Com S 227
Fall 2014
Miniassignment 2
40 points

Due Date: Thursday, March 12, 11:59 pm (midnight)
"Late" deadline (25% penalty): Friday, March 13, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html , 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 Blackboard. Please do this right away.

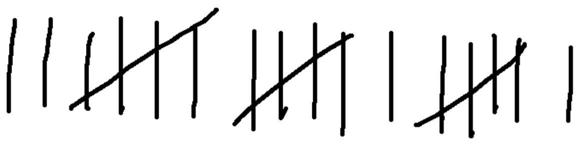
If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: This is a miniassignment and the grading is automated. If you do not submit it correctly, you will receive at most half credit. See the section "What to turn in" at the end of this document

Please start the assignment as soon as possible and get your questions answered right away.

Overview

For this miniassignment you will implement one class, called **`TallyString`**. This is a utility class containing a number of static methods for working with tally strings. A tally string is a representation of a nonnegative integer that is written using just three symbols. In this implementation a zero is represented by the digit '0' by itself, the character LINE (`'|'`) stands for the value 1, and the character STAR (`'*'`) has a value of 5. The symbols `'|'` and '*' can be combined to form a *tally group* whose value is the sum of the individual symbols; for example, `"||**|*|"` is a tally group with value 19.



A zero by itself is also considered to be a tally group, but a zero is not permitted as part of a group containing other characters (it can only appear by itself).

Larger numbers can be made by combining several tally groups, separated by one or more spaces, into a string[1]. The interpretation is that the group that is in the kth position from the right is multiplied by 10 to the power k, similar to the usual base-10 notation for numbers that use digits 0 through 9. (However, the values for tally groups are not necessarily restricted to the range 0 through 9). For example, the string

```
"*|** || |****"
```

which consists of three groups with values 16, 2, and 21, represents the number 1641 (calculated as $16 \times 100 + 2 \times 10 + 21$).

We also define a *normalized form* for a tally string, in which:

- The value of each group is a number in the range 0 through 9
- Each nonzero group has at most one STAR and at most four LINE characters, and all of the LINE characters are to the right of the STAR, if any
- There is exactly one space between groups and no extra spaces at the beginning or end of the string
- There are no extraneous zeros at the beginning of the string

For example, the normalized form of the tally string for the number 1641 is `"| *| |||| |"`, that is, it has four groups corresponding directly to the four digits 1, 6, 4, and 1. Similarly the normalized form of the tally string with value 7048 would be `"*|| 0 |||| *|||"`.

Specification

The specification for this asssignment includes

- this pdf,

[1] The idea of combining tally marks with a symbol for zero and place values appears to have originated with the Mayan civilization about two thousand years ago. However, they used base 20 rather than base 10. See http://www.storyofmathematics.com/mayan.html .

- any "official" clarifications posted on Piazza, and
- the online javadoc:
  http://www.cs.iastate.edu/~cs227/homework/mini2/doc/mini2/TallyString.html

The javadoc describes the required methods and their behavior. (The paragraphs at the top of the javadoc essentially contain the same information as that in the overview section above.) Note that this is a "utility" class whose methods are all static, so there are no constructors.

Where's the main() method??

There isn't one. This isn't a complete program and you can't "run" it by itself. It's just a single class definition that might be part of a larger system. To try out your class, you can write a test class with a main method. For example:

```
public class TallyTest
{
  public static void main(String[] args)
  {
    System.out.println(TallyString.isValidGroup("||**|*|"));
    System.out.println("Expected true");
    System.out.println(TallyString.evaluateGroup("||**|*|"));
    System.out.println("Expected 19");
    System.out.println(TallyString.makeGroup(19));
    System.out.println("Expected '***||||'");
  }
}
```

There is also a specchecker (see below) that will perform a lot of functional tests, but when you are developing and debugging your code at first you'll want to have some simple test cases of your own as in the `main` method above.

How do I make a string with a loop?

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)
    {
      result += s.charAt(i); // add on characters one at a time
    }
    return result;
  }
```

Alternatively you could iterate over the characters left to right, and append them on the *left*:

```
public static String reverse(String s)
{
  String result = "";
  for (int i = 0; i < s.length(); ++i)
  {
    result = s.charAt(i) + result;
  }
  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)
  {
    sb.append(s.charAt(i));
  }
  return sb.toString();
}
```

Getting started

Remember to work incrementally and test as you go. Here are some ideas.

1. As in previous assignments, create a new, empty project and add a package called **`mini2`**. Create the class **`TallyString`** and put in stubs for all the required methods as shown in the javadoc.

2. A good place to begin is with the methods **`isValidGroup`** and **`evaluateGroup`**. As always, start with some simple test cases as shown above in the section "Where's the main method?"

3. For **`makeGroup`**, be sure you have seen the section "How do I create a string with a loop?"

4. In order to continue, you'll now have to think about multiple groups. For example, the value of **`makeNormalizedString`** for the number 367 needs to have three groups, **`"||| *| *||"`**, one for the first digit (hundreds), one for the second digit (tens) and one for the third digit (ones). So a good place to start is with the simple question: how do I write a loop that will give me the values of the individual digits of an integer? As an example, first notice that:

367 % 10 = **7**	367 / 10 = 36
36 % 10 = **6**	36 / 10 = 3
3 % 10 = **3**	3 / 10 = 0 (*done*)

Once you have a loop to generate the place values (e.g. 7, 6, 3) then it's not hard to take advantage of your **`makeGroup`** method to assemble the string.

5. For the evaluateString, you'll need a loop to parse the string to get the individual tally groups, using a Scanner, and combine the values into a single integer. There are many ways to do this. Suppose someone is giving you the digits of a number one at a time, and you don't know how big the number will be. E.g., they say "3" and you don't know if that means 3 or 30 or 300 or 3000. Turns out it doesn't matter. If they give you another digit, just multiply whatever you have by 10 before adding it, e.g., in pseudocode,

```
total = first digit
while there is another digit
    total = total * 10 + next digit
```

Notes

1. You do NOT need arrays or ArrayLists for this assignment. You won't be penalized if you use them, but you would probably just be making things more complicated.

2. Your **`TallyString`** needs to be in a package named **`mini2`**. Do not add any additional Java classes.

The SpecChecker

A SpecChecker will be available by March 9.

Import and run the SpecChecker just as you practiced in Labs 1 and 2. It will run a number of functional tests and then bring up a dialog offering to create a zip file to submit. Remember that error messages will appear in the console output. There are many test cases so there may be an overwhelming number of error messages. *Always start reading the errors at the top and make incremental corrections in the code to fix them.*

When you are happy with your results, click "Yes" at the dialog to create the zip file.

See the document “SpecChecker HOWTO”, which can be found in the Piazza pinned messages under “Syllabus, office hours, useful links” if you are not sure what to do.

Documentation and style

Since this is a miniassignment, the grading is automated and in most cases we will not be reading your code. Therefore, *documentation is not required*. However, we recommend that you document each method as you create it. It is usually easier to write a method correctly after you have written down what it is supposed to do!

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder `miniassignment2`. 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 `miniassignment2`. *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 Blackboard, the zip file that is created by the SpecChecker. The file will be named `SUBMIT_THIS_mini2.zip`. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, **`mini2`**, which in turn contains one file, **`TallyString.java`**.

Submit the zip file to Blackboard using the Miniassignment 1 submission link and verify that your submission was successful by checking your submission history page. 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.”

Please LOOK at the file you upload and make sure it is right!. If you mess something up and we have to run your code manually, you will receive ***at most half the points****.*

We 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 **mini2**, which in turn should contain the file **TallyString.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.