5 Basic Input and Output

Read and understand this short section on input and output. There are some pieces you will have to use as recipes at the moment and you will learn the reasons later.

5.1 Introduction

Before we start any real programming it is essential that we can read-in and write-out information and in particular the value of *variables*. In JAVA input and output is very flexible, and as a consequence rather complex. It is *expected* that input will come via dialogue boxes or menus and output will be graphical or via pop-up windows with buttons and scrollbars etc. This is a tough start for novices, so to simplify things we will use additional *Class*es to do the "hard work". This will allow you to concentrate on learning basic programming language and produce useful and interactive programs. In this section we will learn basic text *input* and *output* via the locally written *Display* and *Input* classes. Further on in the course we will meet the extra *SimpleGraph* class which will allow you to display your output graphically.

Note: In this course we will *not* need file input and output. There is however an *optional* section at the end of this book covering basic file input/output that will be useful in future courses.

5.2 Classes and Methods

JAVA is an "object oriented" language which contains *classes* which define the structure of the data and *methods* which act on these *classes*. This somewhat abstract concept is initially confusing, and is better illustrated by examples rather than pages of definitions.

5.3 Display Class

The *Display* class provides "input and output" screen within a new window to which you can read and write. It also offers basic mouse control, so you can halt your program, wait until you have completed you input instructions and then run it.

The use is best explained by an example that simply read in a `int` and prints it out again:

```
import uk.ac.ed.ph.sciprog.*;                                          // (1)

public class ReadInteger{                                              // (2)
  public static void main(String args[]){                              // (3)

    //                Setup the Display

    Display myDisplay = new Display("A simple input/output program"); // (4)
    Input numberInput = new Input("Give an integer", 10);              // (5)
    myDisplay.addInput(numberInput);                                   // (6)

    //                Wait until ready, then read integer

    myDisplay.waitForButtonPress();                                    // (7)
    int iValue = numberInput.getInt();                                 // (8)
```

```
    myDisplay.println("Value of integer is : " + iValue);            // (9)
  }
}
```

This looks more complex than it really is, so lets go through it *Line by Line*,

(1) This `import`s the locally[1] written `sciprog` classes so making them available in your program.

(2-3) Declares your program as class `ReadInteger`, and start of main program. Note: This program **must** be contained in a file called `ReadInteger.java`.

(4) Construct a `Display` object called `myDisplay`, and sets its title to `"A simple input/output program"`. Note the `new` command, this actually creates the *Display*.

(5) Construct an `Input` object called `numberInput` with a prompt string `"Give an Integer"` and a default value of `10` .

(6) Adds the `Input` object `numberInput` to the `Display` object so that it appears in `myDisplay` when it is shown on the screen.

(7) Causes `myDisplay` to be shown on the screen, then waits for you to change the input value, as required, and press the GO button.

(8) Reads an `int` value from the `numberInput` object and stores it in the `int` variable `iValue`.

(9) Prints the output string

```
"Value of integer is :  " + iValue
```

to the output panel of the display. Note that by "adding" `iValue` to a `String` it is converted into a `String` and appended.

You should type-in this program, calling it `ReadInteger.java` *and make sure it works.*

You can read other data types, in particular `double`, `String` or `boolean` by first creating a `Input` object with the *default* type you wish to read, for example,

```
Input doubleInput = new Input("Give a double" , 0.0);
Input stringInput = new Input("Give a string" , "Hello");
Input booleanInput = new Input("Yes or No" , true);
```

will create `Input` objects that will read a `double`, `String` and `boolean` respectively. Note the `boolean` version will create a clickable "Pop-Down-Menu" rather than an than a editable input field.

Then reading the values with the *methods* `.getDouble()`, `.getString()` or `.getBoolean()` respectively, for example

[1]The first part of the name is the internet address of the School of Physics, this ensures that this class in not confused with a class of the same name from somewhere else.

```
double value = doubleInput.getDouble();
String name = stringInput.getString();
boolean isCorrect = booleanInput.getBoolean();
```

will read the values.

5.4 Format of Variables

We have seen in the previous section and in the above example, that we can display the value of a `int` or `double` by simply "adding" it to a `String`, and then printing the `String`. This works well for `int`s, but with `double` it gives all available digits, typically 14, which is rather unwieldy.

There are *two* schemes in JAVA to deal with this,

1. The `DecimalFormat` class which is part of the `java.text` package. This is very flexible and very powerful but a bit cumbersome to use.
2. The C-style `printf()` methods recently implemented in JAVA[2]

We shall use the second of these schemes, covering the *absolute minimum* now, with more details section on array and Strings.

The general syntax of the `printf()` which can take a variable number of arguments is,

```
printf(String template, Object, Object, .... )
```

where the String `template` layout how the arguments are to be formatted. This rather complex looking syntax is best explained by a couple of examples

```
int intValue = 15;
double doubleValue = 12.546786534;
System.out.printf("Values are %d and %f\n",intValue,doubleValue);
```

will print to the screen

```
Values are 15 and 12.546787
```

with the `%d` being the location of the integer formatted in **d**ecimal, and `%f` being the location of the double being formatted in **f**ixed point format, which by default, gives 6 decimal places.

Note the `"\n"` at the end of the template string, this means *newline characters*.

There are two other important keys for doubles, these being,

- `%e` formats a double in **e**xpotential notation with a default of 6 decimal places with correct rounding.
- `%g` formats a double in **g**eneral notation with 6 significant figures using exponential notation. This is the most generally useful.

You can also control the number of significant figures, *or* number of decimal places as follows,

[2]Only available in JAVA 1.5, also known as JAVA 5.0

`%.3f` will format a double in **f**xed point format with 3 decimal places.

`%.5e` will format a double in **e**xpotential format with 5 decimal places.

`%.7g` will format a double in **g**eneral format with 7 significant figures.

so for example

```
double plank = 6.67E-34;             // Planks Constant
double lightSpeed = 2.999867E8;      // Speed of light.
System.out.printf("Planks constants is %.3g\n", plank);
System.out.printf("Speed of light is %.4e\n",lightSpeed);
```

will give as output,

```
Planks constants is 6.67e-34
Speed of light is 2.9987e8
```

This is the *absolute minimum* you need to know about this highly flexible formatting scheme at this point. There are more details in the later section on arrays and Strings.

Using `printf()` with `Display`

The *Display* class implements the `printf()` method, so you can can write the following:

```
import uk.ac.ed.ph.sciprog.*;

public class Convert{

  public static void main(String args[]){


    Display myDisplay = new Display("A simple input/output program");
    Input wavelengthInput = new Input("Wavelength in nanometres",0.0);
    myDisplay.addInput(wavelengthInput);

    myDisplay.waitForButtonPress();
    double lambda = wavelengthInput.getDouble();

    lambda = lambda/1.0e9;

    myDisplay.printf("Wavelength in metres to 3 places %.3e\n",lambda);
    myDisplay.printf("Wavelength in metres to 5 places %.5e\n",lambda);

  }
}
```

Again you are strongly encouraged to type this program into a file called `Convert.java` and make sure that it works and you understand it. You should then vary the format keys and see what happens, things to try are,

1. Try `%.4f` what happens and why?
2. Try an illegal key, for example `%d` which will try and format a double as an integer, again see what happens.

Note: This example only works once, you will see in the section on loops how to make this program "loop-round" continually asking for input.

Examples

The following on-line source examples are available

- Read an single integer ReadInteger
- Read multiple variables ReadVariable
- Format a double with specified template FormatExample

What Next

Your are now ready to try *Checkpoint 2*.