* Don’t forget to set your Eclipse workspace and working set.
* **You must submit the JAR file, exported (with source code), from your Eclipse project.**
* **You must check your JAR file to make sure all the source files (.java files) are present. It can be opened with file compression programs such as 7-zip or Winrar.**
* **Failure to export properly will result in your work not getting marked.** 
  1. **To submit:**
* **Export your project to a JAR file, with source code.**
* **Name your JAR file ID\_Week06\_Q1.jar. For example, 6623110021\_Week06\_Q1.jar**
* **Submit the JAR file on MyCourseville.**

**Lab Stack Q1 (12 marks, will be scaled to equal to other homeworks)**

**Copy all files to your Eclipse project “src” folder.**

You are given all classes for coding a stack.

Class StackArray and StackLinkedList are given.

Write code for the following method in class StackUtility:

**public** **static** String operate(MyStack s1, MyStack s2) **throws** Exception {

This method uses 2 stacks to decode a secret word built from:

**static** String *alphabets* = "abcdefghijklmnopqrstuvwxyz";

s1 contains an incomplete representation of the word to decode. S2 contains the operations to be used on data of s1 to do the decoding. For example:

The following s1 and s2 will produce the word “cat”:

20

-

< 0 indicates operator subtract.

>=0 indicates operator add.

-2

-4

5

-4

-8

7

1

0

+

0

4

-

S2 can have many dummy (not use) frames.

2

4

S2

S1

S1 can have a dummy (not use) frame.

By popping each 2 values from s1 and applying addition/subtraction (according to s2), we can store the result in another stack (say, s3). The above s1 and s2 will produce the following s3.

2

0

19

S3

Each number in s3 indicates a position in alphabets. Hence we get character at 2nd position (c), 0th position (a), and 19th position (t). Thus “cat” will be the returned string from this example.

* Your code must work on any implementation of Stack, or you will lose points.

JUnit is in StackUtilityTest.java

* testWordSimpleAdd() 1 mark
* testWordSimpleSubtract() 1 mark
* testWordSimpleSequence() 1 mark
* testWord01() 3 marks
* testWord02() 3 marks
* testWord03() 3 marks