**BRAC University**

**Department of Computer Science and Engineering**

**CSE111 Lab 7**

***For all of the tasks below, assume that you are using a character array to store values. Methods of the String class CANNOT be used for any of the tasks, except for toCharArray ().***

***For all tasks, take String inputs and put them in a character array before you proceed to do what is asked in the tasks.***

Implement a **MyString** ADT.

**Elements**

An empty array of characters.

**Structure of the Elements**

A collection of characters. The characters in a string are in sequential (or linear) order – that is, the characters follow one after the other from the beginning of a string to its end. The character positions are numbered beginning with zero. A word, phrase, or sentence is some examples of strings.

The characters in the instance of MyString will be stored in an array.

**CONSTRUCTORS**

**MyString ( )**

Precondition:

None.

Postcondition:

This is the default constructor. It creates an empty MyString object (just a MyString reference).

Example:

```
... main ( ){
...
MyString a = new MyString( );
...
}
```

**MyString (char[ ] charSeq)**

Precondition:

An array of characters charSeq will be given to create the constructor.

Postcondition:

It creates a new MyString object with a character sequence identical to the character array charSeq.

Example:

```
... main ( ){
...
MyString a = new MyString(c); // c is a character array
...
}
```

**MyString (String str)**

Precondition:

A String str will be given to create the constructor.

Postcondition:

This creates a new MyString object whose contents are equivalent to the String str.

Example:

```
... main ( ){

...


MyString a = new MyString("cat");

...

}
```

**Task 1 (length())**

Take a string as input from the user and print the length of the string (Remember that you cannot use any string class methods other than *toCharArray ()*

**Task 2 (charAt ())**

Take a string as input from the user … then take a position *n* as input. "n" must be a valid index (which is an integer) less than the length of the String (Refer to Task1 on how to find the length). If n is invalid, then print "Invalid index", otherwise print the character at that index/position.

**Task 3 (startsWith ())**

Take two Strings as input from the user … check if the second String is a prefix of the first String. If yes, print "true" else print "false". Please ensure your program works for empty string inputs, as in it shouldn't crash if the first or second or both of the strings are empty: P

*Remember that you cannot use any String class methods here, hence character by character comparison will be required.*

**Task 4 (endsWith () )**

Take two Strings as input from the user … check if the second String is a suffix of the first String. If yes, print "true" else print "false". Please ensure your program works for empty string inputs, as in it shouldn't crash if the first or second or both of strings are empty: P

**Task 5 (replaceFirst(char,char))**

Take a String as input from the user. Then take in two more characters, oldChar and newChar (Read in as string and then convert to charArray and the save first index to a char variable). Replace the first occurrence of oldChar with newChar .. You are not permitted to modify the original character array ...: P

**Task 6 (replaceAll(char,char))**

Take a String as input from the user. Then take in two more characters, oldChar and newChar (use the same method as Task 5). Replace all occurrences of oldChar with newChar ..you are not permitted to modify the original character array.

**Task 7 (replaceLast(char,char))**

Take a String as input from the user. Then take in two more characters, oldChar and newChar (use the same method as Task 5 and 6). Replace the last occurrence of oldChar with the newChar… you are not permitted to modify the original character array.

**Task 8 (toLowerCase())**

Take a String as input from the user. Convert all uppercase to lowercase.

**Task 9 (toUpperCase())**

Take a String as input from the user. Convert all lowercase to uppercase.

**Task 10(equals())**

Take two Strings as input; if they are equal print “true”, else print “false”.

**Task 11(equalsIgnoreCase()**

Take two Strings as input; if they are equal regardless of the case, print “true”, else print “false”.

**Task 12(compareTo())**

Take two String as input. Print 0 if the Strings are equal. If the second string is lexicographically before the first one, returns a negative value, else returns a positive value. This task should behave the same as the compareTo() method in the String class, so look at the method definition in java doc for String.

**Task 13(compareToIgnoreCase())**

Take two String as input. Print 0 if the Strings are equal. If the second string is lexicographically before the first one, returns a negative value, else returns a positive value, regardless of the case. This task should behave the same as the compareToIgnoreCase() method in the String class, so look at the method definition in java doc for String.

**Task 14(substring (int))**

Take a String as input, and then an integer. If the integer is within the valid range, then find the substring starting with that integer till the end

**Task 15(substring (int,int))**

Take a String as input, and then two integers. If the integers are within the valid range, then find the substring starting with the character at the position of the first value and ending at the position of the last character.

**Task 16(indexOf(char))**

Take a String as input and then take a character as input (Same method as in Task 5/6). If the character exists in the String, print the location of the first occurrence of the character. If the character is not found, print -1.

**Task 17(lastIndexOf(char))**

Take a String as input and then take a character as input (Same method as in Task 5/6). If the character exists in the String, print the location of the last occurrence of the character. If the character is not found, print -1.

**Task 18(indexOf(char,int))**

Take a String as input and then a character and a number as input. Check the string from location indicated by the integer value, and print the location of the first occurrence of the character. If not found, print -1.

**Task 19(lastIndexOf(char,start))**

Take a String as input and then a character and a number as input. Check the string from location indicated by the integer value, and print the location of the last occurrence of the character. If not found, print -1.

**Task 20(concat())**

Take two strings as input and create a new character array that contains all characters of the first String followed by all characters of the second string. Print the new array and verify.

**METHODS**

**[Some of the more commonly used String class methods-but you CANNOT use the String class methods here. You have to implement these methods on your own.]**

**int length ( )**

Precondition:

None.

Postcondition:

Returns the number of characters in the MyString object.

**charcharAt ( int n )**

Precondition:

"*n*" must be a valid String index (which is an integer) less than the length of the MyString object where you invoke this method (check the validity for *n*, e. g., *n* is an integer, non-negative and less than the length of the String).

Postcondition:

Returns the $n^{th}$ character in the MySrting object.

**boolean startsWith (MyString prefix)**

Precondition:

A MyString object *prefix* that is not null.

Postcondition:

Returns true if the MyString object starts with "*prefix*". Otherwise, returns false.

**boolean startsWith (String prefix)**

Precondition:

A String object *prefix* that is not null.

Postcondition:

Returns true if the MyString object starts with "*prefix*". Otherwise, returns false.

**boolean endsWith(MyString suffix)**

Precondition:

A MyString object *suffix* that is not null.

Postcondition:

Returns true if the MyString object ends with "*suffix*". Otherwise, returns false.

**boolean endsWith(String suffix)**

Precondition:

A String object *suffix* that is not null.

Postcondition:

Returns true if the MyString object ends with "*suffix*". Otherwise, returns false.

**MyString replaceFirst(char oldChar, char newChar)**

Precondition:

Two valid characters "*oldChar*" and "*newChar*".

Postcondition:

Returns a new MyString resulting from replacing the first occurrence of the *oldChar*in this string with the *newChar*.

**MyString replaceAll(char oldChar, char newChar)**

Precondition:

Two valid characters "*oldChar*" and "*newChar*".

Postcondition:

Returns a new MyString resulting from replacing all occurrences of the *oldChar* in this string with the *newChar*.

**MyString replaceLast(char oldChar, char newChar)**

Precondition:

Two valid characters "*oldChar*" and "*newChar*".

Postcondition:

Returns a new MyString resulting from replacing the last occurrence of the *oldChar*in this string with the *newChar*.

**MyString toLowerCase ( )**

Precondition:

None.

Postcondition:

Returns the invoking MyString object if all its characters are already lowercase. Otherwise, returns a new MyString object in which all characters have been converted to lowercase.

**MyString toUpperCase ( )**

Precondition:

None.

Postcondition:

Returns the invoking MyString object if all its characters are already uppercase. Otherwise, returns a new String object in which all characters have been converted to uppercase.

**boolean equals ( MyStringrightStr )**

Precondition:

A MyString object *rightStr* and *rightStr* is not null. (check validity of *rightStr*)

Postcondition:

It returns true if the invoking MyString object and rightStr have the same value (i.e, identical). Otherwise, returns false.

**boolean equalsIgnoreCase ( MyStringrightString )**

Precondition:

A MyString object *rightStr* and *rightStr* is not null. (check validity of *rightStr*)

Postcondition:

It returns true if the invoking MyString object and *rightString* are identical not considering the case (uppercase or lowercase) to each character. Otherwise, returns false.

**int compareTo ( MyStringstr )**

Precondition:

A MyString object *str*. *Str* is not null. (check validity of *str*)

Postcondition:

Returns a value indicating if the invoking MyString object is lexicographically before (returns a negative value), equal to (returns 0), or after (returns a positive value) the MyStringstr.

Example:

... main ( ){

...

a.MyString(b); // a and b are instances of MyString Class

...

}

[This method returns 0 if a and b are identical, returns negative value if a<b and returns positive value if a > b.]

**Int compareTo (String str )**

Precondition:

A String object *str* and *Str* is not null. (check validity of *str*)

Postcondition:

Returns a value indicating if the invoking MyString object is lexicographically before (returns a negative value), equal to (returns 0), or after (returns a positive value) the String str.

Example:

... main ( ){

...

a.MyString("book"); // a is an instances of MyString Class

...

}

**int compareToIgnoreCase(MyStringstr)**

Precondition:

A MyString object *str* where *str* not null (check validity of *str*).

Postcondition:

Returns a value indicating if the invoking MyString object is lexicographically before (returns a negative value), equal to (returns 0), or after (returns a positive value) the MyString *str*, if both are not case sensitive.

**int compareToIgnoreCase(String str)**

Precondition:

A String object *str* where *str* not null (check validity of *str*).

Postcondition:

Returns a value indicating if the invoking MyString object is lexicographically before (returns a negative value), equal to (returns 0), or after (returns a positive value) the String *str*, if both are not case sensitive.

**MyString  substring (int start)**

Precondition:

The argument "*start*" must be a nonnegative String index and is not greater than the length of the MyString object.

Postcondition:

Returns a new MyString object containing the substring from the index "*start*" to the end of the inviokingMyString object.

**MyString substring (int start, int end)**

Precondition:

The "*start*" and "*end*" must be nonnegative String indices and are not greater than the length of the MyString. Moreover, "*start*" must not be greater than "*end*". (check validity)

Postcondition:

Returns a new MyString object containing the substring starting at position "*start*" through position "*end*" of the invoking MyString object. [Here, a total of "*end* –*start* + 1" characters are copied into the new MyString object].

**int indexOf (char ch)**

Precondition:

A character "*ch*", where "*ch*" is not null. (check validity of "ch")

Postcondition:

Returns the position within the invoking MyString object at which the first (the leftmost) occurrence of the character "*ch*" is located. If "*ch*" is not found, -1 is returned.

**int lastIndexOf (char ch)**

Precondition:

A character "*ch*", where "*ch*" is not null. (check validity of "*ch*")

Postcondition:

Returns the index within this Mystring object of the last (rightmost) occurrence of the specified character "ch". If "ch" is not found, -1 is returned.

**int indexOf ( char ch, int start )**

Precondition:

A character "*ch*", where "*ch*" is not null. (check validity of "*ch*") and the starting position "*start*" to start searching within the MyString object.

Postcondition:

Returns the position within the invoking MyString object at which the first (the leftmost) occurrence of the character "*ch*" is located, with "start" specifying the position at which to begin the search. If "*ch*" is not found, then -1 is returned.

**int lastIndexOf (char ch, int start)**

Precondition:

A character "*ch*", where "*ch*" is not null. (check validity of "*ch*") and the starting position "*start*" to start searching within the MyString object.

Postcondition:

Returns the index within this string of the last occurrence of the specified character, searching from the position "*start*". If "*ch*" is not found, -1 is returned.

**int indexOf ( MyStringstr)**

Precondition:

A MyStrjng object *str* that is not null. (check validity of *str*)

Postcondition:

Returns the position within the invoking MyString object at which the first (the leftmost) occurrence of the MyStringstr is located. If "*str*" is not found, then -1 is returned.

**int lastIndexOf (MyStringstr)**

Precondition:

A MyStrjng object *str* that is not null. (check validity of *str*)

Postcondition:

Returns the position within the invoking MyString object at which the last (the righttmost) occurrence of the MyStringstr is located. If "*str*" is not found, -1 is returned.

**int indexOf (String str)**

Precondition:

A Strjng object *str* that is not null. (check validity of *str*)

Postcondition:

Returns the position within the invoking MyString object at which the first (the leftmost) occurrence of the String *str* is located. If "*str*" is not found, then -1 is returned.

**int lastIndexOf (String str)**

Precondition:

A Strjng object *str* that is not null. (check validity of *str*)

Postcondition:

Returns the position within the invoking MyString object at which the last (the rightmost) occurrence of the String *str* is located. If "*str*" is not found, -1 is returned.

**MyString concat (MyStringstr)**

Precondition:

A MyString object *str*. The object *str* is not null (check validity of *str*).

Postcondition:

This Returns a new MyString object that containins the MyString object that invoked this method with *str*, added to it at the end.

Example:

string1.concat(string2);

// string 1 and string2 are instances of MyString class.

**MyString concat (char[ ] charSeq)**

Precondition:

A character array.

Postcondition:

This Returns a new MyString object that containins the MyString object that invoked this method with *charSeq*, added to it at the end.

**MyString concat (String str)**

Precondition:

A String *str*. The object *str* is not null. (check validity of *str*)

Postcondition:

This Returns a new MyString object that containins the MyString object that invoked this method with a string *str*, added to it at the end.