application** or **Ctrl 2**.

Compile Errors

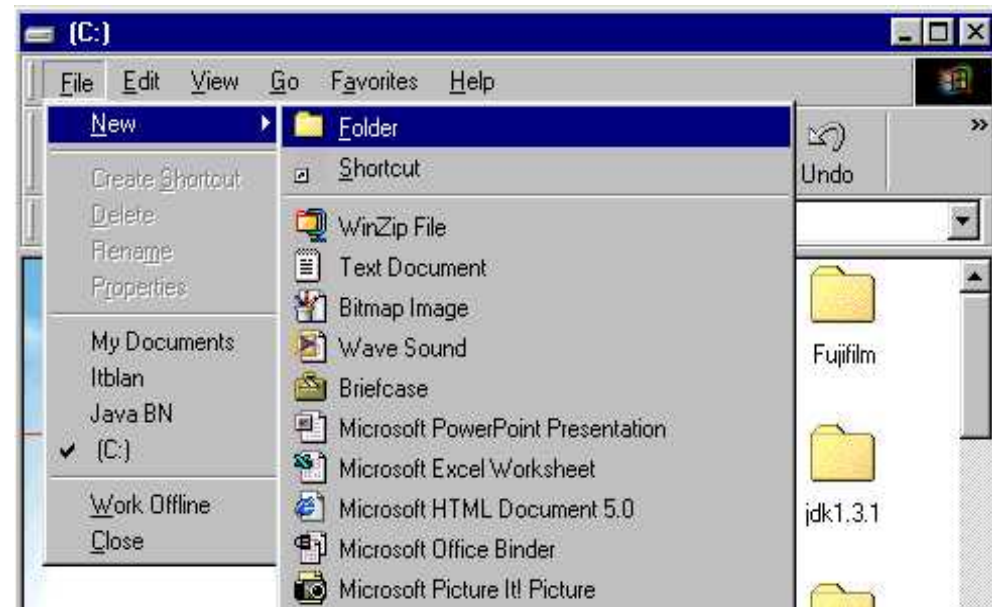
- Compiling means you are attempting to convert your source code to object code, and the compiler may find errors. Successful compilation from a .java file creates a .class file.
- To minimise compile errors – take care when keying in code.
- Take care with:
 - File name must be the same as the class name
 - uppercase and lowercase
 - spacing
 - types of brackets, open & close () { }
 - semi-colon at the end of instruction lines

Saving files on Windows

- Some Windows text editors, including Notepad, add a three letter ".txt" extension to all the files they save without telling you.
- Thus you can unexpectedly end up with a file called "HelloWorld.java.txt."
- This is wrong and will not compile.
- Enclose the filename in double quotes in the Save dialog box to make editor save the file with exactly the name you want.

Creating Folders

- Before you begin your lab session today..On your named drive create a folder called Java. The command is **File/New/Folder**.

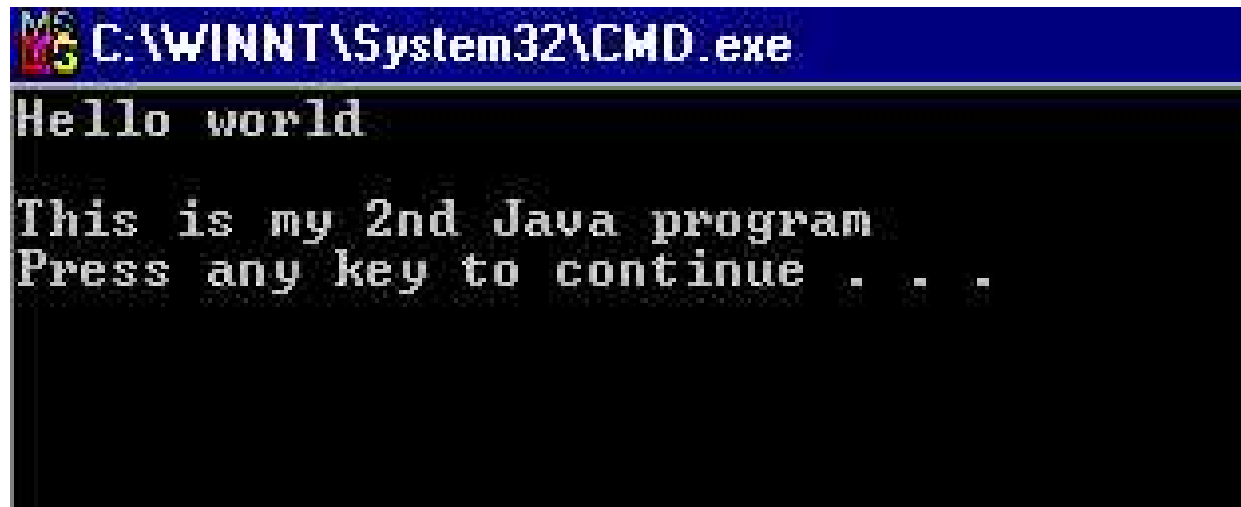


- Within the Java folder create 9 sub-folders as follows:
 - Lect1
 - Lect2
 - Lect3 etc...

Lecture 1 - Program Example 2

```
class Hello2
{
    public static void main(String[] args)
    {
        System.out.println("Hello world");
        System.out.println("");
        System.out.println("This is my 2nd Java program");
    }
}
```

Result



The screenshot shows a Windows command prompt window with a blue title bar that reads "MS-DOS C:\WINNT\System32\CMD.exe". The black command prompt area displays the output of the Java program: "Hello world" on the first line, an empty line on the second, and "This is my 2nd Java program" on the third. Below the output, the text "Press any key to continue . . ." is visible, indicating the program is waiting for a key press to terminate.

Editor Techniques

Editor Techniques

- **To select a line of code for copying**
Move the cursor to the left margin
Mouse Pointer changes from I to arrow
Click on Mouse and whole line selected
- **To select multiple lines of code for copying**
As above, click and drag on mouse
- **Shortcut Keys**

Select Text	
Ctrl and X	Cut
Ctrl and C	Copy
Re-Position Cursor	
Ctrl and V	Paste

How a Program Runs

How a Program Runs

- Computer programs consists of **one or more instructions** that will be executed in a given **sequence**.
- All programs developed in this lecture will execute top-down **in order** in which they are written.
- Each instruction will **terminate** with a **semi-colon**.

Note: Some lines of code however, are not instructions and do not require semi-colons.

For Example:

```
class declaration  
main method
```

Simple Output

Writing Messages to the Screen

- Screen is a rectangle window that can display up to 25 lines of text.
- Each line is eighty characters in length.
- To write a message to the screen we need:

to **define** what a **message**
is

an **instruction** to display a given
message

Writing Messages to the Screen

- A message consists of a sequence of characters enclosed by double quote marks.
 - Examples
 - “I love rock music”
 - “Programming is cool”
- To display a message we use the instruction **System.out.println(m1)** where **m1** is the **message**.
- For example to display the message “**I love rock music**”
System.out.println(“I love rock music”);
- To display the message “**Programming is cool**”
System.out.println(“Programming is cool”);

Reminder - Each instruction is terminated by semi-colon. If omitted will result in syntax error.

Program – Lecture 1 Example 3

```
class L1EX3
{
    public static void main(String[] args)
    {
        System.out.println("I love rock music");
        System.out.println("Programming is cool");
    }
}
```

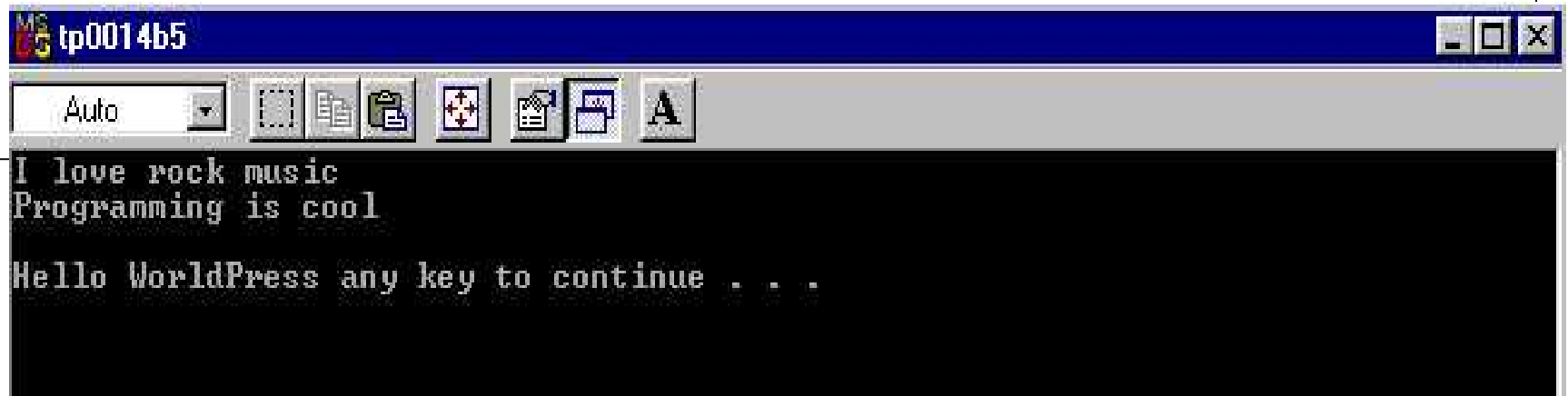
Controlling the Cursor

- Current position of cursor on the screen determines the starting print position for the next message.
- There are 2 ways to display message to screen:
 - 1st way - **System.out.print(m1)**
In this example the message will be printed on the screen and the cursor will **remain on same line**
 - 2nd way - **System.out.println(m1)**
In this example the message will be printed on the screen and the **cursor will move** to the **next line**.
- To print a blank line:
System.out.println("");

Controlling the Cursor Program Example

Program- Lecture 1 Example 4

```
class L1EX4
{
    public static void main(String[] args)
    {
        System.out.println("I love rock music");
        System.out.println("Programming is cool");
        System.out.println(" ");
        System.out.print("Hello World");
    }
}
```



Output in Java - Concatenation

Two strings can be joined together with the plus symbol (+), known as the **concatenation operator**.

```
System.out.println("Hello " + "world");
```

Outputting values on the screen

Values and expressions can also be printed on the screen using these output commands.

for example

30 people visiting the cinema, each charged an entrance fee of 7.50, the total cost of tickets could be displayed as follows:

```
System.out.println("cost = " + (30*7.5) );
```

Declaring Variables

Simple data types in Java

The types of value used within a program are referred to as **data types**.

price of a cinema ticket : **real number**

how many tickets sold : **integer**

In Java there are a few simple data types that programmers can use.

Often referred to as the **scalar types**

The scalar types of Java

<u>Java type</u>	<u>Allows for</u>	<u>Range of values</u>
byte	very small integers	-128 to 127
short	small integers	-32768 to 32767
int	big integers	-2147483648 to 2147483647
long	very big integers	-9223372036854775808 to 9223372036854775807
float	real numbers	+/- $1.4 * 10^{-45}$ to $3.4 * 10^{38}$
double	very big real numbers	+/- $4.9 * 10^{-324}$ to $1.8 * 10^{308}$
char	characters	Unicode character set
boolean	true or false	not applicable

Declaring variables in Java

- The procedure of creating **named locations** in computer's memory that will contain values while a program is running;
- these **named locations** are called **variables** because their values are allowed to *vary* over the life of the program.

To **create a variable in your program you must do 2 things:**

- give that variable a **name** (of your choice...something relevant);
- decide which **data type** in the language best reflects the kind of values you wish to store in the variable.

Naming variables

You can choose any name for variables as long as

- the name is **not already a word** in the Java language (such as **class**, **void**);
- the name has **no spaces** in it;
- the name does not include operators such as + and -;
- the name starts either **with a letter**, an underscore (_), or a dollar sign (\$).

The convention in Java programs is to begin the name of a variable with a **lowercase letter**.

Choosing a suitable data type

(Refer to slide 49)

Four Java types can be used to hold integers (**byte**, **short**, **int** and **long**).

Two Java types that can be use to hold real numbers (**double** and **float**).

The difference among these types is the range of values that they can keep, however

- the **int** type is often chosen to store integers
- the **double** type often chosen to store real numbers

The data type char is used to hold characters.

Once name and type decided upon, the variable is declared as follows:

dataType variableName ;

Declaring a variable: an example

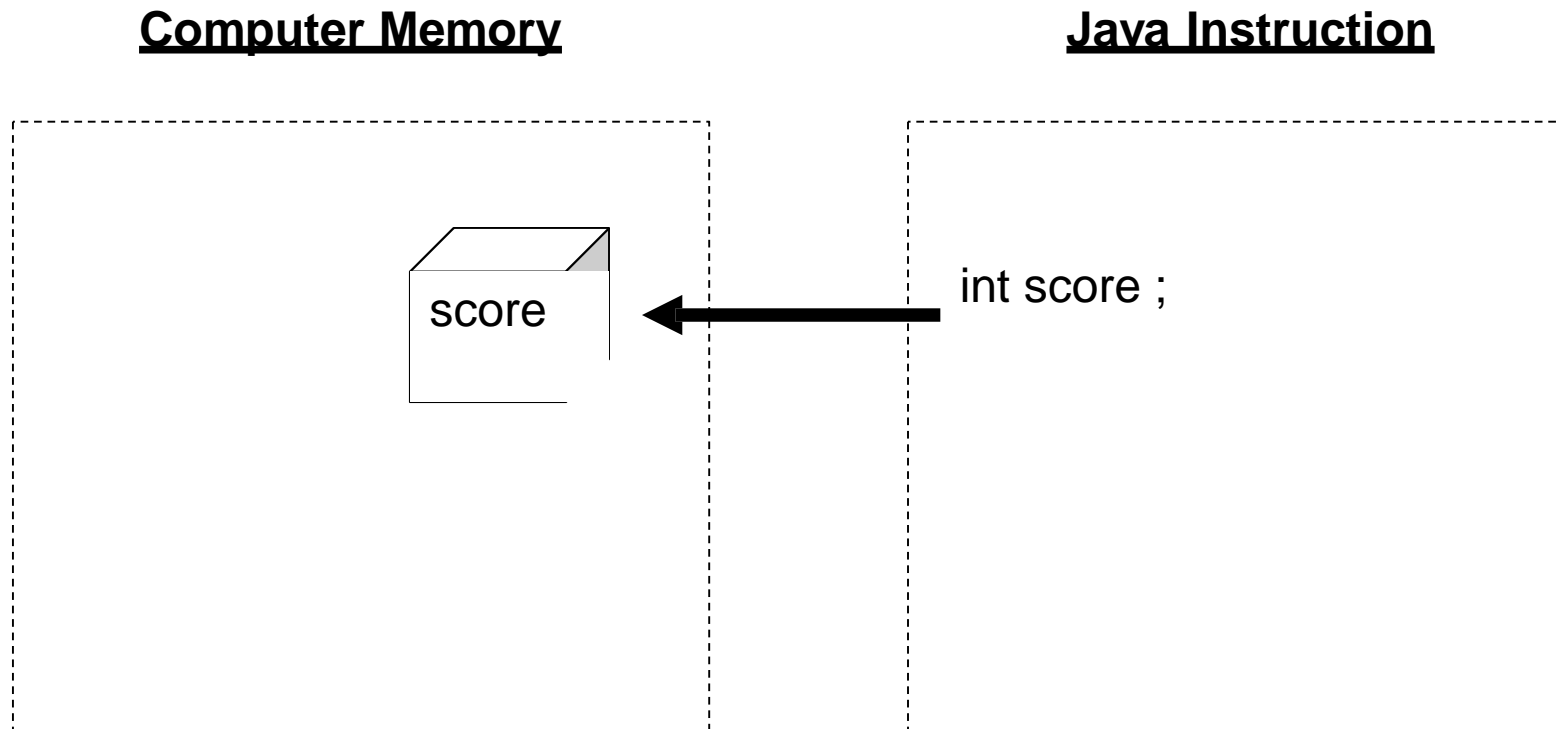
Let's create a variable to keep a player's score in a computer game.

a score will
always be a
whole number

good
meaningful
name

int score ;

The effect of declaring a variable in Java



Declaring many variables

Assume that the player of a game can choose a difficulty level (A, B, or C).

```
int score; // to hold score
```

```
char level; // to hold difficulty level
```

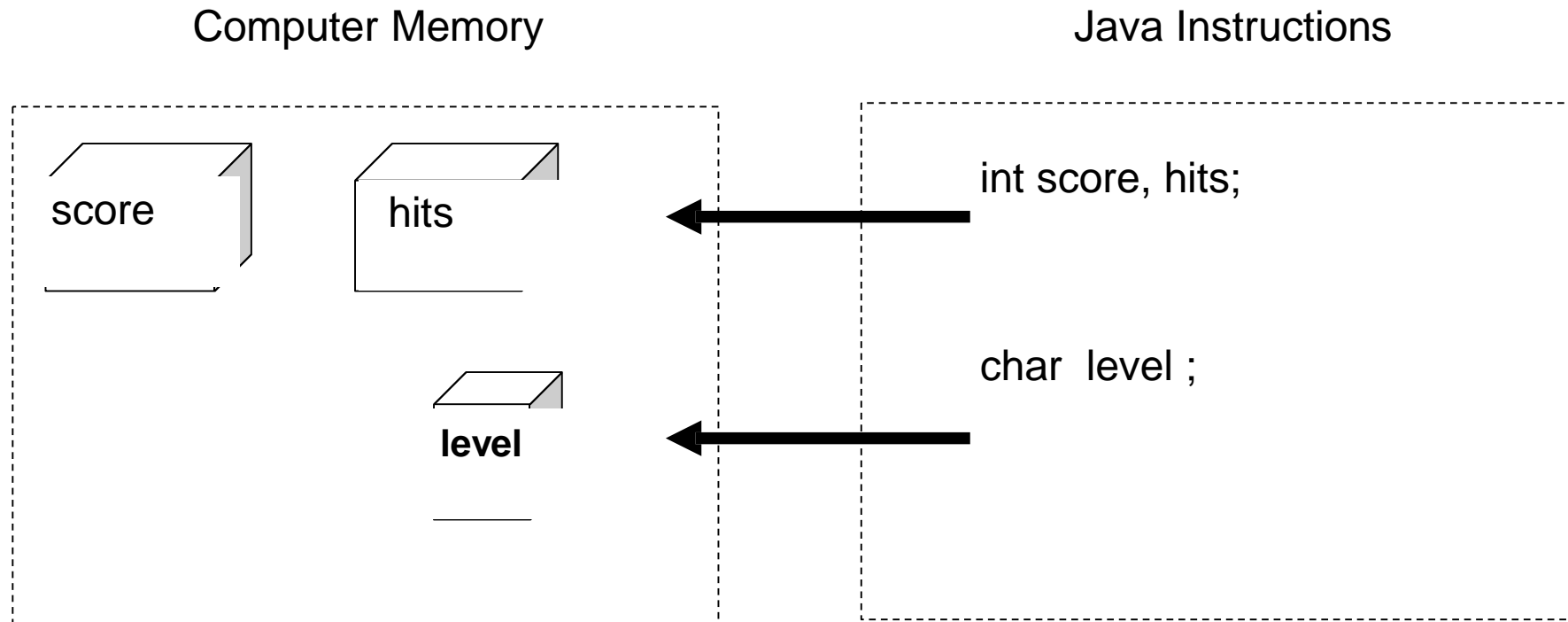
Declaring variables of the same type

Several variables can be declared on a **single line** if they are **all of the same type**.

Assume that there are ghosts in the house that hit out at the player; the number of times a player gets hit by a ghost can also be recorded.

```
int score, hits; // both the same type  
char level ;      // different type
```

The effect of declaring many variables in Java



Assignments in Java

Assignments in Java

Assignments allow **values to be placed inside variables.**

Written in Java with the use of the equality symbol (=), known as **the assignment operator.**

Simple assignments take the following form:

variableName = value;

```
score = 0;
```

Initializing variables

You may **combine** the **assignment statement** with a **variable declaration** to put an initial value into a variable as follows:

```
int score = 0;
```

Note, the following declaration will **not** compile in Java:

```
int score = 2.5 ;
```

This will not compile...**WHY???**

Because 2.5 is a **double** value and we declared it as an integer

Putting values into character variables

When assigning a value to a character variable, you must **enclose the value in single quotes**.

for example

set the initial difficulty level to A

```
char level = 'A';
```

Re-assigning variables

Remember: you need to declare a variable only once.

You can then assign to it as many times as you like.

```
char level = 'A';  
level = 'B';
```

(when declared **level** contained the letter A...then that was overwritten and now level contains B)

Re-assigning variables

Whenever a value is placed in a memory location, this value replaces the previous value in that location.

The previous value is therefore destroyed!

Input from the Keyboard

Next
Lecture!!!!