Introduction to Programming

2. Programs, Algorithms and Data Types

Input and Output

- In simple programs, the user is the source of input (through the keyboard) and the ultimate destination of output (via the screen)
- Last week we saw how to output to the screen using the class TextIO. On the next slide there is a summary of the methods that TextIO provides to handle input from the keyboard. We'll use some of these in an example program later in the lecture.

Console input with TextIO

- TextIO provides a set of methods which return a value read from the keyboard
- There is one for each primitive type (discussed later in the lecture) and also methods that return a String (see section 2.4.3 of Eck's book). The following examples assume each value typed at the keyboard is followed by an end-of-line:

```
i = TextIO.getlnInt(); // Reads a value of type int.
d = TextIO.getlnDouble(); // Reads a value of type double.
b = TextIO.getlnBoolean(); // Reads a value of type boolean.
c = TextIO.getlnChar(); // Reads a value of type char.
w = TextIO.getlnWord(); // Reads one "word" as a String.
s = TextIO.getln(); // Reads an entire input line as a String.
```

Programs and Algorithms

- For the rest of this lecture will look at how we write some of the instructions that programs execute when they are run in order to process their inputs to produce their outputs
- Will start with the idea of an *algorithm*

Algorithm

- A set of step-by-step instructions to solving a problem or doing a task
  - Basis of computer programs; also technical instruction manuals, recipe books…
  - Finite number of steps (must reach a goal)
  - Many standard algorithms for common tasks (usually maths and computing)
  - No intelligence required to follow one
- Named after:
  - Persian mathematician, Abu Ja’far Muhammad ibn Musa **al-Khwarizmi** (circa 800 A.D.)

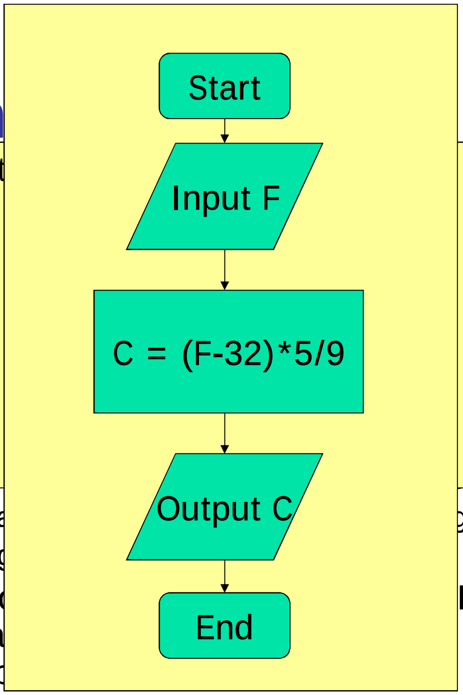
Example algorithm

- Convert ºFahrenheit to ºCelsius
  1. Take number of degrees in Fahrenheit scale
  2. Subtract 32
  3. Divide by 9
  4. Multiply by 5 – result is in Celsius scale
- None of these steps takes any special skill or ability (a £2 calculator can do the arithmetic) under the direction of a human
- Most algorithms are not so simple (more later on how to describe these)
- Mostly, algorithms are simple representations of what you do to complete a task

Writin

```
public st
{

}
```



- T ake it
  (ritten form
- V Structured
  lgorithm
  - Diagra grams
  - A prog
- Most pr DL at an early stage, a language as the design b

Explaining the Java Program

```
// Comments are in blue – to explain the code, ignored by compiler
// These comments start with a double / and stop at the end of line

public class Application {   // Introduces a new class (called Application)
   static float convertFtoC(float F) {   // Introduces a new method (called
                                         // convertFtoC) that returns a result
         return (F-32) * 5 / 9;  // This does the job and returns a result
   }  // This curly bracket ends the convertFtoC method

   public static void main(String[] args)  {  // Every Java application has
                                               // this method – defines start
      float tempF;   // This declares a variable (tempF) of type float
      TextIO.put("Enter temperature in °F: ");   // This prints a message
      tempF = TextIO.getlnFloat();                // This gets keyboard input
      TextIO.putln("Temperature in °C is " + convertFtoC(tempF));
   }      // The line above calculates the result and displays it on the screen.
}    // This curly bracket ends the Application class
```

Algorithms in Java

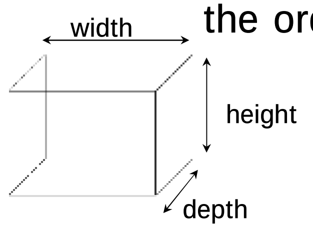
- Simple algorithms can be written directly in Java
  - Not recommended for describing algorithms when developing large systems, since Java adds too much detail to be used at an early stage
  - Need to follow Java *syntax rules* – the grammar of the language
- An algorithm language needs to contain
  - ways of defining/describing data
  - clear, simple formats for expressing instructions
  - ways of imposing *structure* on what is written
    - Structure: how groups of statements are packaged and controlled
    - e.g. main(){ } method is a structure element

Java - instructions

- An executable instruction (or statement) in a Java program often does one of three things
  - Assigns a value to a variable (replacing any previous value it stored)
    - e.g. x = y + 5;
  - Calls a method that performs some task
    - e.g. drawCircle();
  - Calls a method that returns some result that the program uses and which may be assigned to a variable
    - e.g. y = sqr(x);
- Other Java statements may declare variables, or define methods, or control in some way when statements are executed.

Statements and Sequences

- Fundamental rule of programming –
  - In the absence of any other structure, a sequence of statements gets executed in the order in which they appear:



```
areaTop = width * depth;
areaSide = height * depth;
areaFront = width * height;
totalArea = 2 * (areaTop + areaSide + areaFront);
```

- Another order of execution might produce the wrong answer

Blocks

- In Java, a group of individual statements can be combined by enclosing them in braces { }
- We will see later that such ***blocks*** are essential to structured programming...

```
{
        areaTop = width * depth;
        areaSide = height * depth;
        areaFront = width * height;
        totalArea = 2 * (areaTop + areaSide + areaFront);
}
```

Basic Java Applications

- A program is a *sequence of instructions*
- The kind of Java program that we will study is a Java *application*
  - An application has a method named *main()*
  - The main() method is part of a Java class
  - When you run the program, the instructions that start running are the instructions in the main() method

The main() method

- Looks like this

```
public class AnExampleApplication {
   public static void main(String[] args) {
         // The instructions go here, inside main()
         // When the program runs, these are the
         // instructions that get executed.
   } // end main()
} // end class
```

- This source code would be in a .java file with the same name as the class (in this case, AnExampleApplication.java)

Console Applications

- Many Java applications produce graphical interfaces for their users and respond to input from the mouse as well as the keyboard
- In this module we restrict ourselves to Java applications that display text output on the console (terminal screen) and take their input from the keyboard

Programs need to remember things

- When a user types something at the keyboard, a program will often need to remember what is typed
  - It might be a value such as an account number
  - It might be a choice on a menu that needs to be referred to more than once
- The program will often complete calculations, and need to remember the result to use in later calculations or output

Variables

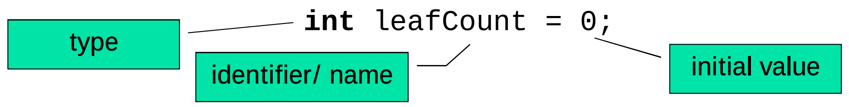
- Programs use computer memory to store values
  - Input data
  - Results of calculations
  - Temporary storage of intermediate results, etc.
- A *variable* is a named location in the memory of the computer that is used to store a value of a given type

Declaring a variable

- In Java, a variable must be *declared* before it can be used.
- The declaration gives the variable a name, and it states what kinds of values the variable can store.
- When the variable is declared, memory is set aside to store the value that the variable is given to hold.

Declaring Variables

- A variable declaration states the **type** of the variable as well as its **name** (identifier) and may assign an initial **value**



```
int leafCount = 0;
```

- The above declaration results in a location in computer memory being set aside to store integer values, and stores the value 0 in that location
- Once declared, the variable name can be used to retrieve the value stored in the variable…

```
TextIO.putln(leafCount); // prints value (e.g. 0)
```

- and can be used to update the value

```
leafCount = leafCount + 1; // add 1 to leafCount
```

Types

- Every variable has a *type*
- A type is a set of values and the set of things that you can do with those values
- Java provides two kinds of types
  - "Primitive" types
  - Classes (and other reference types)

Java's Primitive Types

- Java has eight "primitive" types
- These are used to represent numbers, characters (letters), and true/false values
- The names of the primitive types are all Java keywords – note that they are written entirely in lower case
- The four primitive types we will use most are
  - **int** – values are integers (0, 12, 32767, 10000001)
  - **double** – values are real numbers (0.0, 2.005, 3.141592653589793, 1.0e6 [which is same as 1000000.0])
  - **char** – values are individual characters ('a', 'A', '9', '%')
  - **boolean** – values are **false** and **true**

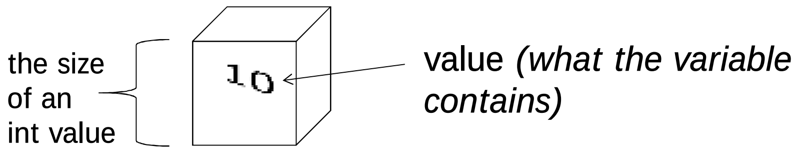
Types continued

- In Java, a variable must be declared before it is used
  - That is, the compiler needs to be told the name and type of the variable before the program can use the variable
- The names of the primitive types are keywords that can be used to declare the type of the variable
- Each variable can store a single value at any given time, the type named in declaring the variable indicates what type of value this is

Variables: summary

- Declaring a variable sets aside memory at a named location to store a value

```
int leafCount = 10;
```



Name: `leafCount`
Type: `int`
(address: 0xf840026018)

Assignment and Equality

- An assignment statement is a statement that stores a value in a variable
- In Java, uses = operator

```
int answer;
answer = 42;  // assignment statement
```

- Do *not* confuse with == operator, which compares two values for equality

```
if (answer == 42) { /* do something */ }
```

More on variables

- As a program runs, it encounters variable declarations. These result in memory for the variable being allocated
- How long the memory is allocated for, and how long the variable exists, depends on where and how the variable is declared.

Local Variables

- A variable declared inside a method or a block is an example of a *local variable* and exists only while the method or block is executing

```
public class LocalVariablesExample {
   public static void main(String[] args) {
     int count = 0;
     boolean isZero = (count == 0); // true
     if (isZero) { // note beginning of block
        int inner = count + 1;
        TextIO.putln(inner);
     } // end of block, inner no longer exists
   } // end of method, count & isZero no longer exist
}
```

Initialising Local Variables

- You must give a value to a local variable before you can use the variable as a value
  - You can give it a value when you declare it, as in the previous slide, or later, but you *must* have given it a value before you try and access the value stored in the variable
  - Assigning the first value to a variable is termed *initialising* the variable

```
public class LocalVariablesExample3 {
   public static void main(String[] args) {
     int count1 = 0;  // declared & initialised
     int count2;  // declared
     count2 = 0;  // initialised
     System.out.println(count1 + count2);
   }
}
```

Initialising Local Variables

- The compiler will give an error if you try and access the value of a variable when the variable might not have been given a value
  - *e.g.* the code below will not compile as *count* has not been initialised before its value is used

```
public class LocalVariablesExample3 {
   public static void main(String[] args) {
     int count; // declared, not initialised
      boolean isZero;
      isZero = (count == 0); // won't compile!
   }
}
```

The Application Example again!

```
public class Application {

    static float convertFtoC(float F) {
            return (F-32) * 5 / 9;
    }

    public static void main(String[] args)  {
       float tempF;   // This declares a local variable, tempF
      TextIO.put("Enter temperature in °F: ");
      tempF = TextIO.getlnFloat(); // This initialises the local variable.
      float tempC = convertFtoC(tempF); // This declares and initialises
                                        // another local variable, tempC.
                                        // Also, uses stored value of tempF.
      TextIO.putln("Temperature in °C is " + tempC); // Uses value of tempC.
    }
}
```

Reading for Next Week

- **Eck, chapter 2**
  - Chapter 2 provides a more extended treatment of what we covered today
  - The later sections also include coverage of things that we will meet in later lectures
    - Enums
    - Files
- Next week we will look at some details of operations on types, and more on program statements

End

- Questions?