Western Engineering

WESTERN UNIVERSITY
ELECTRICAL & COMPUTER ENGINEERING

ES1036B
PROGRAMMING FUNDAMENTALS FOR ENGINEERS

WINTER 2019

Lab03- Flow Control and Loop Statements

Instructor:
DR. QUAZI RAHMAN

Prepared by:
RILEY BLOOMFIELD

1 Deliverables

To earn full marks for this assignment you must submit the following file(s) on OWL before 11:55PM on the day of your lab session:

- Username_lab03_q1.java
- Username_lab03_q2.java
- Username_lab03_q3.java

where *username* is the beginning part of your email (before @uwo.ca).

It is mandatory that you submit the .java file(s) containing the high-level Java code you have typed. You may see other files types (such as .class) that are generated as you complete the lab but these files will not be accepted.

Additionally, it is **mandatory** that you demonstrate your codes to your TA and obtain their approval before leaving the lab to earn your marks. Codes completed and submitted without a demonstration will not be graded.

Because codes must be demonstrated to earn marks, it is expected that you will enter the lab session with working or almost working codes that can be completed before the end of the session. If you have any difficulties with the labs during the week, you are encouraged to contact your TA for help.

2 Introduction

This lab assignment will explore flow control of a program in Java and that will allow us to create non-linear programs with execution branches.

If any of the instructions presented are not clear, please do not hesitate to contact your lead teaching assistant Riley Bloomfield at rbloomf@uwo.ca.

2.1 Conditional Expressions

Previously we have always thought of code as a linear progression of commands. The first command at the top of our file executes first, and the program counter steps down with each line executing sequentially, top to bottom. In more complicated programs, program paths diverge based on different inputs or events. Only certain blocks of code are executed if a condition is true or not. The code sample below uses an *if* control flow statement to distinguish if a code block will execute based on a condition. The condition adjacent to the *if* keyword is the conditional expression used to determine the program flow. For more details on the *if* statement, please see lecture unit 3, slides 6-10.

```
public static void main ( String [] args ) {
    // Declare and initialize an integer variable
    int number = 10;
    // Syntax of an if statement
    if ( number > 0) {
        System.out.println("Number is positive.");
    }
    System.out.println("This statement is always executed.");
}
```

Building on this logic, we can chose one path **or** the other using the *if-else* statement. For more details on this statement, please see lecture unit 3, slides 8-13.

2.2 Switch Statements

We have previously covered using **if-else** statements to control the flow of our program's execution. Using many of these statements can become messy, and the **switch** statement can be used to cleanly present multiple paths for execution. The syntax for a **switch** statement in Java is as follows:

```
    public static void main(String[] args) {
        int month = 8;
        String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
            case 7:  monthString = "July";
                     break;
            case 8:  monthString = "August";
                     break;
            case 9:  monthString = "September";
                     break;
            case 10: monthString = "October";
                     break;
            case 11: monthString = "November";
                     break;
            case 12: monthString = "December";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
        System.out.println(monthString);
    }
```

In this example, ”August” will be printed. This example was taken from the Java documentation site and should be referenced for more information concerning switch statements.
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

Pay close attention to the **break** statements inside each case. If this statement is missing, once a case is executed the program will *fall through* into the next case until a break is hit.

More examples of control flow can be seen in lecture unit 3, slides 16, 18, 19-21, 24-25, 28, 30, 32, 45-46, 53-55.

2.3 Loop Structures

An introduction to loop structures can be seen in lecture unit 3, slides 71-84. Keep in mind while learning new looping statements that the can all be used to perform the same tasks, only with different syntax and organization.

2.3.1 While Loop

A *while* statement is a common loop structure used to repeatedly perform a block of code. See the code below that the print statement on line 4 will be printed multiple times according to the entrance condition using a `while` loop. See if you can spot a problem with program execution that will occur when this program is run.

```
1 public static void main(String[] args) {
      int a = 5;
3     while(a < 10) {
          System.out.println("a = " + a);
5     }
      System.out.println("bye");
7 }
```

Note that the condition at the start of the while loop will determine if execution will enter the associated code bock and is also checked after every iteration to check if execution should leave the loop.

2.3.2 Do-While Loop

For details on the *do-while* statement, please review lecture unit 3, slides 106-109. This statement is similar to the `while` statement except that one iteration of the loop is executed before the associated condition is examined, so a single iteration will always be iterated even if the condition is false. The syntax is also slightly different. See the example below from slide 108 showing an example of input validation.

```java
public static void main(String args[]){
    int x;
    Scanner input = new Scanner(System.in);
    do {
        System.out.println("Enter an integer between 3 and 30:");
        x = input.nextInt();
    } while(!((x>=3)&&(x<=30)));
    System.out.println("You've entered: "+ x);
    input.close();
}
```

2.3.3 Validation Using While Loops

We can use this loop structure to repeatedly ask the user to input a correct value if we specify that inputs must be in certain ranges. This lab will explore using loops to validate integer inputs to ensure they are within specific ranges. Please review this validation code taken from lecture unit 3, slide 90 that validates a correct character is input to the program.

```java
public static void main (String[] xyz) {
    System.out.print("Enter your choice: ");
    Scanner input = new Scanner(System.in);
    char choice = input.next().charAt(0);
    choice = Character.toLowerCase(choice);

    while(!(choice=='y' || choice =='n')) {
        //same as: while(choice!='y' && choice !='n')
        System.out.print("Invalid choice! enter again: ");
        choice = input.next().charAt(0);
        choice = Character.toLowerCase(choice);
    }
    System.out.println("you entered " + choice);
    input.close();
}
```

3 Good Programming Practices

Below are several programming practices that should be followed in order to write and maintain quality codes.

- It is proper convention for class names to begin with a capital letter. Make sure that your class names are always capitalized. The naming convention of the submitted files has been changed, please see the section above (section 1).
- Include comments in your program: This will help you remember what each part of your code does, especially after long breaks from your work.
- Choose meaningful and descriptive names for your variables. There is a balance between descriptive names and code readability but always err on the side of descriptive.
- It is recommended that you follow a lower camel-case naming strategy for variables. Since descriptors cannot contain white-space characters (spaces or tabs) words should be separated by capital letters.
- Initialize variables when declaring them. This means giving them initial values which are easier to track in your program if logical errors are present with your output.
- Indent and properly format your code. You should write your codes so that you and your teaching assistant can read and understand your code easily: You will be marked on this.

4 Lab Assignment Questions

You must complete the three assignment questions below for full credit. For each question, create a new class file named with the appropriate naming convention outlined at the start of this lab.

Question 1 - Find the Largest Integer

4.0.1 Requirement

Write a program (according to the specifications outlined below) finds the largest of four integers input by the user.

4.0.2 Specifications

- Your program will begin by prompting the user to input four integer values for comparison.
- Conditional logic using *if-else* and *switch* statements must be used to find the largest of the four integer variables assigned by the user.

- You **may not** use arrays (covered later in this course) or any sorting or maximum-finding Java library methods for this question.
- For each integer input by the user, you must use a *while* or *do-while* loop statement to ensure the value is between **-1000** and **1000** inclusive and your program will continuously ask the user to input a valid integer until they have correctly input one.
- Your validation should also ensure that all user inputs are not divisible by 17 or 23.

4.0.3 Hints

- You may find it useful to *nest* conditional statements by first checking an exterior condition and placing *if-else* statements inside another with interior conditions. This is not required but may be helpful.

4.0.4 Sample Output

Console Output

```
Please enter the first integer:
1005
Incorrect input, please enter a value in the range
of -1000 to 1000 inclusive:
5
Please enter the second integer:
9
Please enter the third integer:
-2000
Incorrect input, please enter a value in the range
of -1000 to 1000 inclusive:
-5000
Incorrect input, please enter a value in the range
of -1000 to 1000 inclusive:
12
Please enter the third integer:
7

The largest integer input is 12.
```

Console Output

```
Please enter the first integer:
5
Please enter the second integer:
9
Please enter the third integer:
12
Please enter the third integer:
7

The largest integer input is 12.
```

Question 2 - Palindrome Calculator

4.0.5 Requirement

Write a program (according to the specifications below) that will ask the user to input an integer value and divisor and will output all palindromic numbers between 1 and the value input divisible by the input divisor.

4.0.6 Specifications

- Your program must begin by asking the user to input an integer value in the range 1 to 9999 inclusive. You must validate this input using a loop structure and continuously ask the user to input a valid range maximum until the input satisfies the condition.
- You should then ask the user to input a divisor between 1 and 150 inclusive. You must validate this input using a loop structure and continuously ask the user to input a valid range maximum until the input satisfies the condition.
- Once both integers have been input, your program should enter a loop that iterates through all integers from 1 to the maximum range value input by the user. On each iteration of the loop, you should check if the current loop counter is palindromic.
- Every palindrome that is found that is also divisible by the input divisor should be printed to the console as a space-separated list (see sample output for more details).

4.0.7 Hints

- Palindromic numbers represent the same value forwards as they do backwards. For example, 151 is a palindrome, as are 1, 66, 8, 999, 919, 12321, 5789875, etc.
- Remember to increment your loop counter at the beginning or end of the loop. If the loop condition allows execution to enter a loop but nothing inside the loop changes the truthfulness of the condition, an infinite loop may be entered.

- An example on finding string palindromes is shown in lecture unit 3, slides 153-154. You should also review previous lab questions on separating integer digits to solve this problem if you're stuck.

4.0.8 Sample Output

Console Output

```
Please enter an integer between 1 and 9999 inclusive:
789
Please input an additional divisor between 1 and 150 inclusive:
22

The palindromes are:
22  44  66  88  242  484  616
```

Console Output

```
Please enter an integer between 1 and 9999 inclusive:
99999
Invalid input, please enter an integer between 1 and 9999 inclusive:
0
Invalid input, please enter an integer between 1 and 9999 inclusive:
-4
Invalid input, please enter an integer between 1 and 9999 inclusive:
999
Please input an additional divisor between 1 and 150 inclusive:
0
Invalid input, please enter an integer between 1 and 150 inclusive:
-543
Invalid input, please enter an integer between 1 and 150 inclusive:
151
Invalid input, please enter an integer between 1 and 150 inclusive:
22

The palindromes are:
22  44  66  88  242  484  616  858
```

Console Output

```
Please enter an integer between 1 and 9999 inclusive:
543
Please input an additional divisor between 1 and 150 inclusive:
3

The palindromes are:
3  6  9  33  66  99  111  141  171  222  252  282  303  333  363
393  414  444  474  525
```

Question 3 - Integer Statistics

4.0.9 Requirement

Write a program that calculates statistics on a set of integer inputs according to the specifications below.

4.0.10 Specifications

- Print a header to the console similar to the one shown in the sample output.
- First the program will ask the user to input an integer size for the set of integers they would like to use for computation (the number of integers they wish to input in the next step). Validate that the entered integer number is greater than 0. Assume that the user will only enter integer number.
- Your program must use a loop structure (while or do-while) where the user is asked to input a new integer on each iteration, until the total number of integers have been input by the user. You can assume that the user will enter integer values and no validation is required.
- After each iteration, the program will print: the mean, count of even integers, and count of odd integers, minimum value, and maximum value input thus far to the console.
- Mean will be computed as according to the equation,

  $$\mu = \frac{\sum_{n=1}^{N} x_n}{N}$$

  where the numerator is the sum of all integers input and the denominator is the total count of integers input.For example, if the user enters 3 integers, the mean will be = (x1+ x2+ x3)/3, where x1, x2 and x3 are the inputted values. Note that the mean value must be computed accordingly to accommodate real number results.

4.0.11 Sample Output

Console Output

```
------------------------------
Student Name
Lab #3
Date:
Question #2
------------------------------

Enter an integer number for the number of loop iterations: 3
Enter integer1: 2
The mean of 1 input(s) is: 2
Max value: 2
Min value: 2
Even count: 1
Odd count: 0

Enter integer2: -2
The mean of 2 input(s) is: 0
Max value: 2
Min value: -2
Even count: 2
Odd count: 0

Enter integer3: 53
The mean of 3 input(s) is: 17.6667
Max value: 53
Min value: -2
Even count: 2
Odd count: 1

Goodbye!
```