Com S 227
Spring 2021
Miniassignment 1
40 points

Due Date: Monday, October 17, 11:59 pm (midnight)
5% bonus for submitting 1 day early (by 11:59 pm Oct 16)
10% penalty for submitting 1 day late (by 11:59 pm Oct 18)
No submissions accepted after Oct 18, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus for details.

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Canvas. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: This is a miniassignment and the grading is automated. If you do not submit it correctly and we have to run it by hand, you will receive at most half credit.

Overview

This is a short set of practice problems involving writing loops. You will write seven methods for the class `mini1.LostInTheLoop`. All of the methods are static, so your class will not have any instance variables (or any static variables, for that matter). There is a constructor, but it is declared `private` so the class cannot be instantiated.

For details and examples see the online Javadoc. There is a skeleton of the class on Canvas. If you use the skeleton code, be sure you put it in a package called `mini1`.

You do *not* need arrays, or ArrayLists, for this assignment. Alhough you will not be penalized for using them, you are probably making things harder.

Advice

Before you write any code for a method, work through the problem with a pencil and paper on a few concrete examples. Make yourself write everything down; in particular, write down things that you need to remember from one step to the next (such as indices, or values from a previous step). Try to explain what you are doing in words. Write your algorithm in pseudocode. Explain it to your mom. Explain it to your dog. Find a bright fourth grader and explain it to her. *If you can't explain your algorithm so that a fourth grader can follow the steps, then you probably can't get the Java runtime to follow your logic either*.

The other key problem-solving strategy is to remember that you don't have to solve the whole problem in your head all at once. Try solving *part* of the problem, or solving a *related, simpler problem*. You may ultimately have to throw away the code, but you will gain experience and insight. For example:

- If you are working on **`memoryGameChecker`**, can you
    a. Given two strings, iterate over the indices and count the total number of indices where the characters match?
    b. Same as above, but stop when you first encounter an index where the characters don't match?
    c. As above, but skip over the first mismatch?
    d. As above but skip over the first **`maxMistakes`** mismatches?

- If you are working on **`compressRuns`**, can you
    a. Iterate over a string, and print out the characters one at a time?
    b. Same as above, but instead of printing them, append the characters to a result string?
    c. As above, but only append a character if it *doesn't* match the last character of the result string? (*Tip: you'll need to treat an empty string as a special case, so you can always initialize the result string with the first character*)

- If you are working on **`countTriangleNumberSum`**, can you
    a. Write a loop to print out the first 10 triangle numbers?
    b. Write a loop to add up their reciprocals?
    c. Same, then check how close the sum is to 2.0?
    d. Same, but instead of 10, stop the loop when the sum is within **`err`** of 2.0?

- If you are working on **`findHighestScore`**, can you
    a. Write a loop that uses a scanner to parse a string of text and just prints the items out one per line?
    b. Same, but assuming that alternate items are numbers, reads the numbers as **`int`**s
    c. Same, but keeps track of the maximum number and then prints it out
    d. Same, but also keeps track of the string preceding the maximum number
    e. Same, but instead of printing it out, returns the string and number with a space in between

    (Note: a good way to find a maximum of a list is to initialize a variable **`max`** to the first item in the list. Then, whenever you find a larger value, update the value of **`max`**. This is much more reliable than initializing **`max`** to some bogus value.)

- If you are working on **`printTree`**, can you
    a. Write a loop that given n, prints out the number of spaces needed at the start of each row of the tree?
    b. Write a loop that given n, prints out the number of spaces *and* the number of forward slashes?
    c. Write a loop that, given a number **`count`** and a character **`c`**, creates a string by concatenating the character **`c`** with itself **`count`** times?

    (Note: the backslash character '**`\`**' has a special meaning in Java strings as part of "escape sequences" such as newline **`\n`** and quote **`\"`**, so to print an actual backslash, or include one in a string, you have to double it: **`\\ )`**

- If you are working on **`wordGameChecker`**, can you
    a. Given two strings **`g`** and **`s`**, iterate over the indices and print out the indices where the characters match?
    b. Given two strings **`g`** and **`s`**, create a new string of the same length that has a '*' character at each index where the characters match, and a '-' character at indices where the characters don't match?
    c. Given two strings **`g`** and **`s`**, create a new string of the same length that has a '*' character at each index where the characters match, and a '-' character at indices where the character in **`g`** does not occur at all in **`s`**, but has a '?' if the character does occur *somewhere* in **`s`**? (*Tip: to check whether a character* **`ch`** *occurs in a string* **`s`***, you can use the boolean expression* **`s.indexOf(ch) >= 0`**.)

- If you are working on **`cancelAdjacentPairs`**, can you
    a. Iterate over a string and print each character, but only if it doesn't match the one after it (e.g. from "abbc" you'd print a, b, c)
    b. Do the same, but *skip over* the second character in the match (e.g., from "abbc" you'd print a, c) (*Tip: this is actually easiest using a while loop*)
    c. Instead of printing the characters, append them to a result string
    d. Repeat the entire process until no further matches are found

How do I modify a string?

You don't. Strings are immutable. Instead, build up a new string with the characters you want. We saw an example like this in lab to "reverse" a given string:

```
public static String reverse(String s)
{
  String result = "";
  for (int i = s.length() - 1; i >= 0; i -= 1)
  {
    result += s.charAt(i);
  }
  return result;
}
```

> Java does have a class called **`StringBuilder`** which acts as a mutable type of string, but in general it will not make anything easier for you in these problems. (Trying to modify a string while you are iterating over it is problematic.)

My code's not working!!

Developing loops can be hard. Some of the problems in this assignment, although they are all short, are probably hard enough that if you don't have a clear idea of what you *want* the code to do, you will be unable to successfully write code that works. You can waste many, many hours making random changes trying to get something to pass the sample tests. *Please don't do that.*

If you are getting errors, a good idea is to go back to a simple concrete example, describe your algorithm in words, and execute the steps by hand.

If your strategy works when you carry out the steps by hand, and you are confident that your algorithm is right but you are still getting errors, you then have a *debugging* problem – at some point you've coded something that isn't producing the result you intend.

In simple cases, you can verify what's happening in the code by temporarily inserting **`println`** statements to check whether variables are getting updated in the way you expect. (Remember to remove the extra **`println`**'s when you're done!)

Ultimately, however, the most powerful way to trace through code is with the debugger, as we are practicing in Lab 6. Learn to use the debugger effectively, and it will be a lifelong friend.

If you have an infinite loop, please refer to link #13 on our Canvas front page for additional tips.

You have absolute power. Use it!

You really do have absolute, godlike power when it comes to your own code. If the code isn't doing what you want it to do, you can decide what you really want, and make it happen. *You are in complete control!*

> (If you are not sure what you *want* the code to do, well, that's a different problem. Go back to the "Advice" section.)

The SpecChecker

A SpecChecker will posted with a number of functional tests. However, when you are debugging, it is usually helpful if you have a simpler test case of your own.

Remember that to call a static method, you prefix it with the *class* name, not with an object reference. For example, here is simple test case for the **`compressRuns`** method:

```
import mini1.LostInTheLoop;
public class SimpleTest
{
  public static void main(String[] args)
  {
    String result = LostInTheLoop.compressRuns ("abbbc");
    System.out.println(result);
    System.out.println("Expected abc");
  }
}
```

You can save yourself from having to type "**`LostInTheLoop`** " over and over again by using the Java feature **`import static`**:

```
import static mini1. LostInTheLoop.*;

public class SimpleTest2
{
  public static void main(String[] args)
  {
    String result = compressRuns ("abbbc");
    System.out.println(result);
    System.out.println("Expected abc");
  }
}
```

Since no test code is being turned in, you are welcome to post your tests on Piazza for others to use and comment on.

Documentation and style

Since this is a miniassignment, the grading is automated and in most cases we will not be reading your code. Therefore, there are no specific documentation and style requirements. However, writing a brief descriptive comment for each method will help you clarify what it is you are trying to do. Likewise, brief internal comments can help you keep track of what you are trying to do when you write a tricky line of code.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`miniassignment1`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`miniassignment1`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled "pre" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Canvas, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_mini1.zip`**. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, **`mini1`**, which in turn contains one file, **`LostInTheLoop.java`**. Always LOOK in the zip file the file to check.

We strongly recommend that you just submit the zip file created by the specchecker, AFTER CHECKING THAT IT CONTAINS THE CORRECT CODE. ***If you mess something up and we have to run your code manually, you will receive at most half the points.***

Submit the zip file to Canvas using the Miniassignment1 submission link and verify that your submission was successful. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found on the Canvas front page.

> We strongly recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **mini1**, which in turn should contain the file **`LostInTheLoop.java`**. You can accomplish this by zipping up the **src** directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and not a third-party installation of WinRAR, 7-zip, or Winzip.