Input, Output, Strings and Characters

This lecture will

- Introduce facilities in the **`sheffield`** package for input and output
- Explain the difference between basic types and objects
- Briefly demonstrate the use of a graphical user interface (GUI)
- Explain the **`String`** class and demonstrate some of the many methods available to manipulate **`String`**s
- Introduce the Java type **`char`** for storing and manipulating Unicode characters

Identifiers

- So far we have encountered four types of identifiers
  - Constant names are in uppercase with the words separated by an underscore , _
  - Variable names are mostly in lower case but use upper case letters to mark the start of new words and start with a lower case letter
  - Class names are like variable names except they start with a capital letter
  - Method names are like variable names except they are always followed by a pair of round brackets possibly with something inside them
- And that's all there are

Constants v. Variables

- The declaration of a constant contains the additional word, `final` and you can't change its value
- You can change variables but up to now we haven't done that much
- In the exercises there was an example which worked out the area of a circle πr^2
- π was a constant and the radius was a variable but neither actually changed

Constants v. Variables

- This works out the volume of a sphere which is $\frac{4}{3}\pi r^3$

```
public class Sphere {
  public static void main (String [] args) {
    int radius = 10;
    System.out.print("The volume of a sphere with radius "
        + radius + " is ");
    System.out.println(
        4.0/3.0 * Math.PI * Math.pow(radius, 3));
  }
}
```

The plus sign joins strings

We need at least one decimal point to avoid integer division

This constant is always available as is Math.E

Parameters of methods are separated by commas

This is a method to raise its first parameter to the power of the second

Constants v. Variables

- This works out the volume of a sphere which is $\frac{4}{3}\pi r^3$

```
public class Sphere10 {
   public static void main (String [] args) {
      int radius = 10;
      System.out.print("The volume of a sphere with radius "
            + radius + " is ");
      System.out.println(
            4.0/3.0 * Math.PI * Math.pow(radius, 3));
   }
}
```

- Actually it only works out the volume of a sphere with a radius of 10
- We need to ask the user what the radius is

EasyReader

- Many programs need to **input** data from the keyboard and **output** a result
- Java provides extensive, versatile input and output (I/O) facilities
- It doesn't provide simple, foolproof ones
- To start with we are going to use some homemade ones
- We have written a class called **EasyReader** which you are going to use
- EasyReader contains a lot of **methods** to read things in

Reading things in

```
import sheffield.*;

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

      EasyReader keyboard = new EasyReader();

      double radius = keyboard.readDouble(
            "What is the radius of your sphere? ");
      System.out.print("The volume of a sphere with radius "
            + radius + " is ");
      System.out.println(
            4.0/3.0 * Math.PI * Math.pow(radius, 3));
   }
}
```

This says what we mean by keyboard

This asks the user a question and reads the answer in

Reading things in

```
import sheffield.*;
public class Sphere {
   public static void main (String [] args) {
      EasyReader keyboard = new EasyReader();

      double radius = keyboard.readDouble(
            "What is the radius of your sphere? ");
      System.out.print("The volume of a sphere with radius "
            + radius + " is ");
      System.out.println(
            4.0/3.0 * Math.PI * Math.pow(radius, 3));
   }
```

```
U:…myjava>java Sphere
What is the radius of your sphere? 1
The volume of a sphere with radius 1 is 4.1887902047863905
U:…myjava>
```

Using EasyReader

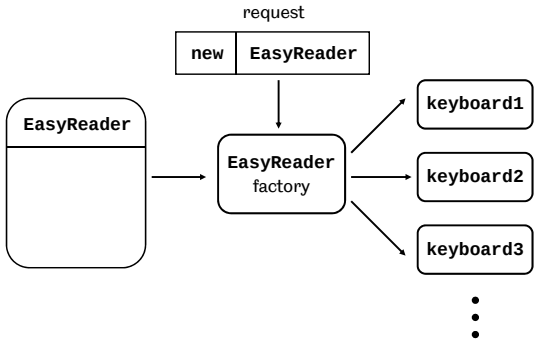
- To use **EasyReader**, we create an **EasyReader** object:

```
EasyReader keyboard = new EasyReader();
```

- This is a variable declaration, but **EasyReader** is a class (not a basic type) so we use the keyword **new** and call a special method, called a **constructor**, which has the same name as the class.
- A new **instance** of the class **EasyReader** is created. This is an **object** of type **EasyReader**
- A **reference** to the object is stored in the variable **keyboard**
- So, **an object is an instance of a class**

A class is a 'blueprint' for objects

- We can make as many instances of a class as memory allows; it's like a factory that makes objects



Comparing basic types and objects

- We can use the **method** **readDouble** of **EasyReader** to read a real number from the keyboard:

```
EasyReader keyboard = new EasyReader();
double radius = keyboard.readDouble("What radius? ");
```

- This can be read as "send the **readDouble** message to the object called **keyboard**, and store the result in the variable **radius**"
- The **double** variable, **radius**, is a location in memory where a real number is stored.
- However, **keyboard** is a location in memory that contains a **reference** (pointer) to another location in memory where an object is stored.
- This distinction is **very important**.

EasyReader's readInt() method

```
import sheffield.*;
public class WallPaper {

   public static void main(String[] args) {
      // get the dimensions of the room from the user
      EasyReader keyboard = new EasyReader();
      int length = keyboard.readInt("Enter the length: ");
      int width = keyboard.readInt("Enter the width: ");
      int height = keyboard.readInt("Enter the height: ");

      // do the calculations
      int carpetSize = length*width;
      int wallpaperSize = 2*height*(length+width);

      // print the result
      System.out.println("Your room needs " + carpetSize +
               " square metres of carpet and");
      System.out.println(wallpaperSize +
               " square metres of wallpaper");
   }
}
```

Running the program

- The user input is shown in yellow

```
U:…myjava>java WallPaper
Enter the Length: 5
Enter the width: 8
Enter the height: 3
Your room needs 40 square metres of carpet and
78 square metres of wallpaper
```

Importing the sheffield package

```
import sheffield.*;

public class WallPaper {
   public static void main(String[] arg) {
      // get the dimensions of the room from the user
      EasyReader keyboard = new EasyReader();
      int length = keyboard.readInt("Enter the length: ");
      int width = keyboard.readInt("Enter the width: ");
      int height = keyboard.readInt("Enter the height: ");

      // do the calculations
      int carpetSize = length*width;
      int wallpaperSize = 2*height*(length+width);

      // print the result
      System.out.println("Your room needs " + carpetSize +
               " square metres of carpet and");
      System.out.println(wallpaperSize +
               " square metres of wallpaper");
   }
}
```

This makes EasyReader available

The sheffield package

- As well as EasyReader we have created two other classes to help you get started with Java's IO
- The full set is
  - EasyReader
  - EasyWriter
  - EasyGraphics
- They are all bundled together in the **sheffield package**
- A **package** is a collection of related Java classes

Importing EasyReader

- The **`import`** statement makes all classes in the package called **`sheffield`** available for use in the program:

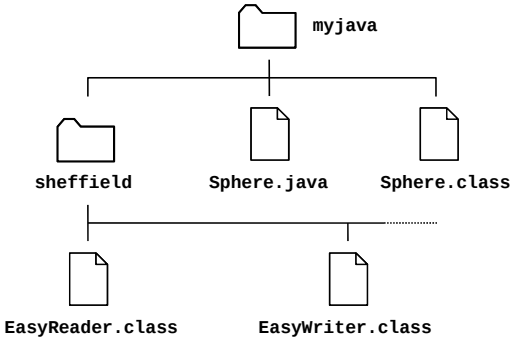
```
import sheffield.*;
```

- If we only want to use **`EasyReader`**, we could write:

```
import sheffield.EasyReader.*;
```

- There is no overhead in importing all classes; only the code for classes that are used is linked
- The import only works if the sheffield package, which is in a directory, is in the same directory as your program

Importing and path names



The EasyWriter class

- As well as **EasyReader**, the **sheffield** package contains **EasyWriter** to help with output
- **EasyWriter** provides methods similar to **System.out.print** and **System.out.println**, but allows formatted output of numbers too
- Avoid using both **System.out** and **EasyWriter** in the same program

Formatted output with EasyWriter

- The **System.out** object is automatically available, so we can write statements like:

```
System.out.println("What are you growing?");
```

- To use **EasyWriter** we need to import the **sheffield** package as with **EasyReader** and create an instance of the **EasyWriter** class (i.e., an **EasyWriter** object) before its methods can be invoked:

```
EasyWriter screen = new EasyWriter();
```

- If you don't need formatted output, its easier to use **System.out** (and the program is one line shorter).

Formatted output of integers

- Integer numbers can be aligned within a field:

```
EasyWriter screen = new EasyWriter();
screen.println(3189,6);
screen.println(13,6);
screen.println(534,6);
```

Parameters are separated by commas

```
3189
  13
 534
```

- The first parameter is the number to be output and the second is the number of character positions it is to take up. The integer is aligned to the right within these character positions

Formatted output of real numbers

- Real numbers can be displayed to a certain number of decimal places and with decimal points aligned
- If the **`println`** method is called with only a real number as a parameter it is the same as **`System.out.println`**
- If it is called with two parameters, a real number and then an integer the second parameter specifies the number of decimal places the first will be printed out to
- If it is called with a third parameter the final parameter specifies how wide it is to be

```
screen.println(3.14159265359);
screen.println(3.14159265359,3);
screen.println(3.14159265359,4,10);
```

```
3.14159265359
3.142
    3.1416
```

Notice rounding

Using text files

- Sometimes it is useful to get programs to take input from a text file or save output to one
- The sheffield package has facilities for this
  - **`new EasyReader()`**
    reads from the keyboard
  - **`new EasyReader("file.txt")`**
    reads from the named file
  - **`new EasyWriter()`**
    displays to the screen
  - **`new EasyWriter("otherfile.txt")`**
    writes to the named file **and destroys any existing version of `otherfile.txt`**

Input from a text file

This line says what file to use. It has to be in the same directory as the program

```
import sheffield.*;
public class CropAreaFromFile {
   public static void main(String[] arg) {
      EasyReader fileInput =
                 new EasyReader("croparea.txt");

      // read the field's length and width
      double width = fileInput.readDouble();
      double length = fileInput.readDouble();

      // write the result
      System.out.println("Your field has an area of "
                 width*length+" metres squared.");
   }
}
```

`readDouble` doesn't need a parameter saying what to type

Running the program

- If the **`croparea.txt`** file contains the following data:

```
2.5
8.2
```

- Running the program gives

```
U:…myjava>java CropArea
Your field has an area of 20.5 metres squared.
U:…myjava>
```

- Taking input from a text file can be useful when debugging programs that require a lot of keyboard input or programs with a lot of data

Output to a text file

```
import sheffield.*;
public class FormattedOutputToAFile {
    public static void main(String[] arg) {
        EasyWriter file = new EasyWriter("output.txt");
        double x = 2.184918284982;
        double y = 127.318291823;
        file.println(x);   // same as System.out.println(x)
        file.println(x,3); // show three decimal places
        // show five decimal places in a field of 10 spaces
        file.println(x,5,10);
        file.println(y,5,10);
    }
}
```

- The file **`output.txt`** is created with the contents:

```
2.184918284982
2.185
   2.18492
 127.31829
```

Input from a GUI

- We won't generally address **graphical user interface** (GUI) programming in this module but a class called **`JOptionPane`** is easy to use for input from a GUI

Note import

```
import javax.swing.JOptionPane;
public class TestDialogue {
    public static void main(String[] args) {
        String response =
            JOptionPane.showInputDialog("Enter name: ");
        System.out.println("Hello "+response);
    }
}
```

- **`JOptionPane`** is part of Java so it doesn't need to be in the same directory

Output from the program

```
        String response =
            JOptionPane.showInputDialog("Enter name: ");
        System.out.println("Hello "+response);
    }
}
```

Output from a GUI

- The **`JOptionPane`** class provides a convenient method for displaying a message too:

```
JOptionPane.showMessageDialog(null,
                    "Put your message here");
```

- This also needs **`import javax.swing.JOptionPane;`**
- See the online documentation for many other ways of customising the **`JOptionPane`** input and output dialogs.

Strings

- We have seen variable declarations of two basic types

```
int radius = 10;
double width = 3.5;
```

- And two classes

```
EasyReader keyboard = new EasyReader();
EasyWriter file = new EasyWriter("output.txt");
```

- We can also declare named String variables

```
String greeting = "Hello";
```

String

- **int** and **double** are **basic types**
- **EasyReader** is a **class**
- Class identifiers start with capital letters
- Class objects are created with the keyword **new**
- What is **String**?

```
String greeting = "Hello";
```

The String class

- A **String** is a sequence of characters of indeterminate length surrounded by a pair of double quotes
- Examples are **"Hello world!"**, **"&*#?"** and **""**
- The **String** data type is not a basic type; it is a **class**
- Since **String** is so commonly used Java allows a short notation in which the **new** keyword is not required

```
String greeting = "Hello";
```

- However we could use **new** if we wanted to:

```
String greeting = new String("Hello");
```

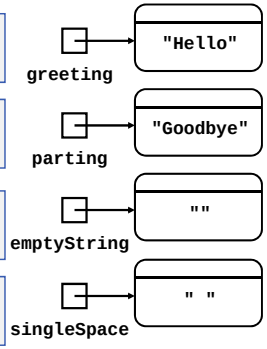
Making instances of the String class

```
String greeting = "Hello";
```

```
String parting = "Goodbye";
```

```
String emptyString = "";
```

```
String singleSpace = " ";
```



Line breaks in programs

- Java will allow spaces or line breaks anywhere except within variable names, reserved words and almost all literal values
- Java will allow spaces within String literals but not line breaks
- This is correct

```
System.out.println("and " + wallpaperSize + " square"+
        " metres of wallpaper.");
```

- This is incorrect

```
System.out.println("and " + wallpaperSize + " square
        metres of wallpaper.");
```

Concatenation

- We **concatenate** (join) strings using the '**+**' operator
- Note that '**+**' is said to be **overloaded**. If the operands are numeric it means addition, if at least one operand is a **character string** it means concatenation

```
System.out.println("There are "+52+" cards");
```

```
There are 52 cards
```

- Character strings can be read in using EasyReader

```
String answer = keyboard.readString(
                "Please answer the question ");
```

Escape sequences

- It is possible to put double quotes and other unprintable characters (e.g. tab key) into **`String`** literals using an **escape sequence** – two characters that result in just one

Escape sequence	Meaning
`\t`	tab
`\n`	newline character
`\"`	double quote character
`\\`	backslash (the character '\')

Example – concatenation

```
public class StringJoin {

   public static void main(String[] args) {
         String s1 = "Hello";
         String s2 = "Hello Hello";
         String s3 = s1 + s2;
         String s4 = s1 + " " + s2;
         String s5 = "He said \"Hello\".";
         String s6 = "He said \"" + s4 + "\".";
         System.out.println(s6);
   }
}
```

```
He said "Hello Hello Hello".
```

Classes can have methods

- The method **`length()`** returns the number of characters in a **`String`**

```
import sheffield.*;
public class StringLength {
   public static void main(String[] args) {
      EasyReader keyboard = new EasyReader();
      String s =
         keyboard.readString("Enter string: ");
      System.out.println(
         "Length is: +s.length());
   }
}
```

```
Enter string: Hello
Length is: 5
```

Substrings

- We can extract part of a **`String`** using the **`substring`** method
- You specify the position of the first character and the position **after** the last character. All character positions are counted from zero

```
s1 = "Sheffield";
s2 = s1.substring(2,7);   // s2 is "effie"
s3 = s1.substring(0,4);   // s3 is "Shef"
```

- If you just use one parameter it is as though the 2nd parameter were the length of the string

```
s1 = "Sheffield";
s2 = s1.substring(4);   // s2 is "field"
```

Other methods of the String class

- **`trim()`**
  Creates a new String with spaces and tab characters removed from both ends of the **`String`** it is applied to but not ones that occur in the middle
- **`toLowerCase()`** & **`toUpperCase()`**
  Creates a new String with the case modified
- **`indexOf(otherString)`**
  When applied to a String it finds the position of the String supplied as a parameter

Using `String` methods

```
public class StringTest {
   public static void main(String [] args)  {
      String aboutACat = "The cat sat on the mat";
      System.out.println(aboutACat.toUpperCase());
      System.out.println(aboutACat.substring(3,7));
      System.out.println(aboutACat.
             substring(3,7).toUpperCase().trim());
      System.out.println(aboutACat.indexOf("cat"));
   }
}
```

```
U:…myjava>java StringTest
THE CAT SAT ON THE MAT
 cat
CAT
4
```

The char data type

- A variable of basic type **char** can be used to store a single character from the **Unicode** character set.
- Unicode has now superseded **ASCII**, which used 7 bits to represent 128 different characters. ASCII could not encode documents in non-Western writing systems (e.g., Arabic).
- The actual Unicode characters that are displayed are determined by a **character encoding**, the most common being UTF-8, which maintains ASCII codes for ASCII characters
- The Unicode standard defines more than 100,000 characters

Assignment with the char type

- Single quotes are used to denote a literal character value:

```
char c1 = 'E';
char c2 = '@';
```

- For the single quote value you need an escape character too

```
char prime = '\'';
```

- A single space or any of the other escape characters can be declared as **char** variables too

Example – convert char to Unicode

```
import sheffield.*;
public class CharToUnicode {
   public static void main(String[] args) {
      EasyReader keyboard = new EasyReader();
      char ch =
         keyboard.readChar("Type a character: ");
      System.out.print("Unicode number is: ");
      System.out.println((int)ch);
   }
}
```

readChar is another **EasyReader** method

```
Type a character: F
Unicode number is: 70
```

Example – convert Unicode to char

```
import sheffield.*;
public class UnicodeToChar {

   public static void main(String[] args) {

      EasyReader keyboard = new EasyReader();
      int number = keyboard.readInt(
         "Type a decimal Unicode number: ");
      System.out.print("char is: ");
      System.out.println((char)number);

   }
}
```

```
Type a decimal Unicode number: 71
char is: G
```

Strings and chars

- A **String** of length 1 is not the same as a **char**
- **String** literals are delimited with double quotes
- **char** literals are delimited with single quotes
- Both

```
String s = 's';
```

and

```
char c = "c";
```

will cause compilation errors

The char type and concatenation

- You can use the + operator to concatenate a character with a character string

```
int cardNo = 6;
char suit = 'D';
System.out.println("The suit is "+suit);
```

```
The suit is D
```

- But not to concatenate a character with a number

```
int cardNo = 6;
char suit = 'D';
System.out.println(cardNo+suit);
```

```
74
```

Summary of key points

- **EasyReader** and **EasyWriter** provide a simple way of handling user input and formatted output but they need an **import** statement
- Variables with a **basic type** correspond to a box in memory that contains a data value (e.g., a number)
- Variables with an object type correspond to a box in memory that contains a **reference** an object and an object is an **instance** of a class

- The **String** class, which provides many useful methods, is unique in that objects can be created without **new**
- The **char** basic type stores a Unicode character
- A **char** is not a **String** of length 1 and trying to treat them as the same causes problems