Java

Programming fundamentals: Introduction to Java

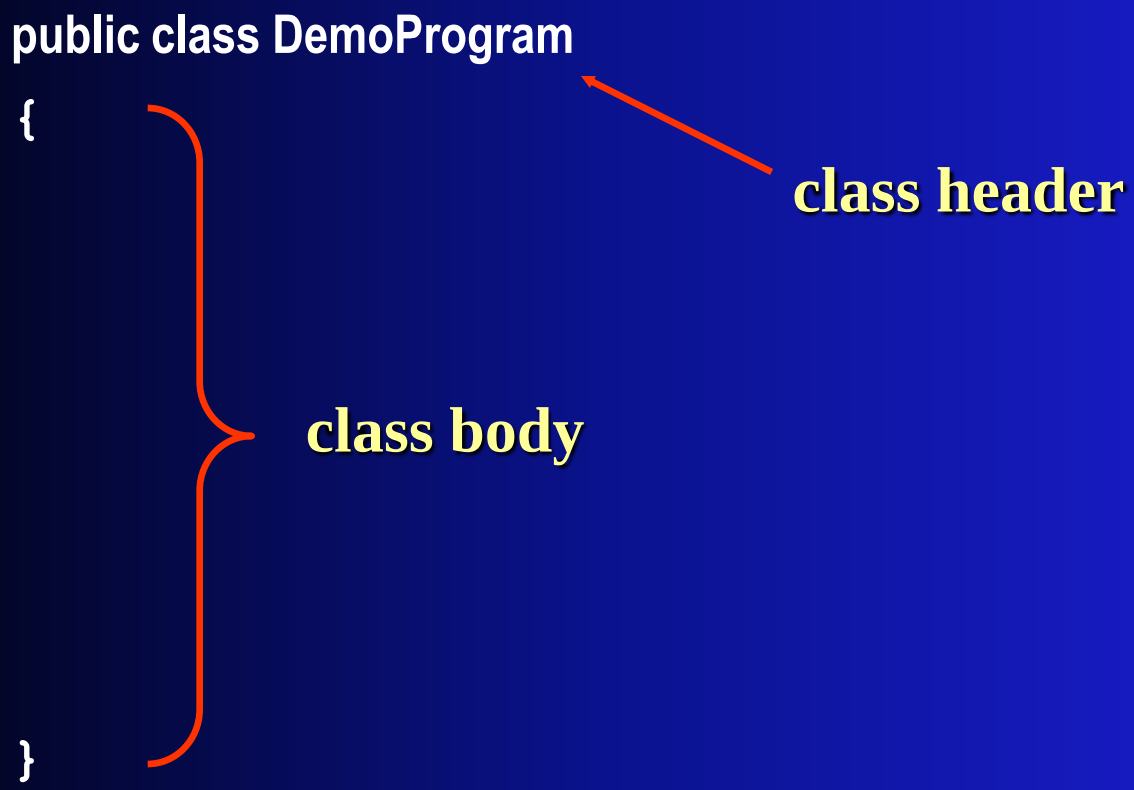
Experience is a wonderful thing – it allows us to recognise a mistake when we make it again

Lecture objectives

To be able to understand the following fundamental concepts of the Java programming language:

- The structure of a Java program
- Printing program output
- Program readability and layout
- The declaration and use of variables
- Assignment
- Primitive data
- Arithmetic expressions and operator precedence

A java program is made up of one or more *classes*



A class contains *methods*

An application always has a *main* method

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

        }
}
```

method body

method header

A method contains *statements*

A statement is a single command
ending with a semicolon “;”

```
public class QuoteWatson
{
     public static void main (String[] args)
    {
        System.out.println (“A quote by Thomas Watson, Chairman IBM, 1943:”);

        System.out.println (“I think there is a world market for may be 5 computers.”);
    }
}
```

A quote by Thomas Watson, Chairman IBM, 1943:

I think there is a world market for may be 5 computers.

Program readability

- Programs are not only read by compilers
  - They need to be read by other programmers, who will need to change the code over time
- It is critical that your program can be easily understood
- Some important techniques:
  - Comments
  - White space
  - Indentation
  - Placement of curly braces
  - Naming of identifiers

Comments

- *Comments* are an important part of every program
  - are purely for documentation purposes
  - ie they are ignored by the compiler
  - can be used anywhere in a program

- Essential comments are:
  - Program header
    - Program name, author, date written, program description
  - Statement description
    - To describe a particular statement eg a mathematical formula that may not be clear to someone reading the program

Comments

- Space precludes using adequate comments in lecture examples
- Java comments can take two main forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks        */
```

```
//*******************************************************************************
//  QuoteLordKelvin.java                    Author: J. Terry.
//  Date:  01/07/2002
//  Demonstrates the basic structure of a Java application
//             note the curly braces are aligned
//              note the groups in side the curly braces are indented  a few spaces
//*******************************************************************************

public class QuoteLordKelvin
{
     public static void main (String[] args)
     {
          System.out.println ("A quote by the President of the Royal Society 1896");

          System.out.println ("Heavier than air flying machines are impossible.");
     }
}
```

```
A quote by the President of the Royal Society 1896
Heavier than air flying machines are impossible.
```

White Space

- Spaces and blank lines are collectively called *white space*
  - it separates parts of a program
  - it enhances program readability

Alternative curly brace Placement

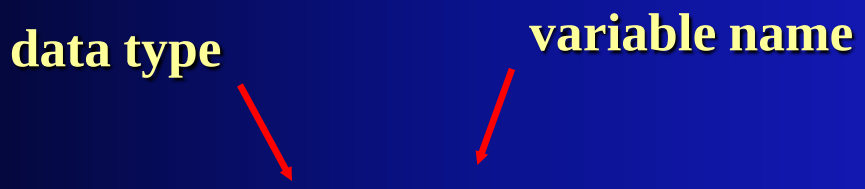
- Put each left brace at the end of a line.
- The matching right brace is lined up with the first character on that line:

```
public class JavaRules {
    public static void main(String[] args) {
        System.out.println("Java rules!");
    }
}
```

- This method is in common usage but we will not use it

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared*, specifying the variable's name and the type of information that will be held in it




```
int count, temp, result;
```

- **Multiple variables of one type can be declared separated by commas**
- **End declaration with semi-colon**

Initialising Variables

- ***Initialising*** a variable means to assign a value to the variable for the first time.

```
int total;
total = 0;
```

- Variables can be initialised at the time they're declared

```
int total = 0;
```

- More than one variable of the same type can be declared and initialised together

```
int rate = 0, value = 100;
```

```
//********************************************************
//  PianoKeys.java       Author: Lewis and Loftus
//
//  Demonstrates the declaration, initialization, and use of an
//  integer variable. Also  demonstrates  concatenation
//********************************************************

public class PianoKeys
{
    public static void main (String[] args)
    {
        int keys = 88;

        System.out.println ("A piano has " + keys + " keys.");
    }
}
```

A piano has 88 keys.

Assignment

- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```

  - The expression on the right is evaluated and the result is stored in the variable on the left
  - The value that was in `total` is overwritten
  - You generally only assign a value to a variable that is consistent with the variable's declared type

```
//  Sport.java
// Author: J. Terry     Date: 28/07/2003
//  Demonstrates the use of an assignment statement to change the value stored in a variable

public class Sport
{
      public static void main (String[] args)
     {
          int players = 5;              // declaration with initialization
          System.out.println ("A basketball team has " + players + " players.");

          players = 6;                  // assignment statement
          System.out.println ("A volleyball team has " + players + " players.");

          players = 15;
          System.out.println ("A rugby union team has " + players + " players.");
      }
}
```

A basketball team has 5 players.
A volleyball team has 6 players.
A rugby union team has 15 players.

Primitive Data Types

- There are eight primitive (or simple) data types in Java
- Four of them represent integers:
  - `byte, short, int, long`
- Two of them represent floating point numbers:
  - `float, double`
- One of them represents characters:
  - `char`
- And one of them represents boolean values:
  - `boolean`

Numeric Primitive Data

- The difference between the various numeric primitive types is their size, and therefore the values they can store
- We will predominantly use `int` and `double`

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	$+/- 3.4 \times 10^{38}$ with 7 significant digits	
double	64 bits	$+/- 1.7 \times 10^{308}$ with 15 significant digits	

Characters

- A **`char`** variable stores a single character from the *Unicode character set*
- The Unicode character set uses 16 bits per character, allowing for 65,536 unique characters
- Character literals are delimited by single quotes:

```
'a'    'X'    '7'    '$'    ','    ' '
```

```
char charVariable = 'X';
char anotherCharVariable = '&';
```

Boolean

- A `boolean` value represents a true or false condition
- A `boolean` can be used to represent any two states, such as a light bulb being on or off
- The reserved words `true` and `false` are the only valid values for a `boolean` type

```
boolean done = false;
boolean repeatLoop = true;
boolean invalidInput = true;
```

Identifiers

- *Identifiers* are the words you devise to use in a program ie.
  - variables, constants, methods or the program name

    eg total, count, doCalculation, DemoProgram

- Use letters, digits, the underscore (_), and the dollar sign
- An identifier cannot begin with a digit
- Java is *case sensitive*, therefore `Total` and `total` are different identifiers
- There is no length limit
  - make the identifier name understandable

Conventions

- A rule that we agree to follow, even though it's not required by the language, is said to be a *convention*

- A common Java convention is beginning a class name with an uppercase letter :

```
DemoProgram
String
```

- Names of variables (except constants) and methods, by convention, never start with an uppercase letter

Identifier Conventions

- When an identifier consists of multiple words, it's more readable to mark the boundaries between words
- One way to break up long identifiers is to use underscores between words:

  first__name

- Another technique is to capitalise the first letter of each word after the first. This technique is the common convention used in Java. Use it.

  firstName

  totalSalesValue

Reserved Words

- The Java reserved words (you do not need to remember them – most will become obvious)

```
abstract    default     goto        operator    synchronized
boolean     do          if          outer       this
break       double      implements  package     throw
byte        else        import      private     throws
byvalue     extends     inner       protected   transient
case        false       instanceof  public      true
cast        final       int         rest        try
catch       finally     interface   return      var
char        float       long        short       void
class       for         native      static      volatile
const       future      new         super       while
continue    generic     null        switch
```

Operators

- Java's arithmetic operators:

+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Remainder

Addition and subtraction with `int` operands

6 + 2 equals 8

6 - 2 equals 4

6 * 2 equals 12

Division and Remainder with `int` operands

- The division operator (`/`) returns an integer (the fractional part is discarded)

16 / 3	equals?	**`5`**
9 / 12	equals?	**`0`**

- The remainder operator (%) returns the remainder after dividing the second operand into the first

16 % 3	equals?	**`1`**
9 % 12	equals?	**`9`**

Floating point Operands
eg `double`

- Only `+`, `-`, `*`, and `/` accept `double` operands:

  6.1 + 2.5 equals 8.6

  6.1 - 2.5 equals 3.6

  6.1 * 2.5 equals 15.25

  6.1 / 2.5 equals 2.44

- When `int` and `double` operands are mixed, the result is a `double`

  6.1 + 2 equals 8.1

  6.1 - 2 equals 4.1

  6.1 * 2 equals 12.2

  6.1 / 2 equals 3.05

Operator Precedence

- *, /, and % take precedence over + and –.

- How the compiler read expressions:

5 + 2 / 2	⇒ 5 + (2 / 2)	equals 6
8 * 3 - 5	⇒ (8 * 3) - 5	equals 19
6 - 1 * 7	⇒ 6 - (1 * 7)	equals –1
6 + 2 % 3	⇒ 6 + (2 % 3)	equals 8

Associativity

- Precedence rules are of no help when it comes to determining the value of 1 - 2 - 3.
- The binary +, −, *, /, and % operators are all left associative:

2 + 3 - 4	⇒ (2 + 3) - 4	equals 1
2 * 3 / 4	⇒ (2 * 3) / 4	equals 1

- Parentheses can be used to override normal precedence and associativity rules

Assignment Revisited

- Assignments often use the old value of a variable as part of the expression that computes the new value.
- ie. the right and left hand sides of an assignment statement can often contain the same variable

First, one is added to the original value of `count`

```
count  =  count + 1;
```

Then the result is stored back into `count` (overwriting the original value)

Increment and Decrement Operators

- Shortcuts – use them sparingly

- The *increment operator* (++) adds one to its operand
- The *decrement operator* (−−) subtracts one from its operand

```
count++;
```

is equivalent to

```
count = count + 1;
```

A Program

```
//*************************************************************
// ConvertTemp.java
// Converts a Fahrenheit temperature to Celsius
//*************************************************************************
public class ConvertTemp
{
    public static void main(String[] args)
    {
        double fahrenheit = 98.6;
        double celsius = (fahrenheit - 32.0) * (5.0 / 9.0);
        System.out.println("Celsius equivalent: “ + celsius);
     }
}
```

Celsius equivalent: 37

Constants

- A ***constant*** is a type of variable, but one whose value doesn't change during the execution of a program.
- Constants can be named by assigning them to variables:

```
double base = 32.0;

double conversionFactor = 5.0 / 9.0;
```

- To prevent a constant from being changed, the reserved word `final` can be added to its declaration:

```
final double base = 32.0;

final double conversionFactor = 5.0 / 9.0;
```

Naming Constants

- A convention
- The names of constants are written entirely in uppercase letters, with underscores used to indicate boundaries between words:

```
final double BASE = 32.0;
final double CONVERSION_FACTOR = 5.0 / 9.0;
```

- Programs are easier to read and modify
- Inconsistencies and typographical errors are less likely

```
//********************************************************************
//  TempConverter.java       Author: Lewis/Loftus
//  Computes the Fahrenheit equivalent of a specific Celsius
//  value using the formula F = (9/5)C + 32.
//********************************************************************
public class TempConverter
{
   public static void main (String[] args)
   {
      final int BASE = 32;
      final double CONVERSION_FACTOR = 9.0 / 5.0;

      int celsiusTemp = 24;                  // value to convert
      double fahrenheitTemp;

      fahrenheitTemp = (celsiusTemp  * CONVERSION_FACTOR ) + BASE;

      System.out.println ("Celsius equivalent: " + celsiusTemp);
    }
}
```

Input and Output

- Programs use both input and output

- ***Input*** is any data the program requires
  - It can be already stored in the program or
  - Entered by a user or
  - Obtained from a file or other input device

- ***Output*** is any data produced by the program
  - It can be displayed to the terminal or
  - Written to a file or
  - Used to operate devices

Displaying output on the screen

System.out.println( )

- Prints what is contained in the argument ie. between ( )
- `println()` always advances to the next line after displaying its argument

System.out.print( )

- Does not advance to the next line after displaying its argument

```
//********************************************************
//  Countdown.java       Author: Lewis and Loftus
//
//  Demonstrates the difference between print and println.
//********************************************************
public class Countdown
{
   public static void main (String[] args)
   {
      System.out.print ("Three... ");
      System.out.print ("Two... ");
      System.out.print ("One... ");
      System.out.print ("Zero... ");
      System.out.println ("Liftoff!");       // appears on first output line
      System.out.println ("Houston, we have a problem.");
   }
}
```

Three…Two…One…Zero…Liftoff!

Houston, we have a problem.

Displaying output

- Strings are any characters contained within double quotes
- They are often displayed as output

```
System.out.println("This string will be displayed");
```

- System.out.println( ) and System.out.print( ) can only display arguments of a single type egs Strings or variables

```
System.out.println(fahrenheitTemp);  //displays the value of the variable
```

- To display mixed types we need to concatenate or join numeric types into a string

Concatenation

- The + operator can be used to combine multiple items into a single string for printing purposes:

```
System.out.println(“todays temp: " + fahrenheit);

System.out.println(“todays temp: " + fahrenheit + “ degrees”);
```

The plus operator

- So the plus operator (+) is used for addition and concatenation
- The function that the + operator performs depends on the type of the information on which it operates
- If both operands are `strings`, or if one is a `string` and one is a number, it performs `string` concatenation
- If both operands are numeric, it adds them
- The + operator is evaluated left to right
- Parentheses can be used to force the operation order

```
//********************************************************
//  Addition.java       Author: Lewis and Loftus
//  Demonstrates the difference between the addition and
//       string concatenation operators.
//********************************************************
public class Addition
{
    //-----------------------------------------------------
    //  Concatenates and adds two numbers and prints the sum
    //-----------------------------------------------------
    public static void main (String[] args)
    {
        System.out.println ("24 and 45 concatenated: " + 24 + 45);

        System.out.println ("24 and 45 added: " + (24 + 45));
    }
}
```

```
24 and 45 concatenated: 2445
24 and 45 added: 69
```

Printing long Strings

- Every statement in java ends with a semicolon not with the end of a line
- So any statement can run over multiple lines

- To print a long character string, you can break it up and concatenate the parts

```
System.out.println (“This is a particularly long string ”
                    + “of characters, so it needs to be ”
                    + “broken up and concatenated”);
```

Displaying a Blank Line

- Leave the parentheses empty when calling `println`:

```
System.out.println();     // Write a blank line
```

Escape Sequences

- Escape sequences start with \ followed by a character and are inserted in a string being displayed to alter the display

- `\n` to start a new line:

```
System.out.println("A hop,\nand a jump");
```

```
A hop,
and a jump
```

- `\t` represents a tab space

```
System.out.println(“John’s total:\t “ + salesTotal);
```

```
John’s total:		225
```

Problem Sets

- You will need to write for all the problem sets provided.
- Remember that programming is about solving problems.
- The material in this lecture are sufficient for you to do all the problem sets.
- Ensure that you grasp these principles by reading through all the materials.

Lecture Outcomes

- Today we have covered:
  - The structure of a Java program
  - Printing program output
  - Program readability and layout
  - The declaration and use of variables
  - Assignment
  - Primitive data
  - Arithmetic expressions and operator precedence
- Questions?