# *Programming II (420-B20-HR)*

# *Lab 5 – Intro to String Processing*

Date assigned: Wednesday, February 17, 2016

Date due: **Wednesday, February 17, 2016**

**Learning Objectives**

1. Learn to create, add to and insert text into **StringBuffer** objects.
2. Learn to use miscellaneous methods of the **StringBuffer** class.
3. Learn the difference between object identity and object equality;
4. Learn to compare strings lexicographically;
5. Learn to create and use **StringTokenizer** objects;
6. Learn to extract data fields from a **String** using a **StringTokenizer** object.
7. Review static class variables and methods.

**Methods Used:**

*String methods:*

int compareTo(String *string)*

int compareToIgnoreCase(String *string)*

*StringBuffer constructor:*

**StringBuffer ( String***string* **)**

*StringBuffer methods:*

**StringBuffer append ( Type** *t* **)**

**StringBuffer insert ( int** *offset***, Type** *t* **)**

StringBuffer replace(int *start*, int *end*, String *str*)

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

*StringTokenizer constructor:*

**StringTokenizer ( String***str* [*,***String***delimiter*]**)**

*StringTokenizer methods:*

**int countTokens( )**

**boolean hasMoreTokens( )**

**String nextToken(** [,**String** *delimiter*]**)**

**To Be Handed In:**

1. The ***username*\_B20\_L05\_Project** folder should be uploaded to **Moodle**.

**To Start:**

1. Download and unzip the folder **B20\_L05\_Project** from **Moodle**. Rename it to ***username*\_B20\_L05\_Project**.
2. Start **Eclipse** and select your **420-B20\Labs** folder as your workspace.
3. Create a new **Java Project** called ***username*\_B20\_L05\_Project**.

# The StringBuffer Class

Purpose***:*** Learn to use the **StringBuffer** class to dynamically create strings.

To Do:

## Open **Repeater** in the **b20\_l05\_strings** package. The **repeat()** method repeats an input string *n* times. The "pseudo-location" of the string that is created is displayed.

## Open **TestRepeater**. This program inputs a string and calls the **repeat()** method to display the string 5 times. Run the program. Enter your name at the prompt.

How many different locations are created for **newString**? \_\_\_5\_\_\_\_\_

***Explanation***:

Every time a String changes value a new physical object is created. Since the method adds the repeated string to the end of the string 5 times, 5new strings are created after the original instantiation (for a total of 6 strings created!)

This leaves 5 strings that must be deleted from memory. This is done automatically by Java and is called *garbage collection.*

## In cases where a string is going to be frequently modified, it is more efficient to use the **StringBuffer** class than the **String** class.

***The StringBuffer Class:***

A string buffer is like a [String](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html), but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through method calls.

**StringBuffer Methods:**

**append() :** adds to the end of the string

**insert():**  inserts data into a string

## Create a second copy of **repeat()** in **Repeater** and make the following changes:

### Change the method name to **repeatStringBuffer()**

### Change the **String newString** to a **StringBuffer**.

### Use the **append()** method instead of the concatenate operator to add to the end of it.

### Convert **newString** to a **String** to return it.

## In **TestRepeater** add the following lines before the closing } of main().

System.out.println("\nUsing a StringBuffer:");

t = overAndOver.repeatStringBuffer(s,5);

System.out.println(t);

## Test your changes.

How many different locations are created for **newString** in **repeatStringBuffer()**? \_\_1\_\_\_\_\_\_

# Process each character of a string

Purpose: Learn to process each character of a string.

To Do:

The **reverse()** method in the **StringMethods** class in the **b20\_l05\_strings** package is the same as the version on in the notes.

## Run **StringProcessingExamples**. Select "Miscellaneous String Methods" from the "Exercises" menu. Type a sentence. Click the buttons. At the moment, each button just displays the sentence you entered.

In *MiscStringFrame*:

## Add two statements to the **btnReverse\_actionPerformed()** method. The first statement should display "The sentence in reverse is: " after the display of the input sentence. The second statement should display the returned value from a call to the **StringMethods reverse()** method that passes in the sentence you read. (Note: the methods in StringMethods are all static so you do not have to instantiate an object to call them.)

For example, if the sentence was "The cat in the hat came back.", your output should look like:

The cat in the hat came back.

The sentence in reverse is:

.kcab emac tah eht ni tac ehT

## Test your changes.

In *StringMethods*:

## Write the code for the **asciiString()** method. It should convert each character to its ASCII integer equivalent. For example, if the String was "Mississippi", the output would be:

77 105 115 115 105 115 115 105 112 112 105

Refer to the **reverse()** method to see how to process every character in a **String**. Use a StringBuffer for the result string.

*Hint:* To convert a char to its ASCII equivalent use the **int** cast operator.

In *MiscStringFrame*:

## Add two statements to the **btnToAscii\_actionPerformed()** method. The first statement should display "The sentence in ASCII is: " after the display of the input sentence. The second statement should display the returned value from a call to the **StringMethods asciiString()** method that passes in the sentence you read.

## Test your changes.

# Miscellaneous StringBuffer Methods

Purpose:Learn to use the **replace()** and **substring()** methods of the **StringBuffer** class.

To Do:

## Run **StringProcessingExamples**. Choose "Miscellaneous String Methods" from the "Exercises" menu. Type a sentence. Click the **To Sentence** button. Right now, the program just inputs a sentence and echoes it back to the user.

We want to write a method to convert an input string into sentence format. A sentence is considered in sentence format if

* it has no leading or trailing spaces
* it begins with an uppercase letter
* it ends with a period, exclamation mark or question mark.

## Add the following static method to **StringMethods**:

**public static String toSentenceCase(String s)**

**{**

**// Trim the leading and trailing spaces from the input string,**

**// convert it to lowercase and assign it to the result**

**// StringBuffer**

**StringBuffer result = new StringBuffer(s.trim().toLowerCase());**

**return result.toString();**

**} // toSentenceCase(String)**

## At the moment, the method just trims leading and trailing spaces from the string and converts it to lower case. Now we are going to use the **StringBuffer** **replace()** and **substring()** methods to convert the first character to uppercase.

The StringBuffer substring() method

The **substring()** method returns a new String that contains a subsequence of characters currently contained in this sequence. The substring begins at the specified start and extends to the character at index end - 1.

**Format:**

String substring(int start, int end)

**Parameters:**

start - The beginning index, inclusive.

end - The ending index, exclusive.

**Returns:**

The new string.

**Example:**

Given:

**StringBuffer state = new StringBuffer("Thinking about Java");**

Then:

**state.substring(0,7)** - evaluates to **"Thinkin"**

The StringBuffer replace() method

The **replace()** method replaces the characters in a substring of this sequence with characters in the specified String. The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists. First the characters in the substring are removed and then the specified String is inserted at start. (This sequence will be lengthened to accommodate the specified String if necessary.)

**Format:**

public StringBuffer replace(int start, int end, String str)

**Parameters:**

start - The beginning index, inclusive.

end - The ending index, exclusive.

str – the String that will replace previous contents.

**Returns:**

This object.

**Throws:**

**StringIndexOutOfBoundsException** - if start is negative, greater than length(), or greater than end.

**Example:**

Given:

**StringBuffer str = new StringBuffer("Thinking about Java");**

Then:

**str.replace(0,5,"Dream")** - evaluates to **"Dreaming about Java"**

#### To convert the first letter of the first word to uppercase, use the **replace()** method to replace the first character with the uppercase version of the character. The **start** is 0 (the first character), the **end** is 1. To get the uppercase version of the first character use: **result.substring(0,1).toUpperCase()**

#### If the sentence does not end with '.', '!' or '?', add a '.' to the end of the sentence. (**Hint**: create a String variable that is the last character in the **StringBuffer**. If it is not '.', '!' or '?', append a '.' to the end of **result**.

## In the **btnToSentence\_actionPerformed()** of the **MiscStringFrame** class, add a statement to call your **toSentenceCase()** method to convert the input sentence to sentence case and display the result. Run **StringProcessingExamples** to test your changes using the following test cases.

| **Test Case** | **Input Sentence** | **Expected Output** |
| --- | --- | --- |
| Sentence already in sentence case | “The cat in the hat.” | “The cat in the hat.” |
| Sentence all lowercase | “the cat in the hat.” | “The cat in the hat.” |
| Sentence all uppercase | “THE CAT IN THE HAT.” | “The cat in the hat.” |
| Sentence is a question | “The cat in the hat?” | “The cat in the hat?” |
| Sentence is an exclamation | “The cat in the hat!” | “The cat in the hat!” |
| Sentence has leading and trailing spaces | “ The cat in the hat. ” | “The cat in the hat.” |
| Sentence does not end with “.”, “!” or “?” | “The cat in the hat” | “The cat in the hat.” |
| Sentence has leading and trailing spaces, does not begin with uppercase letter, does not end with “.”, “!” or “?” | “ the Cat in the Hat ” | “The cat in the hat.” |

# Object Identity and Object Equality

Purpose: Learn the difference between object identity and object equality.

To Do:

## Open and run the **StringEquality** class.

## Complete the following table with the results you got. Write **I** in a box if the two Strings are identical and **E** in the box if the strings are equal. For example, **s1** is both equal to and identical to **s1**, therefore, both **I** and **E** are written in the box.

|  | **s1** | **s2** | **s3** | **s4** | **s5** | **s6** | **s7** | **"Lab 5"** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **s1** | I E | *E* |  | *E* |  |  |  | *E* |
| **s2** | *E* | I E |  | *E* |  |  |  |  |
| **s3** |  |  | I E |  | *I E* |  |  |  |
| **s4** | *E* | *E* |  | I E |  |  |  |  |
| **s5** |  |  | *I E* |  | I E |  |  |  |
| **s6** |  |  |  |  |  | I E | *I E* |  |
| **s7** |  |  |  |  |  | *I E* | I E |  |
| **"Lab 5"** | *E* |  |  |  |  |  |  | I E |

## Use the **equalsIgnoreCase()** method instead of the **equals()** method in the **testEqual()** method of the **StringEquality** class. Test your change. How does the output differ from the previous run?

\_\_All of the strings are now equal, but not all are identical\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# The compareTo() method

Purpose: Learn to use the **compareTo()** method to compare two strings.

To Do:

The compareTo() method:

The **compareTo()** method and **compareToIgnoreCase()** compares two strings.The **equals()**  method determines whether or not two strings are equal, but doesn't determine which is greater or less lexicographically.

**Format:**

int toCompare( String *anotherString*)

int toCompareIgnoreCase( String *anotherString*)

## Run **StringProcessingExamples**. Choose "*CompareTo Method*" from the "*Exercises*" menu. Record the results for the following input strings in the appropriate column:

|  |  |  |  |
| --- | --- | --- | --- |
| **str1** | **str2** | **str1.compareTo(str2)** | **str2.compareTo(str1)** |
| "abc" | "def" | -3 | 3 |
| "abc" | "abcdef" | -3 | 3 |
| "abc" | "ABC" | 32 | -32 |
| "abc" | "abc" | 0 | 0 |
| "dog" | "cat" | 1 | -1 |

## Modify **btnCompare\_actionPerformed()** method in the **StringCompareFrame** class so that it uses **compareToIgnoreCase()** instead of **compareTo()** and:

### if the first string is lexicographically less than the second string, it displays:

*str1* is less than *str2*

### if the first string is lexicographically equal to the second string, it displays:

*str1* is the same as *str2*

### if the first string is lexicographically greater than the second string, it displays:

*str1* is greater than *str2*

## Test your changes on the same data you used earlier.

# The StringTokenizer class

***Purpose:*** Learn to use the **StringTokenizer** class.

***To Do:***

We want to know how many words are in a sentence and to output the words of the sentence on separate lines. We could do this using the **indexOf()** and **substring()** methods of the **String** class, but the **StringTokenizer** class provides us with an easier way to do this.

**The StringTokenizer Class:**

The **StringTokenizer** class breaks a string into strings separated by delimiters.

**Constructor Format**:

**StringTokenizer ( String***str* [*,***String***delimiter*]**)**

where:

*str* is the String you want to tokenize

*delimiter* is the separator between tokens.

If *delimiter* is omitted, the delimiter is a blank space.

***Examples:***

StringTokenizer st1 = new StringTokenizer("Just two words");

- creates a **StringTokenizer** object with a value of "Just two words" and a delimiter of space. There are three tokens in the String: *Just*, *two* and *words*.

StringTokenizer st2 = new StringTokenizer("username:password", ":");

**-** creates a **StringTokenizer** object with a value of "username:password" and a delimiter of ":". There are two tokens in the String: *username* and *password*.

**StringTokenizer** methods:

**int countTokens( )** - returns the number of tokens left in the sentence

**boolean hasMoreTokens( )** - returns true if there are more tokens and false if there are no more tokens

**String nextToken(** [,**String** *delimiter*]**)** - returns the next token delimited by *delimiter.* (The default delimiter is the space character.)

The countTokens() method:

## Open **StringTokenizerFrame** and import **java.util.StringTokenizer**. Add the following lines after assigning the sentence in the **fldSentence\_actionPerformed()** method:

StringTokenizer myTokenizer = new StringTokenizer(sentence);

resultsDisplay.setText("There are "

+ myTokenizer.countTokens()

+ " words in the sentence.\n");

## Run **StringProcessingExamples**. Choose "StringTokenizer Methods" from the "Exercises" menu. Type a sentence and hit enter to see how the **countTokens()** methods works.

The hasMoreTokens() and nextToken() methods:

## Add the following lines to the end of the **fldSentence\_actionPerformed()** methodin **StringTokenizerFrame**

resultsDisplay.append("The words are: \n");

while(myTokenizer.hasMoreTokens())

resultsDisplay.append (myTokenizer.nextToken()+"\n");

## Run the program. What is output? \_\_There are 5 words in the sentence\_\_

## Create a second **StringTokenizer** called **eToken** that is initialized to **sentence** and has the letter "e" as a delimiter. Add code to the class to display the number of tokens in this tokenizer and to display all the tokens in the same way that you did for the **myTokenizer** object.

## Test your changes on the following sentence: "*A rat in the house might eat the ice cream.*"

***Questions:***

How many words are there? \_\_11 words\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How many tokens delimited by "e" are there? \_\_\_7\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What are the tokens delimited by "e"?

\_A rat in th\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

hous\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

might\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

at th\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ic\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

cr\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

am\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Is the delimiter included in the token? \_\_\_no\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Challenge: Pig Latin Converter

Purpose: Practise using Strings to convert a sentence into pig Latin.

A **PigLatinConverter** class is included in the project. It includes one static public method (**toPigLatin(String)**) and two private static methods (**wordToPigLatin(String)** and **isVowel(char)**).

**Algorithm to convert a word to pig latin:**

Look at each word in the sentence. If its first letter is a vowel, simply add "yay" to it. Otherwise, remove all the letters from the beginning of the word up to the first vowel. Put them after the word. Then add "ay".

**For example,**

if the word is "on" – it begins with a vowel and therefore becomes "onyay"

if the word is "Java" – the first vowel is 'a' and therefore, the 'J' is put at the end and the pig latin word is "avaJay"

if the word is "sheep" – the first vowel is 'e' and therefore, the "sh" is put at the end and the pig latin word is "eepshay"

To Do:

## For bonus marks, write the code for the **toPigLatin()** and **wordToPigLatin()** methods. Use a **StringBuffer** for your results and a **StringTokenizer** to retrieve the separate words. Test your class with the **PigLatinConverterFrame**.

# Homework

## Complete the **Week 5 Quiz** on Moodle by Feb. 21.