Com S 227
Spring 2021
Miniassignment 1
50 points

Due Date: Friday, March 19, 11:59 pm (midnight)

5% bonus for submitting 1 day early (by 11:59 pm Mar 18)

10% penalty for submitting 1 day late (by 11:59 pm Mar 20)

No submissions accepted after March 20, 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 eight methods for the class `mini1.LoopaholicsAnonymous`. 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, though you will not be penalized for using them.

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 it 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*. For example:

- If you are working on **`findEscapeCount`**, can you
    a. work by hand through the example calculation given in the Javadoc, and get the same answers?
    b. write a loop that has the correct loop body but always just repeats the steps, say, 5 times?
    c. write a condition to check that $a^2 + b^2 <= 4$?
    d. write a condition to check that $a^2 + b^2 <= 4$ AND the given maxIterations hasn't been reached?

- If you are working on **`findNth`**, can you
    a. iterate over a string and just print the characters one at a time?
    b. iterate over a string and print "boo" each time you find a matching character?
    c. iterate over a string and *count* how many times time you find a matching character?
    d. iterate over a string and print the index, each time you find a matching character?
    e. iterate over a string and print the index, only if it's the Nth time you find a matching character?

- If you are working on **`getRun`**, can you
    a. return a string with just the single character at the given index?

   b. given a character **`ch`** and an index, count how many of the following characters match **`ch`**?
   c. instead of *counting* the matching characters, *append* each one onto your string? (Alternatively, use the **`substring`** method)

- If you are working on **`isArithmetic`**, can you
   a. parse the string and just print the numbers one at a time? (*Tip: use a* **`Scanner`**, *and call* **`useDelimiter(",")`** *on it so that it will recognize input items as separated by commas rather than whitespace*.)
   b. calculate the difference between the first two numbers?
   c. parse the string and calculate the difference between each number and the previous number?
   d. verify whether all those differences are the same?

- If you are working on **`subsequenceWithGaps`**, can you
   a. find the first index of a character in a string?
   b. find the first index of a character but only *after* a specified start position in a string?
   c. iterate over one string, and find the index of each of its characters in the second string?
   d. iterate over one string and find the index of each of its characters in the second string, restricting your search to characters that are *after* the previous one you found?
   (Tip: you might find the String **`indexOf`** *methods useful)*

- If you are working on **`takeApartPreservingRuns`**, can you
   a. iterate over a string, and print out alternate characters?
   b. iterate over a string, and create a pair of strings containing the alternate characters? (e.g., from "abcde" return "ace bd") ?
   c. instead of alternate characters, take alternate groups of matching characters (*Tip: use your* **`getRun`** *as a helper method!*)

- If you are working on **`longestAscendingSubstring`**, can you
   a. write an algorithm to find the length of the longest run of consecutive, matching characters? *(Tip: this was page 9 of lab 5!)*
   b. modify your algorithm so that instead of the longest run of *equal* characters, it finds the longest run of *ascending* characters? (*Tip:* **`char`** *is a primitive type, so you can compare the ordering of* **`char`** *values using the < and > operators.)*

    c. modify your algorithm so that instead of returning the length of the longest run, it returns the starting index of the longest run?
    d. modify your algorithm so that instead of returning the starting index of the longest run, it returns the actual longest run as a String?

- If you are working on **`printStars`**, can you
    a. write a helper method that, given a number d, constructs and returns a String of d dashes?
    b. write a helper method that, given number d and a number n > d, returns a String of the correct form for one line of the output? (For example, given n = 5 and d = 3, return the 10-character string `"---*  *---"`)
    c. write an algorithm to just print out the number of dashes that should appear at the beginning of each line? (For example, if n = 5, you'd print 4, 3, 2, 1, 0, 0, 1, 2, 3, 4)

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 it.*

If you are getting errors, a good idea is to go back to a simple concrete example, and trace execution of your code by hand to see if the code is doing what you want it to do. You can also trace 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, the best way to trace through code 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 #12 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.)

How do I make a string with a loop, as suggested for `getRun` or `printStars` or `takeApartPreservingRuns`?

Start with an empty string and concatenate an additional character in each iteration. For example, here is one way to create the reverse of a given string:

```
  public static String reverse(String s)
  {
    String result = "";  // start with empty string
    for (int i = s.length() - 1; i >= 0; i = i - 1)
    {
      result += s.charAt(i); // add on characters one at a time
    }
    return result;
  }
```

As an aside, experienced Java programmers would probably use a **`StringBuilder`** object:

```
  private static String reverse(String s)
  {
    StringBuilder sb = new StringBuilder();
    for (int i = s.length() - 1; i >= 0; i = i - 1)
    {
      sb.append(s.charAt(i));
    }
    return sb.toString();
  }
```

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 **`getRun`** method:

```
import mini1.LoopaholicsAnonymous;
public class SimpleTest
{
  public static void main(String[] args)
  {
    String result = LoopaholicsAnonymous.getRun("bbbbbbcbc", 2);
    System.out.println(result);
    System.out.println("Expected bbbb");
  }
}
```

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

```
import static mini1.LoopaholicsAnonymous.*;

public class LooperTester2
{
  public static void main(String[] args)
  {
    String result = getRun("bbbbbbcbc", 2);
    System.out.println(result);
    System.out.println("Expected bbbb");
  }
}
```

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, **`LoopaholicsAnonymous.java`**. Always LOOK in the zip file the file to check.

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 in the Piazza pinned messages under "Syllabus, office hours, useful links."

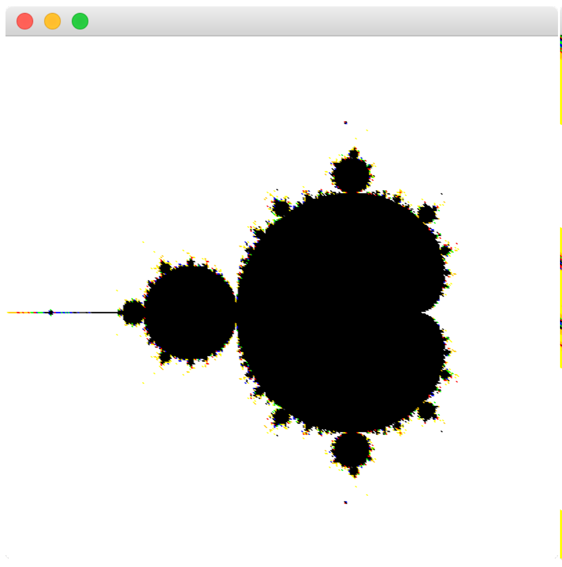
We strongly recommend that you just submit the zip file created by the specchecker, AFTER CHECKING THAT IT CONTAINS THE CORRECT CODE. ***If you mess something up and we have to run your code manually, you will receive at most half the points.***

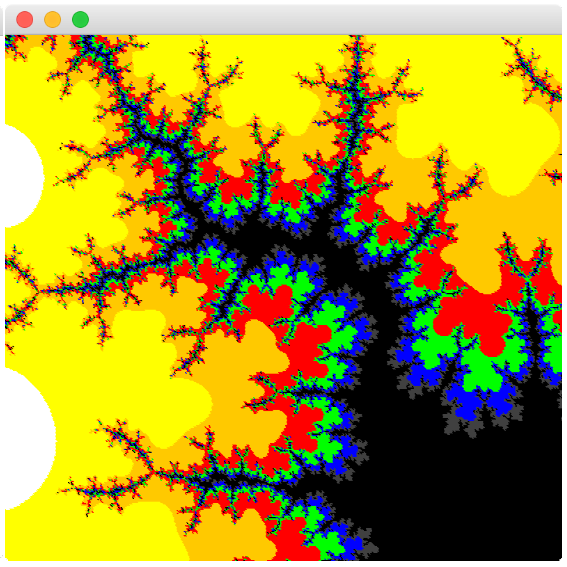
> We strongly recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **mini1**, which in turn should contain the file **`LoopaholicsAnonymous.java`**. You can accomplish this by zipping 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.

Optional reading: What's the point of that method `findEscapeCount`?

This section is not part of the assignment, but could be fun to investigate. It describes a graphical application that can be used to visualize the effect of your **`findEscapeCount`** *method.*

The algorithm you are implementing in **`findEscapeCount`** defines a representation of what is known as the *Mandelbrot set*, which can be loosely defined[1] as the set of points (x, y) for which the value $\sqrt{a^2 + b^2}$ is always less than or equal to 2, no matter how many times the given steps are repeated. If this value ever goes over 2, we say it has "escaped" the bound. To approximate this set of (x, y) pairs, we pick a maximum number of iterations (say 100) and see whether the value escapes within that many iterations. If not, we assume it is part of the set. The number of iterations it actually takes to escape for a given (x, y) is an interesting piece of information, and the number is often used to study the *boundary* of the Mandelbrot set, which exhibits a detailed self-similar or "fractal" structure (looks the same no matter how much you zoom in). People typically create a graphical representation using different colors to depict different ranges of escape values.
To experiment with what this means, take a look at the application MandelbrotExample.java. This is a simple GUI that depicts the Mandelbrot set based on your code for findEscapeCount. Points that are in the set are colored black, and points on the boundary are given a color depending on the escape value. You can select an area with the mouse to repeatedly zoom in. The initial screenshot on the left depicts the whole set, in a 3 x 3 region at the origin. The image on right is zoomed in to an area of roughly .005 x .005 on the top of the upper "bulb". You can edit the CUTOFFS array near the top of the file to change the way the colors are assigned (see the getColor method).





[1] More precisely, it's the set of *complex* numbers $c = x + yi$ such that the modulus of z_n never exceeds 2 in any number of iterations of the steps $z_{n+1} = z_n^2 + c$. In our case we're doing the same arithmetic using the notation $c = x + yi$ and $z = a + bi$ (where i is a symbol representing the square root of -1).

The GUI code is based on the Java Swing libraries. This style of programming is specialized (not to mention somewhat tedious) and we don't cover it in this course. However, you might want to investigate it on your own. Much of what you find about Swing on the www is out of date or incorrect. The official Oracle tutorial is detailed and accurate,
http://docs.oracle.com/javase/tutorial/uiswing/start/index.html
and there is a simpler collection of examples on Steve's web page,
http://web.cs.iastate.edu/~smkautz/
(scroll down to "Other stuff").