COMP1102/8702 – Practical Class 6 – Selection and Iteration

Aims and Objectives

This laboratory has been designed to help you

- design and modify basic classes
- make use of alternative selection and iteration statements

Start *IntelliJ* and open the project named "Practical06" (download if from FLO).

Task 1

1. `Counter` objects can print sequences of integers. Each time the method `countUp` is called, a sequence is output, starting from the next number. Study the program and its output given below and ensure you understand how it works.

```
public class Number {

   public static void main (String[] args) {
      System.out.println("Starting Application...") ;
   // Create a new Counter object and store a reference to it in
   // the newly declared variable c:
      Counter c = new Counter(1);
   // Invoke the method countUp from the object referred to by c
   // with an actual parameter of 2
      c.countUp(2);
   // Invoke the method countUp from the object referred to by c
   // with an actual parameter of 3
      c.countUp(3);
   } // end method main

} // end class Number

public class Counter {
// An instance variable to keep track of which number we are up to
   private int currentCount  ;
// The constructor receives an int as a parameter and stores
// it in currentCount
   public Counter(int startValue) {
      currentCount = startValue ;
   }
 // A method to print out the next n numbers
   public void countUp(int n) {
      System.out.println("*** Counting up " + n) ;
      for (int step = 1; step <= n; step++) {
         System.out.println("counter = " + currentCount) ;
         currentCount = currentCount + 1;
      } // end for loop
   } // end method countUp
}  // end class Counter
```

2. Compile and run the program. It should produce the following output:

```
Starting Application...
*** Counting up 2
counter = 1
counter = 2
*** Counting up 3
counter = 3
counter = 4
counter = 5
```

3. Add a method called `countDown` to the class `Counter` which counts down (rather than up as the method `countUp` does).

   Hint: the definition of `countDown` will look very similar to `countUp`. All you need to do is remove the statement which increases the value of `currentCount` and add a statement which decreases `currentCount` - but not in the same place!

4. At the end of the *main* method, use the variable `c` to call to the `countDown` method with an actual parameter of 4.

5. At the end of the *main* method, use the variable `c` to call to the `countUp` method with an actual parameter of 2.

   Compile and run the application. It should produce the output:

```
Starting Application...
*** Counting up 2
counter = 1
counter = 2
*** Counting up 3
counter = 3
counter = 4
counter = 5
*** Counting down 4
counter = 5
counter = 4
counter = 3
counter = 2
*** Counting up 2
counter = 2
counter = 3
```

Checkpoint 27

Have the program source code and output marked by a demonstrator

Task 2

Modify the program developed in Task 1 in the following ways.

1. Modify the `countDown` method so that it will not count down below zero (that is, it will not allow the counter to become negative). This **must** be done by extending the termination condition in the *for*-loop.

2. At the end of the *main* method add a call to the `countDown` method with an actual

parameter of 6, following by a call to the `countUp` method with an actual parameter of 2.

Compile and run the application. It should produce the output:

```
Starting Application...
*** Counting up 2
counter = 1
counter = 2
*** Counting up 3
counter = 3
counter = 4
counter = 5
*** Counting down 4
counter = 5
counter = 4
counter = 3
counter = 2
*** Counting up 2
counter = 2
counter = 3
*** Counting down 6
counter = 3
counter = 2
counter = 1
counter = 0
*** Counting up 2
counter = 0
counter = 1
```

Checkpoint 28

Have the program source code and output marked by a demonstrator

Task 3

Modify the program developed in Task 2 in the following ways.

1. Modify the class `Counter` so that the size of the counting steps can be specified when the object is constructed rather than being fixed at 1.

   Hint: you will need another instance variable to store the size of the counting steps. The last statement of the constructor should print the starting value and the step size (see the example below).

2. Modify the method *main* so that it creates a `Counter` object starting at 1 with a step size of 3. Call the `countUp` method from this object with an actual parameter of 5.

3. Define a second constructor for the `Counter` class which takes a single parameter, the starting value, and sets the step size to 1. The last statement of the constructor should print the starting value (see the example below).

4. Add code to the end of the *main* method to create another `Counter` object, using the new constructor, with a starting value of 5.

5. Call the `countUp` method from this new `Counter` object with an actual parameter of 2.

Compile and run the application. It should produce the output:

```
Starting Application...
Creating Counter object with a starting value of 1 and a step size of 3
*** Counting up 5
counter = 1
counter = 4
counter = 7
counter = 10
counter = 13
Creating Counter object with a starting value of 5
*** Counting up 2
counter = 5
counter = 6
```

Checkpoint 29

Have the program source code and output marked by a demonstrator

Task 4

Modify the program developed in Task 3 in the following ways.

1. Add another formal parameter called `op` of type `char` to the method `countUp`. It specifies how to calculate the next value of the counter according to the following rules:

Value of op	Action
'+'	currentCount = currentCount + increment
'-'	currentCount = currentCount - increment
'*'	currentCount = currentCount * increment
any other value	print "Invalid operation"

2. Change the first call to `countUp` to include the actual parameter `'*'`. Change the second call to `countUp` to include the actual parameter `'-'`. Add a third call to `countUp` to include the actual parameter `'?'` – an invalid operation. The *main* method should now be as follows:

```
public static void main (String[] args)
   System.out.println("Starting Application...") ;
   Counter c = new Counter(1,3) ;
   c.countUp(5,'*') ;
   Counter c1 = new Counter(5) ;
   c1.countUp(2,'-') ;
   c1.countUp(2,'?') ;
```

Compile and run the application. It should produce the output:

```
Starting Application...
Creating Counter object with a starting value of 1 and a step size of 3
*** Counting up 5, operation is *
counter = 1
counter = 3
counter = 9
counter = 27
counter = 81
Creating Counter object with a starting value of 5
*** Counting up 2, operation is -
counter = 5
counter = 4
*** Counting up 2, operation is ?
counter = 3
Invalid operation: ?
```

Checkpoint 30

Have the program source code and output marked by a demonstrator

Task 5 (Extension Practice)

Modify the program developed in Task 4 in the following ways.

1. Modify the class `Counter` by including a method called `printMax` which prints the largest value a counter has reached. This may not be its current value. You will need to store the maximum value to be able to print it out!
2. Add a method called `reset` to the the `Counter` class which resets the counter to its original starting value and returns the last value of the counter.
3. Modify the *main* method to be:

```
public static void main (String[] args) {
   System.out.println("Starting Application...") ;
   Counter c = new Counter(1,3) ;
   c.countUp(5,'*') ;
   c.printMax() ;
   System.out.println("Reset returned " + c.reset());
   c.countUp(2,'+') ;
   c.printMax() ;
```

Compile and run the application. It should produce the output:

```
Starting Application...
Creating Counter object with a starting value of 1 and a step size of 3
*** Counting up 5, operation is *
counter = 1
counter = 3
counter = 9
counter = 27
counter = 81
Max = 81
Reset returned 243
*** Counting up 2, operation is +
counter = 1
counter = 4
Max = 81
```