Cosc 1p03 Assignment 3

Abstract Data Types (ADT)

An Abstract Data Type is a wrapper which provides external functionality by defining an interface. The implementation behind the interface is often left to the code to implement as they see fit. One of the most common ADTs is the String library. It was developed across multiple languages to support common String manipulations. Before the String ADT came into existence most programmers who required any sort of String like functionality used character arrays or linked lists to simulate a String, writing their own manipulation routine.

For this assignment we will time warp back to the pre-String era and create our own. Below is an interface for a String library.

```
package String_T;

public interface StringT {

   //Concatenates 2 strings,  this.ConCat(S)
    public StringT ConCat(StringT S);

    //Returns a string from index 0 to i inclusive.
    public StringT Before(int i);

    //Returns a string from i to end.
    public StringT After(int i);

    //Returns the length of this.
    public int Length();

    //Creates a deep clone of this.
    public StringT Clone();

    //Returns the character at index i of this;
    public char CharAt(int i);

    //Returns a character array representation of this.
    public char[]   ToArray();

    //Returns a StringT composed of the characters between
    // i inclusive and j not inclusive
    public StringT SubString(int i, int j);

    //Complete the below 2 functions, which are to
    //implement some non-trivial behaviour
    //Look at the supplied link to String to get some ideas
    //It is left to you.

    //public StringT SomeKoolFunction1(???);
    //public someType SomeKoolFunction2(???);
}
```

The interface assumes that no String library exists but only character arrays. We will call the data type StringT to avoid conflict with the actual String data type. An implementation of the interface should be called MyStringT, which provides an implementation of the given interface, and is immutable. You will

notice that the last 2 functions are not defined. You can choose any 2 behaviours of a standard String operation to implement. The first must return a **StringT** and the 2nd some other type. Here is the String JavaDoc, https://docs.oracle.com/javase/8/docs/api/java/lang/String.html for some ideas.

Thus, to create a String the constructor must support a character array as a parameter to the implementation constructor. This constructor will convert the character array to an internal representation. The Default constructor will create an empty data structure, and thus represent an empty String.

The library can return a character array with the function **ToArray** as defined above.

All exceptions which constitute incorrect usage of the methods of **StringT**, should be trapped, and returned to the caller as **StringTException**. For example**, CharAt(-1)** would reference a non existent character in ***this***, and thus should throw an exception.

Create a test harness to completely test your implementation. Be sure that your test harness completely tests all aspects of your implementation and proves to the marker that you have done so. Place your test harness in a package called, **Test**.

One aspect of the test harness, it to print without using String. You will notice that printing to ***system.out*** supports printing a character array. If you are using **AsciiDisplayer** you may use the following construct ***out.writeString(new String(your character array));*** which is a small work around.

Speaking of printing a StringT, add an **iterator** to **StringT** so that the iterator can return each character in sequence. This is a much easier implementation then shown in class, since **StringT** is not a bounded type. Thus iterator<E> is a straightforward implementation. You will need to implement next and hasNext. The test harness must then test this in both forms. I.e. using **for-each** and **next – hasNext**.

What is the purpose behind this assignment?

The purpose is for you to define and create an abstract data type and implement a test harness to support your creation. This is all about process, and you should be aware that simply creating a new **StringT** is only a part of the assignment. Most of your effort should be directed to bullet proofing your code from bad usage. Thus, creating solid code.

Submission

Submit a zip file of your assignment via Brightspace. Include with your submission a comprehensive output of your test harness showing the robustness of your implementation. Be sure to provide proper commenting and layout of your code. The marker will be running your code.

End: