

Assignment 1: Java Language Refresher

Introduction

The subject of this report is a series of java programs that were created by our group in response to the software development exercises set in Assignment 1. They are a series of relatively short programs, created either in Notepad or Symantec Cafe, and compiled in a DOS window using the “javac” compiler.

In this report we will analyse each program in turn, taking source code from one member of the group. Complete source code from all programs is available in the appendix. We will also explain tricky pieces of code and show how we tested the programs.

Research

To research writing a report, a number of searches were made on the Internet, bringing some interesting results. For example, a page on a Student Help site, <http://www.livhope.ac.uk/livhope/gnu/stuhelp/report.htm>, has the following information, which has been edited to conserve space:

“ Writing a report

1. Set a structure which deals with the material logically e.g.

Title page

Contents

Executive summary

Introduction

Aims and objectives

Background to study

Methodology

Sources of evidence used.

Description of how evidence was collected and analysed.

Discussion of the limitations of the sources and methods of collection and analysis.

Presentation of results

Analysis and discussion of results

Evaluation and conclusion

Bibliography

Appendices “

Other sites have different information regarding creating reports, including the following pages: <http://fbox.vt.edu:10021/eng/mech/writing/>,
<http://www.msue.msu.edu/aee/dissthes/guide.htm>,
<http://www.neltec.com/scifair/report.htm>

Exercise 1: “Hello World”

One of the most basic programs that can be made, and the subject of the first lesson of the majority of programming courses, the “Hello World” program is intended to introduce the basics of the programming language in hand, the basics upon which all future programs will use and follow.

In java, the nuts and bolts of the “Hello World” program is 1 line long, namely the line where we issue the command to output some text onto the screen, `System.out.println("Hello World!");`. The rest of the program is the “basics” that were mentioned above, such as the class definition Hello, which must be the same as the filename of the java program, and the main class declaration, which includes the option of passing in parameters at run-time.

Apart from this the program is unremarkable, and only served to clear the programming cobwebs that developed in our heads over the summer.

```
// A program to print out the words 'Hello World'

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

Exercise 2: “Hello World” 10 times

Our second program was identical to the program above, except that we needed a loop construct before we issued the command to output the “Hello World” text. The loop construct used was the standard FOR loop, in the following format:

```
for( int i=0; i<10; i++ )
System.out.println("Hello World!");
```

The ‘for’ loop has the effect of executing the code that follows a number of times determined by the contents of the ‘for’ pair of brackets. In this case we used the variable ‘i’ to count from 0 to 9, therefore executing the code in the following line ten times.

Testing the above pair of programs was extremely easy. As long as the programs compiled correctly and the words “Hello World” were seen on the screen at run-time, we took that the programs needed no more testing. The most difficult part of the testing was making sure the spelling was correct and that the loop actually executed exactly 10 times!

Exercise 3: Even / Odd recognition

Design

The goal of this program was to show on the screen two columns side by side; one column to contain the numbers 1 to 10, and the second column to show whether the number in the first column was odd or even. In mathematics, one way of defining an even number is that it leaves remainder zero on division by 2. Similarly, a way of defining an odd number is that it leaves remainder one on division by 2. We used the above two facts when designing our program.

The main method of our program basically initialised a for loop that counted from one to ten, using the command `for(int i=1; i<11; i++)`. Inside this loop, the first thing that we did was output the number “i” i.e. the counting variable, to the screen. Using the above mathematical facts, we then define a new variable “z” to contain the remainder of the counting variable “i” on division by 2. We do this using the modulus operator, %.

```
int z = i % 2;
```

It is interesting to note that the Symantec Cafe help file cites the command “mod” as the one to use when finding the modulus of a number on division by a number. It makes no reference to the fact that the correct syntax to use is “z = a % b”, not “z = a mod b” as suggested in the help file. (where z, a and b are any e.g. integer variables)

We then perform a simple test on z to see whether the modulus was zero or one. IF the modulus was one i.e. greater than zero, then we know that “i” was currently odd, and we output the letter “N” to the screen. Otherwise, when the modulus or the variable z is zero, this means that “i” is currently even and we output the letter “Y” to the screen.

```
if (z > 0)
{
    System.out.println("N");
}
else System.out.println("Y");
```

Testing

The testing strategy for this program and all others was as follows: First, after typing in the draft code, we ironed out any errors generated when the program was compiled. These included typing errors such as in `System.out.println("Hello");`, syntax errors such as using the incorrect amount of brackets or not including a semicolon where it should be, and errors such as using incorrect variable types.

Next, after the program compiles correctly, we find and correct any logical errors in our code. These would be errors where the program doesn't do what it should, and produces nonsensical or incorrect output.

In order to do this, the program's objectives need to be set out clearly, such as the list below for this program:

- (1) *The program needs to output **10** numbers in a column on the screen*
- (2) *Alongside each number must be a letter, either "N" or "Y"*
- (3) *If the number is even, then the letter must be "Y". If it is odd, then the letter must be "N".*

Using the objectives, we run the program and analyse if the actual output of the program matches the expected output. In a program like this, it is only important to achieve the correct output on screen. In programs where e.g. user interaction and calculations on user input is needed, more stringent testing is needed where we input specific values that are more likely to cause the program to crash or produce incorrect output. When testing a program, we try to explore any avenue that an end user may take, and try to make the program as reliable, correct and robust as possible.

The following is the output gained from this program. As you can see, the output matches the expected output as defined in the program objectives.

```
M:\e2027\Assignment 1>java EvenOdd
Symantec Java! JustInTime Compiler Version 210.050 for JDK 1.1
Copyright (C) 1996-97 Symantec Corporation
1 N
2 Y
3 N
4 Y
5 N
6 Y
7 N
8 Y
9 N
10 Y
```

Exercise 4: the “getSquare” program

Design

This program required the programmer to create two methods in his program. The main method, of course, and a separate method called “getSquare” that squares an input parameter and returns the value. This is the method that was used:

```
public static int getSquare(int i) // takes an integer input parameter
{
    i = i*i;
    return i;
}
```

When the method is called, we pass in an input parameter that is squared and returned to the method call. For example, if the following code was in the main method: `int result = getSquare(5);` , then the variable “result” would contain the number 25 after the code executes.

The way that this works is that the getSquare method takes a value when it is called, an integer value denoted by “i”. Inside the method, we change this value to the square of itself, using the code `i = i*i;` (i becomes i²). We then return the value i, as we have to because the method is a public static int.

Testing

In the main method, we have decided that we wanted to square 10 values to test the program more comprehensively. Again we used a loop to do this, which you can see below along with the output the program produces.

```
for (int count=1; count<11; count++)
{
    System.out.print("The Square of " +
                    count + " is ");
    int result = getSquare(count);
    System.out.println(result);
}
```

The Square of 1 is 1
The Square of 2 is 4
The Square of 3 is 9
The Square of 4 is 16
The Square of 5 is 25
The Square of 6 is 36
The Square of 7 is 49
The Square of 8 is 64
The Square of 9 is 81
The Square of 10 is 100

Analysing the output, we see that the program indeed squares values correctly and presents the output in a user-friendly way. The method that a program presents output is important in programming. If the output was just e.g. `6 36`, the chances are that an end user wouldn’t know what those two numbers meant. With some short explanation, we attach meaning to the numbers.

Summary

- ❶ We have written code to solve the first four exercises from the E2027 Assignment 1 instruction sheet
- ❷ We have tested the programs to make sure they produce the correct output, as specified in the aforementioned sheet.
- ❸ We have inserted comments in the source code to improve readability
- ❹ We have produced this report

Conclusions

The preceding four programs use simple loops and code to solve an array of problems varying from simple text output to slightly more complicated calculation output. The programs are short, easy to test once they compile correctly, and simple to use, requiring no user intervention. Programs 5 and 6 are more complicated, and you can see first versions of these programs in the appendix. In summary, we would say that the collection of programs deserve the title of “language refresher”.

Appendix: Source Code and MS-DOS Output

Exercise 1: Write a program to print the following message to the screen: “Hello World”.

Program Code

```
// A program to print out the words 'Hello World'  
  
public class Hello  
{  
    public static void main(String args[])  
    {  
        System.out.println("Hello World!");  
    }  
}
```

MS-DOS Output

```
M:\e2027\Assignment 1>java Hello  
Symantec Java! JustInTime Compiler Version 210.050 for JDK 1.1  
Copyright (C) 1996-97 Symantec Corporation  
Hello World
```

Exercise 3: Write a program that displays the numbers 1 to 10 in a column on the screen. Along side each number print a 'Y' if the value is even and a 'N' if the value is odd.

Program Code

```
/*
 *
 * Gareth Evans, Ian Roberts, Ioan Williams
 *
 * Started: 27th September 1999
 * Finished: 27th September 1999
 *
 * Exercise 3
 *
 * Write a program to display number in a column
 * and print alongside whether they are odd or even
 *
 */

public class EvenOdd
{
    public static void main(String args[])
    {
        for(int i=1; i<11; i++)    // a loop that executes 10 times
        {
            System.out.print(i + " ");

            int z = i % 2;
            // Let z be the variable that holds the remainder of i on division by two

            if (z > 0) // if the remainder is not zero then the number is odd
            {
                System.out.println("N");
            }
            else System.out.println("Y");
        }
    }
}
```

MS-DOS Output

```
M:\e2027\Assignment 1>java EvenOdd
Symantec Java! JustInTime Compiler Version 210.050 for JDK 1.1
Copyright (C) 1996-97 Symantec Corporation
1 N
2 Y
3 N
4 Y
5 N
6 Y
7 N
8 Y
9 N
10 Y
```

Exercise 4: Write a program that includes both a “main” method and a method called “getSquare”. Use the “getSquare” method to calculate the square of a number passed as a parameter from the main method.

Program Code

```
/*
 *
 * Gareth Evans, Ian Roberts, Ioan Williams
 *
 * Started: 27th September 1999
 * Finished: 27th September 1999
 *
 * Exercise 4
 *
 * Write a program to square some numbers
 *
 */

public class Square
{
    public static int getSquare(int i) // takes an integer input parameter
    {
        i = i*i;
        return i;
    }

    public static void main(String args[])
    {
        for (int count=1; count<11; count++) // I will square 10 numbers
        {
            System.out.print("The Square of " + count + " is ");
            int result = getSquare(count);
            System.out.println(result);
        }
    }
}
```

MS-DOS Output

```
M:\e2027\Assignment 1>java Square
Symantec Java! JustInTime Compiler Version 210.050 for JDK 1.1
Copyright (C) 1996-97 Symantec Corporation
The Square of 1 is 1
The Square of 2 is 4
The Square of 3 is 9
The Square of 4 is 16
The Square of 5 is 25
The Square of 6 is 36
The Square of 7 is 49
The Square of 8 is 64
The Square of 9 is 81
The Square of 10 is 100
```

Exercises 5/6: Early versions

Design

On the next two pages you will see the code that we have produced in order to try to solve the problems in exercises 5 and 6. The programs work, although further development is needed to iron out glitches and to make the program work more efficiently.

Our main problem when writing the program was that code written for the integer squaring in exercise four didn't seem to translate well to byte, float and double squaring in this exercise. Also, where we returned an integer in exercise four, here we couldn't return a single value, and had to use a String to return the complete set of results. Another problem was that code such as `ByteSquare = ByteSquare*ByteSquare;` didn't want to compile, and returned errors complaining about "explicit cast needed" and so on.

In exercise 4, we got around these problems by writing a method that took four input parameters and returned a String containing the squares of these four input parameters.

```
public static String getSquare(int IntSquare, byte ByteSquare,
                               float FlSquare, double DoSquare)
{
    String ReturnString = new String();
    ReturnString = (IntSquare*IntSquare + ", " + ByteSquare*ByteSquare
                   + ", " + FlSquare*FlSquare + " and " + DoSquare*DoSquare);
    return ReturnString;
}
```

In exercise 5, the same sort of code was implemented, except that a fifth input parameter was introduced to denote the power, and that we needed four internal variables to store the values needed to be returned in the return String. The following code was used to perform the mathematical process of raising a number to an n^{th} power (Using the notion that $x^n = x \times x \times x \times \dots \times x$ (n times)).

```
for(int i=1; i<=Power; i++)
{
    IntSquare=IntSquare*IntTemp;
    ByteTemp2=ByteTemp*ByteTemp2;
    FlSquare=FlSquare*FloatTemp;
    DoSquare=DoSquare*DoubleTemp;
}
```

Note: for some reason, we needed to convert the byte to an integer to perform the process shown above. This will be investigated when developing the program further.

In conclusion, the following programs are at an early stage of development, but manage to meet their goals, albeit in a contrived sort of way.

Exercise 5: Modify the program in (4) to square values represented as a byte, int, float or double.

Program Code

```
/*
Written by  Ian C Roberts
           Gareth Alun Evans
           Ioan B Williams

Date       30-09-99
Description This is a program that can obtain the
           square of a number whether it's an
           integer, float, or double
*/

public class DataType
{
    public static String getSquare(int IntSquare, byte ByteSquare,
                                   float FlSquare, double DoSquare)

    // input 4 variables: an Integer, a Byte, a Float and a Double

    {
        String ReturnString = new String();

        ReturnString = (IntSquare*IntSquare + ", " + ByteSquare*ByteSquare
                       + ", " + FlSquare*FlSquare + " and " + DoSquare*DoSquare);

        // The String above now contains the four input variables squared,
        // and separated by commas

        return ReturnString;
    }

    public static void main(String args[])
    {
        // Create some sample values

        int    IntSquare  = 2;
        byte   ByteSquare = 16;
        float  FlSquare   = 200;
        double DoSquare   = 16384;

        System.out.print("The squares of " + IntSquare + ", " + ByteSquare + ", ");
        System.out.print(FlSquare + " and " + DoSquare + " are ");
        System.out.println(getSquare(IntSquare, ByteSquare, FlSquare, DoSquare));
    }
}
```

MS-DOS Output

```
M:\e2027\Assignment 1>java DataType
Symantec Java! JustInTime Compiler Version 210.050 for JDK 1.1
Copyright (C) 1996-97 Symantec Corporation
The squares of 2, 16, 200.0 and 16384.0 are 4, 256, 40000.0 and 2.68435456E8
```

Exercise 6: Modify the program in (5) to raise values to an arbitrary power. The power can be another method parameter.

Program Code

```
/*
  Written by   Ian C Roberts
              Gareth Alun Evans
              Ioan B Williams

  Date        30-09-99
  Description  This is a program that can obtain the
              nth power of a number whether it's an
              integer, float, or double
*/

public class DataTypePower
{
    public static String getSquare(int IntSquare, byte ByteSquare,
                                   float FlSquare, double DoSquare, int Power)

    // 5 input parametres: 4 numbers, 1 power number
    {
        String ReturnString = new String();

        // The following variables are needed in the multiplication process
        // to store the value of the original input variable
        // Two "Byte" variables needed because of an 'explicit cast' error

        int IntTemp = IntSquare;
        int ByteTemp = ByteSquare;
        int ByteTemp2 = ByteTemp;
        float FloatTemp = FlSquare;
        double DoubleTemp = DoSquare;

        // In the following loop we multiply the input parametres by themselves
        // a number of times determined by the 'power' input parameter

        for(int i=1; i<=Power; i++)
        {
            IntSquare=IntSquare*IntTemp;
            ByteTemp2=ByteTemp*ByteTemp2;
            FlSquare=FlSquare*FloatTemp;
            DoSquare=DoSquare*DoubleTemp;
        }

        ReturnString = (IntSquare + ", " + ByteTemp2 + ", " + FlSquare + " and " + DoSquare);
        return ReturnString;
    }

    public static void main(String args[])
    {
        // create some values

        int    IntSquare   = 2;
        byte   ByteSquare  = 16;
        float  FlSquare    = 20;
        double DoSquare    = 100;
        int    Power       = 5;

        System.out.print("These four numbers, " + IntSquare + ", " + ByteSquare + ", ");
        System.out.println(FlSquare + " and " + DoSquare + " raised to the power " + Power + " are ");
        System.out.println(getSquare(IntSquare, ByteSquare, FlSquare, DoSquare, Power));
    }
}
```

MS-DOS Output

```
M:\e2027\Assignment 1>java DataTypePower
Symantec Java! JustInTime Compiler Version 210.050 for JDK 1.1
Copyright (C) 1996-97 Symantec Corporation
These four numbers, 2, 16, 20.0 and 100.0 raised to the power 5 are
64, 16777216, 6.4E7 and 1.0E12
```