# CIS 3145 Class Notes: Text Chapter 09

## Strings Concepts

**Objectives**

* Create a String object
* Use standard String class methods
* Use the StringBuilder class to create a mutable sting object

**String Processing**

There are two ways of working with string variables, using a **String** class or **StringBuilder** class.

With the **String** Class a value is assigned to a variable with the String constructor, a literal string, or variable. There are many methods for ***manipulating*** a **String** variable. For example, the **length** method determines the size of the string and the **split** method returns an array (which will be covered in the chapter on arrays) of smaller strings based on a delimiter. Because the String variable holds an object, these methods are called from the object with the dot notation: **object.method**().

More important for this chapter are the **index** methods. Both the **indexOf**() and **lastIndexOf**() methods take a target **String value** as an **argument** and return an integer value that represents the first or last location respectively of that target string inside the string variable. The location is the ordinal position of the target string where the first character of the string has a position value of 0, not 1. In this example, the target string is “j”. The myString object calls the index methods on the string data inside of itself. Both index methods can take a second startIndex **argument** which tells the method where to start the search from.

String myString = "java jam";

int startLocation = myString.indexOf("j"); // the integer value is 0

int endLocation = myString.lastIindexOf("j"); // the integer value is 5

The **charAt** method is the opposite of the index method. It takes an **index** as the **argument** and returns the character at that location. The statements below both return the character “j”.

char myCharacter1 = myString.charAt(0);

char myCharacter2 = myString.charAt(5);

The **substring** method is like the **charAt** method but it returns a string instead of a single character.

There is another set of methods used to ***compare*** one string object to another object. The **isEmpty** method is important for data validation (checking that a user entered a value in a textbox). The “**equals”** methods are used in place of the **Equality** relational operator used in the ***Boolean Expression*** of If statements (use the equality operator with primitive data types). These **equal** **methods** are needed since using == on objects only returns true when the two variables hold the same underlying object (the variables **point** to a single memory location where an object is stored). The **equal methods** return true when the underlying **data** are the same.

String myString1 = "Sue";

String myString2 = "Sue"; //String 2 is a separate object than String 1

String myString3 = myString1; //String 3 and String 1 point to the same underlying object

If (myString1 == myString2) -> will be false

if (myString1.equals(myString2)) -> will be true

If (myString1 == myString3) -> will be true

**StringBuilder Class**

One problem with **String** objects is that they can’t be changed; they are immutable. When we assign a new value to a **String** variable the old object is destroyed and a new one is created. The **StringBuilder** class is mutable and allows us to efficiently change and update a variable that holds string information. This class does what the **String** class can do and more. For example, it can **insert** values into a string such as for turning 10 digit characters into a formatted phone number.

There are three **constructors** for creating a StringBuilder object. The default constructor with no arguments creates an object that is 16 characters long, a constructor with an integer argument will create an object that has a capacity equal to the integer value, and the third constructor takes a String argument and creates an object the **size** of the string **plus 16**.

The StringBuilder object has some methods identical to a String object, such as the **charAt** and **substring** methods. In addition, the StringBuilder has methods for changing a single character in the variable: **deleteCharAt** takes an integer argument and deletes the character at the integer index location, and **setCharAt** takes two arguments, an integer and character, and replaces the character at the integer index location with the new character argument value.

The StringBuilder object also has the **insert**, **replace**, and **delete** methods to change more than a single character.