Date: November 4 8:20 PM

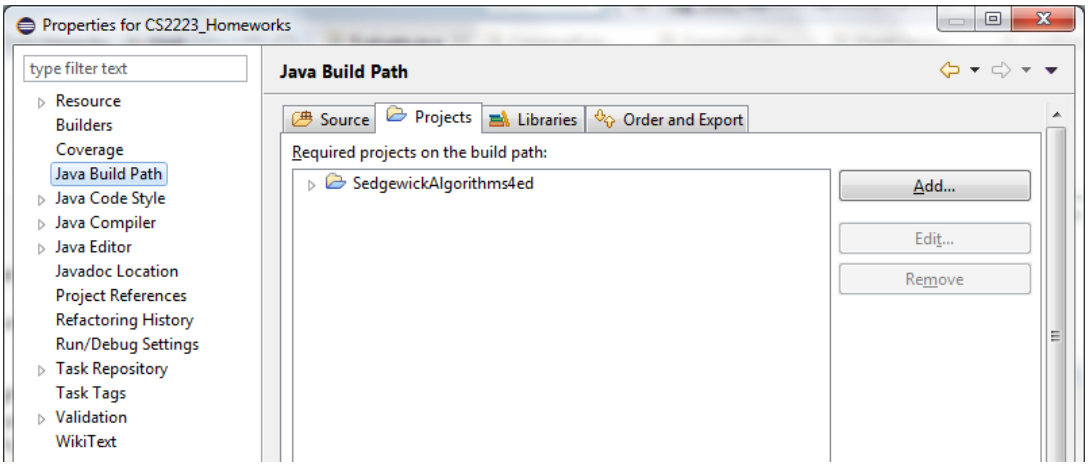
CS 2223 B15 Term. Homework 1 (100 pts.)

Homework Instructions

- This homework is to be completed individually. If you have any questions as to what constitutes improper behavior, review the examples I have posted online http://web.cs.wpi.edu/~heineman/html/teaching_/cs2223/b15/#policies .
- Due Date for this assignment is 2PM November 6th. Homeworks received after 2PM receive a 25% late penalty. Homeworks received after 6PM will receive zero credit.
- Submit your assignments electronically using the blackboard site for CS2223. Login to my.wpi.edu and go to CS2223 under "My Courses" then go to "Assignments" and submit your homework under "HW1". You must submit a single ZIP file that contains all of your code as well as the written answers to the assignment.
- All of your Java classes must be defined in a packager USERID where USERID is your CCC user id.
- Submission information is found at the end of this document.

First Steps

Your first task is to copy all of the files from the Git repository that you will be modifying/using for homework1. First, make sure you have created a Java Project within your workspace. Be sure to modify the build path so you will have access to the shared code I provide. To do this, select this project and right-click to bring up the Properties for the project. Choose the option **Java Build Path** on the left and click the Projects tab. Now **Add…** the SedgewickAlgorithms4ed project to your build path.



Once done, create the package **`USERID.hw1`** inside this project which is where you will complete your work. Start by copying the five source files from **`algs.hw1`** (BinaryStringSearch, ConfirmCrossingPoint, CrossingPoint, Evaluate, FixedCapacityStackOfStrings) and paste them into the USERID.hw1 package.

Date: November 4 8:20 PM

Stack Experiments (30 pts.)

On page 129 of the book there is an implementation of a rudimentary calculator using two stacks for expression evaluation. I have created the `Evaluate` class which you have copied into your **`USERID.hw1`** package. Note that all input (as described in the book) must have spaces that cleanly separate all operators and values.

1. **(5 pts.)** Run this program on input "`( sqrt ( 6 ) )`" and explain the observed output
2. **(5 pts.)** Run this program on input "`2 + 3`" and explain the observed output
3. **(5 pts.)** Run this program on input "`2 + 3 + 4 + 5`" and explain the observed output
4. **(5 pts.)** Run this program on input "`sqrt ( 12 + 4 )`" and explain the observed output
5. **(5 pts.)** Modify `Evaluate` to allow for certain pre-defined mathematical constants. In particular, you must support:
    a. "`pi`" has value of 3.141592653589793 (Note Java defines this as Math.PI)
    b. "`e`" has value of 2.718281828459045 (Note Java defines this as Math.E)
6. **(5 pts.)** Once done, run your program on the input "`( sqrt ( e * pi ) )`" and explain the observed output.

Date: November 4 8:20 PM

Programming Exercise (20 pts.)

A two dimensional (2D) array of **`boolean`** values has a *crossing point* if it consists of **`false`** values everywhere **except** in a single row and a single column that contain **`true`** values. The following, for example, is a 2D array whose crossing point is at (2, 4) because it is in row labeled #2 and column labeled #4 (using zero-based indexing). You can assume that the array has at least two rows and two columns.

	0	1	2	3	4	5	6	7
0	F	F	F	F	T	F	F	F
1	F	F	F	F	T	F	F	F
2	T	T	T	T	T	T	T	T
3	F	F	F	F	T	F	F	F

1. **(10 pts.)** Modify the **`CrossingPoint`** class so that it takes a two-dimensional array of **`boolean`** values and prints to standard output the crossing point indexes. You can assume that the array has at least two rows and two columns. You can assume that the array has a single crossing point, using the above definition.

Your code must conform to the following method signature. You may define a 'main' method to test your code but we will be writing our own tests to validate your code.

```
package heineman.hw1;
public class CrossingPoint {
  public void locate (boolean ar[][]) {
       // your code here…
  }

  // sample client to demonstrate crossing point
  public static void main(String[] args) {
       boolean [][] ar =  {
             { false, false, false, false, true, false, false, false },
             { false, false, false, false, true, false, false, false },
             { true,  true,  true,  true,  true, true,  true,  true },
             { false, false, false, false, true, false, false, false },
       };
       new CrossingPoint().locate(ar);
  }
}
```

The output of your program must be the following where NNN is an integer value

```
Crossing Point: 2,4
Required NNN array lookups
```

2. **(5 pts.)** Assuming that your **`CrossingPoint`** implementation processes an array of M rows and N columns, define exactly the maximum number of array locations `ar[r][c]` that you must inspect to locate the crossing point. That is, for any two-dimensional Boolean array, regardless of its contents, you

want to determine the fewest locations you must inspect in the worst possible case. You should use the terms M and N in your computation. You can assume that the array has a single crossing point, using the above definition.

3. **(5 pts.)** Modify the program **`ConfirmCrossingPoint`** so it confirms the calculation from question 2 on a 5x7 2D array of your choosing.

Stack Exercises (25 pts.)

For this question, review the **`FixedCapacityStackOfString`** class which is a copy of the code which you can find on p. 133 of the book. As you will see, the input to the sample program is a string consisting of a series of **tokens** separated by spaces. The "`-`" string represents a pop operation and all other string tokens are pushed onto the stack.

1. **(5 pts.)** What is the maximum number of array locations ~~affected with~~ inspected by each **`pop`** operation?

2. **(5 pts.)** What is the maximum number of array locations ~~affected with~~ inspected by each **`push`** operation?

For this next question, assume that you create a **`FixedCapacityStackOfString`** stack of size N. Further assume that all **tokens** in the input string are either a "`-`" or a single lower case letter. Thus "`a b - c -`" would be a valid input.

Finally, assume you are given a properly formatted input string containing 3N tokens (or in other words it has 3N-1 spaces and the total length of the string is 6N-1 characters).

3. **(5 pts.)** What is the maximum number of consecutive "`-`" tokens that can appear in the input string to avoid throwing an Exception or having the program print BAD INPUT?

4. **(5 pts.)** What is the minimum number of "`-`" tokens that must appear to avoid throwing an Exception or having the program print BAD INPUT?

5. **(5 pts.)** What is the minimum number of single lower case tokens that must appear to avoid throwing an Exception or having the program print BAD INPUT?

Date: November 4 8:20 PM

Binary Search Exercise (25 pts.)

You have a string S which contains N **unique** tokens in sorted order, separated by spaces. Each of these tokens is exactly K>0 characters in length and is only composed of lower case letters drawn from a standard English alphabet of 26 letters. For example, "add boy cat" would be a valid string with K=3 while "cart fall deal" would not be a valid string for K=4 because the words are not in ascending order.

The total length of the string S is N(K+1)–1 characters. Alternatively, since K is a fixed constant, then given a string S you can determine the number of tokens that it contains as (len(S)+1)/(K+1).

Be inspired by the **`rank`** implementation on page 47 of the book to complete the implementation of the **`BinaryStringSearch`** such that it will return the proper answer by examining just a few tokens in the string (relative to the size of the entire string S). In particular, you should be able to demonstrate that you only need to inspect about $\log_2 N$ tokens.

1. **(15 pts.)** Validate your implementation on sample token strings and be sure to cover the edge cases (i.e., when there is just one token in the string).

2. **(10 pts.)** In the worst case, count the maximum number of **character** comparisons that **`rank`** must perform on any input. This is the worst case analysis. State your answer in terms of K and N.

BONUS QUESTION (5 pts)

Do not attempt this bonus question until you have completed this entire assignment. What if we allow the individual tokens to be of any length greater than 0 and less than or equal to K? For example, if K = 11, the following would be a valid tokens String with N=13 tokens:

```
String target = "a alpha beta charlie david dog drinking elephant fraught me myself neverwhere pontificant";
```

In the worst case, count the maximum number of **character** comparisons that **`rank`** must perform on any input. State your answer in terms of K and N.

BONUS BONUS QUESTION (1 pt)

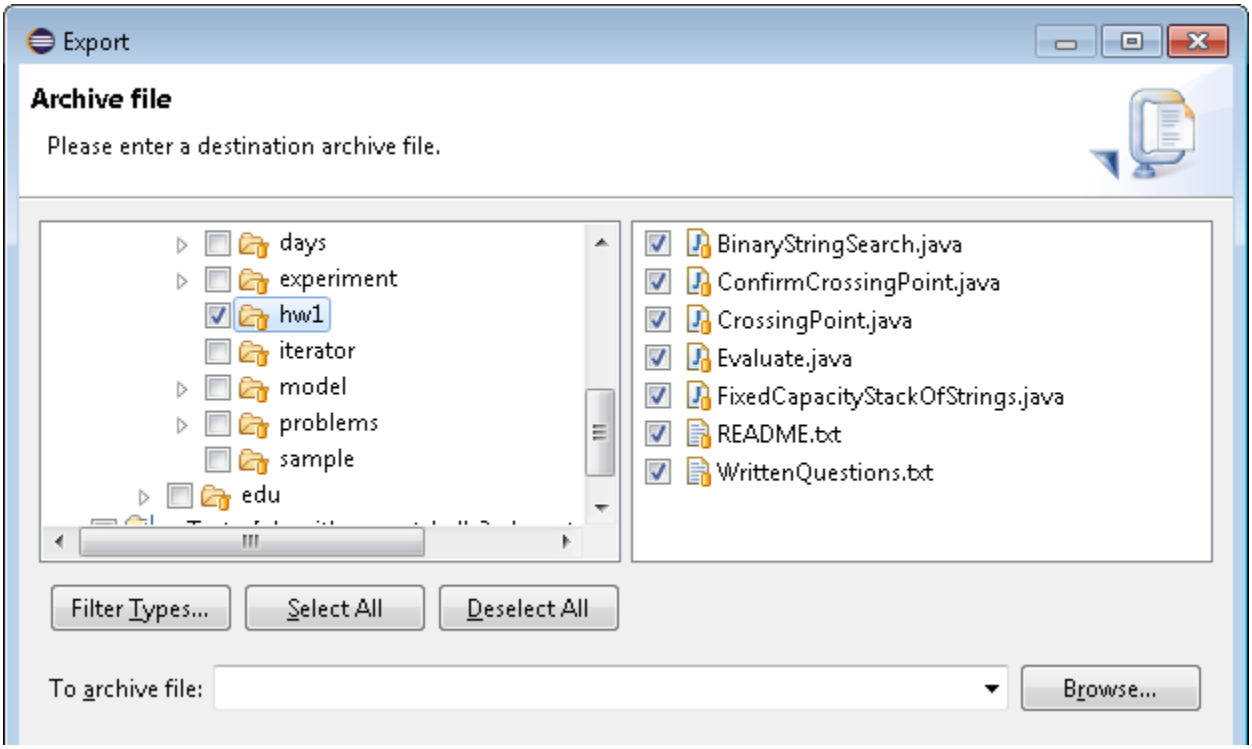
Seriously? You want more? Ok, add extra methods **`remove`** and **`add`** that do the obvious functionality without impairing the ability of this class to perform **`rank`** as intended. Note that the **add** method must maintain set semantics.

Date: November 4 8:20 PM

Submission Details

Each student is to submit a single ZIP file that will contain the implementations. In addition, there is a file "WrittenQuestions.txt" in which you are to complete the short answer problems on the homework.

The best way to prepare your ZIP file is to export your entire **`USERID.hw1`** package to a ZIP file using Eclipse. Select your package and then choose menu item "**Export…**" which will bring up the Export wizard. Expand the **General** folder and select **Archive File** then click **Next**.



You will see something like the above. Make sure that the entire "hw1" package is selected and all of the files within it will also be selected. Then click on **Browse…** to place the exported file on disk and call it USERID-HW1.zip or something like that. Then you will submit this single zip file in my.wpi.edu as your homework1 submission.

Addendum

If you discover anything materially wrong with these questions, be sure to contact the professor or TA/SAs by emailing cs2223-staff@cs.wpi.edu. You can also post to the discussion forum for HW1.

When I make changes to the questions, I enter my changes in red colored text as shown here.