CSCI 145 Week 8

Assignment 2

Assignment Description

In this assignment you will create a class LString. The LString will be a linked list class that imitates the standard Java String and StringBuilder classes. You must submit your program by 11:59 pm on Sunday, May 24th, 2015. This assignment will be worth 15% of your grade for the course.

Please read the entire assignment before beginning work.

Linked Strings

Java has two interesting String-like types:

- String which is an immutable (not changeable) list of chars.
- StringBuilder which is a mutable (changeable) list of chars.

StringBuilders are intended to be used to construct Strings. Both classes maintain the underlying list of chars using an array. The StringBuilder class can allocate a new array if the size of the StringBuilder object grows too large. (Obviously, since String is immutable, String does not require this logic.)

For this assignment, you will build a "Linked String" class called LString. Instead of using an array of characters, LString will use a linked list of characters (chars). An LString will be a mutable list of chars, similar to the StringBuilder class. An LString can change both its contents and its length during program execution.

Our LString class will contain a combination of String and StringBuilder methods. It will also include methods for converting between Strings and LStrings.

The LString Class

The public constructors and methods required for the LString class are listed here.

LString()	Construct an LString object. The LString object will represents an empty list of chars.
LString(String original)	Construct an LString object. The LString object will represent the same list of chars as original. That is, the newly created LString is a "copy" of the parameter original.
int length()	Return the length (number of chars) in this LString.
String toString()	Create and return an ordinary String with the same contents as this LString.

int compareTo(LString anotherLString)	Return the value 0 (zero) if this LString is equal to (has the same chars as) the parameter anotherLString; a value less than 0 if this LString is lexicographically less than anotherLString; and a value greater than 0 if this LString is lexicographically greater than anotherLString. Note: in lexicographic order "B" < "BB" < "Ba" < "a". A full definition of lexicographic ordering can be found with the description of the compareTo method for String: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#compareTo%28java.lang.String%29
boolean equals(Object other)	Return true if this LString represents the same list of characters as other. Note that the parameter is of type Object and not the type LString. See the note below for how to handle this.
char charAt(int index)	Return the char at the given index in this LString. Remember that the "first" char in the LString has index 0 (zero). If index is less than zero or greater than or equal to the length of this LString, this method should throw an IndexOutOfBoundsException.
void setCharAt(int index, char ch)	Set the char at the given index in this LString to ch. If index is less than zero or greater than or equal to the length of this LString, this method should throw an IndexOutOfBoundsException.
LString substring(int start, int end)	Returns a new LString that is a sub-string of this LString. The substring begins at the specified `start` and extends to (and includes) the character at index end-1. Thus, the length of the substring is end-`start`. If `start` is negative, or end is larger than the length of this LString, or `start` is larger than end, this method should throw an IndexOutOfBoundsException. Note: if `start == end == this.length()`, this method should return a null LString.
LString replace(int start, int end, Lstring lStr)	Replaces this characters in a sub-string of this LString with the characters in `lStr`. The indexes `start` and end must obey the same constraints as the `substring` method, above. This method should also throw an IndexOutOfBoundsException if `start` and end fail the constraints. If `start == end`, the effect of this method is to insert `lStr` at the given location in the LString. If `start == end == this.length()`, the effect is to append `lStr` at the end of this LString. Note that the resulting LString must not share any linked list structure with `lStr`.

Requirements

1. Your class must be named LString.
2. Your class must provide the methods listed above for construction, accessing, and manipulating LString objects.

3. Other than for testing purposes, your LString class should do no input or output.
4. Your package must enable the provided test program, LStringTest.java, to be compiled and run correctly. Use of this program is described in more detail, below.
5. **Important!** The purpose of this assignment is for you to gain experience with implementing and manipulating linked lists. For this reason, you are not allowed to use methods of the Java String or StringBuilder classes to implement LStrings. For example, you could implement the equals method by applying the toString method to both objects and then using equals method for Strings. Here is an explicit list of exceptions to this rule:
    - You may use the String charAt method to implement the LString(String) constructor.
    - You may use String and/or StringBuilder methods to implement the toString method.

    However, you can gain additional insight into the assignment by reading the standard Java documentation for the Java String and StringBuilder classes:
    - String: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html
    - StringBuilder: http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html

Testing

I have provided with this assignment a class LStringTest.java that tests your implementation of LString. The LStringTest class uses a package called JUnit 4 which provides support for testing Java classes.

I have provided the file junit.jar which contains the JUnit software. To compile and run your program in JGrasp you need to identify the location of the this file. You can do this by using the Settings/"PATH / CLASSPATH"/Workspace menu item. Then click on the CLASSPATHS tab. In that tab, click the New button. Now Browse to the junit.jar file. The file path and name should appear in the top of the two text boxes. Click OK (twice). You can test that this works by compiling LStringTest.java.

If you would like to compile using the command line, here is how you do it:

```
$ javac -cp junit.jar LStringTest.java
```

Here is how to run the program from the command line:

```
(Mac OS X and Linux)$ java -cp .:junit.jar LStringTest
(Windows)> java -cp .;junit.jar LStringTest
```

The difference between the two commands is the character separating the . and junit.jar in the the classpath (-cp). (On Mac OS X and Linux it's ':'. On Windows it's ';'.)

Here's a sample successful run of LStringTest.

```
Running constructor, length, toStringtests (10 tests)
Starting tests: ..........
```

```
Time: 0.017
OK! (10 tests passed.)

Running compareTo and equalstests (18 tests)
Starting tests: ..................
Time: 0.020
OK! (18 tests passed.)

Running charAt and setCharAttests (18 tests)
Starting tests: ..................
Time: 0.022
OK! (18 tests passed.)

Running substringtests (63 tests)
Starting tests: ...............................................................
Time: 0.063
OK! (63 tests passed.)

Running replacetests (31 tests)
Starting tests: ...............................
Time: 0.030
OK! (31 tests passed.)

Running specialtests (3 tests)
Starting tests: ...
Time: 0.046
OK! (3 tests passed.)

Congratulations! All tests passed.
```

I will be demonstrating this program in class on Wednesday.

Notes

1. Look for opportunities to (1) use one operation to perform another, or (2) write utility operations (don't make these `public`) that support multiple operations. For example, (1) the `equals` method can use the `compareTo` method, (2) `charAt` and `setCharAt` both need to locate the Nth character in the list.
2. Implementation of the `equals` method is tricky. The parameter is of type `Object` which allows comparison with anything. Here's how you should write the `equals` method:

```
@Override
public boolean equals(Object other) {
   if (other == null || !(other instanceof LString))
      return false;
   else {
```

```
            LString otherLString = (LString)other;
            // Your logic here to compare this and otherLString
            // Return true it they're equal, false otherwise
        }
    }
```

Some notes:

- The @Override tells the compiler that this method overrides the standard equals method.
- The if condition tests to ensure that other is not null and that it's an LString. If either of these conditions fails, the method returns false, indicating inequality.
- The expression (LString)other, converts the reference to an Object to a reference to an LString. This will work since we've already determined that it's actually an LString. (If, for some reason, it was not an LString, an exception would be raised here.)

3. The `substring` method should return a ***new*** LString. This means that the new and old LStrings must not share any linked list nodes. Similarly, the `replace` method must make a copy of the replacement LString lStr.
4. For the `substring` and `replace` methods, don't forget to set the length of the new LString.
5. Don't try to use a special character value as a sentinel (flag), for example, to flag the end of the LString. There are tests that attempt to detect this.
6. When doing this assignment, you should approach things in the following order:
    1. Create an LString class that will allow LStringTest to compile. To do that you will need to create stubs for all the LString methods described above.
    2. Now work through the methods in the following order:
        a) The constructor, `length`, and `toString`.
        b) `compareTo` and `equals`
        c) `charAt` and `setCharAt`
        d) `substring`
        e) `replace`.

    The tests in LStringTest are organized in six phases. The first five phases correspond to the five sets of methods, above. The sixth is a set of special tests that use the already built methods. The LStringTest program will not proceed to a phase unless all tests in the prior phases have passed. By working through the methods in the given order, you will work through the tests in order.

Coding Standards

Your program should follow the standards described as part of Lab 1.

Your program will be graded on conformance to the coding standards as well as correct functionality.

Turn in your program on Canvas

You should turn in your LString.java class on Canvas. You do not need to turn in LStringTest.java or junit.jar. Your program will be tested with the standard version of LStringTest. You should remember not to make any modifications to this class.