LAB 2: METHODS

1. OBJECTIVE

The objectives of Lab2 are to practise on Java methods.

2. INTRODUCTION

A method is an independent collection of source code designed to perform a specific task. By dividing a problem into sub_problems and solve the sub_problems by methods, we obtain a program which is better structured, easier to test, debug and modify. We will practise on writing methods in Java in this lab.

3. *Your Tasks for this LAB*

3.1 In your preferred directory with sub-directory *lab2*, creatre and save the source code into the file Lab2p1.java, and generate the compiled class code as Lab2p1.class.

You may use the program template in Figure 1 to test your methods developed in this lab. The program contains a **main()** which includes a switch statement so that the following methods can be tested by the user. Write the code for each method and use the suggested test cases to test your code for correctness.

```
import java.util.Scanner;
public class Lab2p1 {
  public static void main(String[] args)
  {
    int choice;
    Scanner sc = new Scanner(System.in);
    do {
          System.out.println("Perform the following methods:");
          System.out.println("1:  miltiplication test");
          System.out.println("2:  quotient using division by subtraction");
          System.out.println("3:  remainder using division by subtraction");
          System.out.println("4:  count  the  number of digits");
          System.out.println("5:  position of a digit");
          System.out.println("6:  extract all odd digits");
          System.out.println("7:  quit");
          choice = sc.nextInt();

          switch (choice) {
            case 1: /* add mulTest() call */
                  break;
            case 2: /* add divide() call */
                  break;
            case 3: /* add modulus() call */
                  break;
            case 4: /* add countDigits() call */
                  break;
            case 5: /* add position() call */
                  break;
            case 6: /* add extractOddDigits() call */
                  break;
```

```
                case 7: System.out.println("Program terminating ….");
            }
        } while (choice < 7);
    }

    /* add method code here */

}
```

Figure 1: Program template for Lab 2.

3.2 Write a method that is to test students ability to do multiplication. The method will ask a student 5 multiplication questions one by one and checks the answers. The method prints out the number of correct answers given by the student. The method *random()* from the *Math* class of the Java library can be used to produce two positive one-digit integers (i.e. 1,2,3,4, …) in each question. A sample screen display when the method is called is given below:

```
How much is 6 times 7? 42
How much is 2 times 9? 18
How much is 9 times 4? 36
……..
4 answers out of 5 are correct.
```

The input which is underlined is the student's answer to a question. The method header is:

```
public static void mulTest()
```

Test cases: (1) give 5 wrong answers; (2) give 1 correct answer; (3) give more than 1 correct answer.

Expected outputs: straightforward.

3.3 Write the method divide() which does division by subtraction and returns the quotient of dividing m by n. Both m and n are positive integers (i.e. 1,23,4,…). Division by subtraction means that the division operation is achieved using the subtraction method. For example, divide(12,4) will be performed as follows: 12-4=8, 8-4=4, and then 4-4=0, and it ends and returns the result of 3 as it performs three times in the subtraction operation. No error checking on the parameters is required in the method. The method header is given below:

```
public static int divide(int m, int n)
```

Test cases: (1) m = 4, n = 7; (2) m = 7, n = 7; (3) m = 25, n = 7.

Expected outputs: (1) 4/7 = 0; (2) 7/7 = 1; (3) 25/7 = 3.

3.4 Write the method modulus() which does division by subtraction and returns the remainder of dividing m by n. Both m and n are positive integers. No error checking on the parameters is required in the method. The method header is given below.

```
public static int modulus(int m, int n)
```

Test cases: (1) m = 4, n = 7 (2) m = 7, n = 7 (3) m = 25, n = 7.

Expected outputs: 4 % 7 = 4; (2) 7 % 7 = 0; (3) 25 % 7 = 4.

3.5 Write a method to count the number of digits for a positive integer (i.e. 1,2,3,4,…). For example, 1234 has 4 digits. The method countDigits() returns the result. The method header is given below:

```
public static int countDigits(int n)
```

Test cases: (1) n : -12 (give an error message); (2) n : 123; (3) n : 121456;

Expected outputs: (1) n : -12 - Error input!! (2) n : 123 - count = 3; (3) n : 121456 - count = 6.

3.6 Write the method position() which returns the position of the first appearance of a specified digit in a positive number n. The position of the digit is counted from the right and starts from 1. If the required digit is not in the number, the method should return -1. For example, position(12315, 1) returns 2 and position(12, 3) returns -1. No error checking on the parameters is required in the method. The method header is given below:

```
public static int position(int n, int digit)
```

Test cases: (1) n : 12345, digit : 3; (2) n : 123, digit : 4; (3) n : 12145, digit : 1;

Expected outputs: (1) position = 3; (2) position = -1; (3) position = 3.

3.7 Write a method extractOddDigits() which extracts the odd digits from a positive number n, and combines the odd digits sequentially into a new number. The new number is returned back to the calling method. If the input number n does not contain any odd digits, then returns -1. For examples, if n=1234567, then 1357 is returned; and if n=28, then –1 is returned. The method header is given below:

```
public static long extractOddDigits(long n)
```

Test cases: (1) n : 12345; (2) n : 54123; (3) n : 246; (4) n : -12 (give an error message)

Expected outputs: (1) oddDigits = 135; (2) oddDigits = 513; (3) oddDigits = -1; (4) oddDigits = Error input!!