Northeastern University

Lecture 10: Object Oriented Programming - 5

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Liang, Y. Daniel. Introduction to Java Programming and Data Structures, Comprehensive Version, 12th edition, Pearson, 2019.

Outline

- Objects and Classes
- Thinking in Objects

Outline

- Objects and Classes
- Thinking in Objects

The `String` Class

Strings Are Immutable

- A String object is immutable; its contents cannot be changed.
- Does the following code change the contents of the string?

```
String s = "Java";

s = "HTML";
```

- The answer is no. Why?

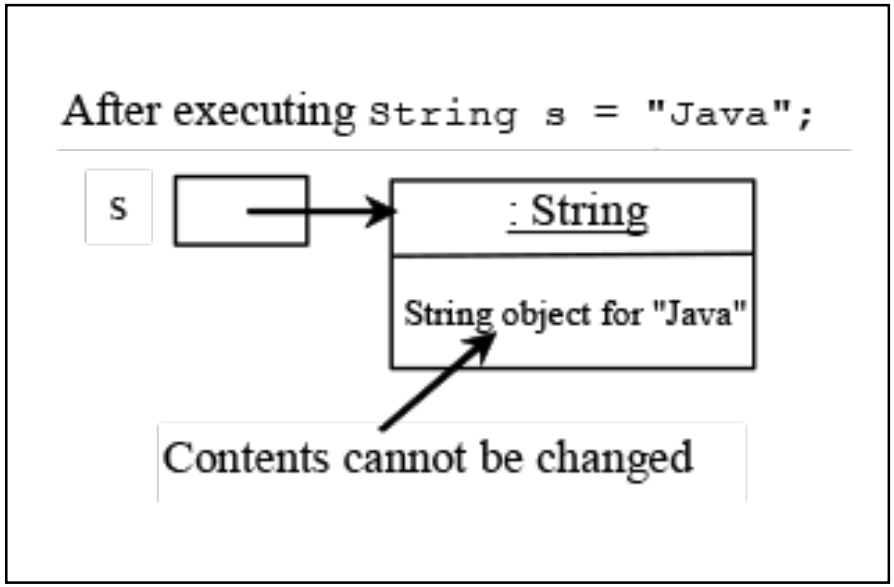
Trace Code

```
String s = "Java";

s = "HTML";
```

Trace Code

```
String s = "Java";
```

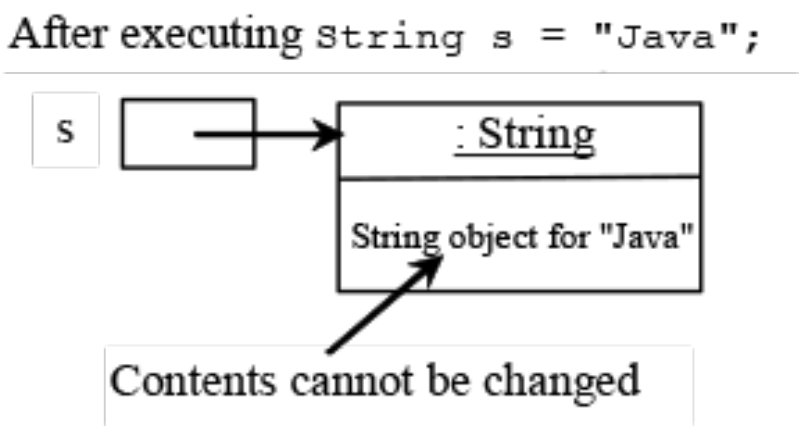
```
s = "HTML";
```

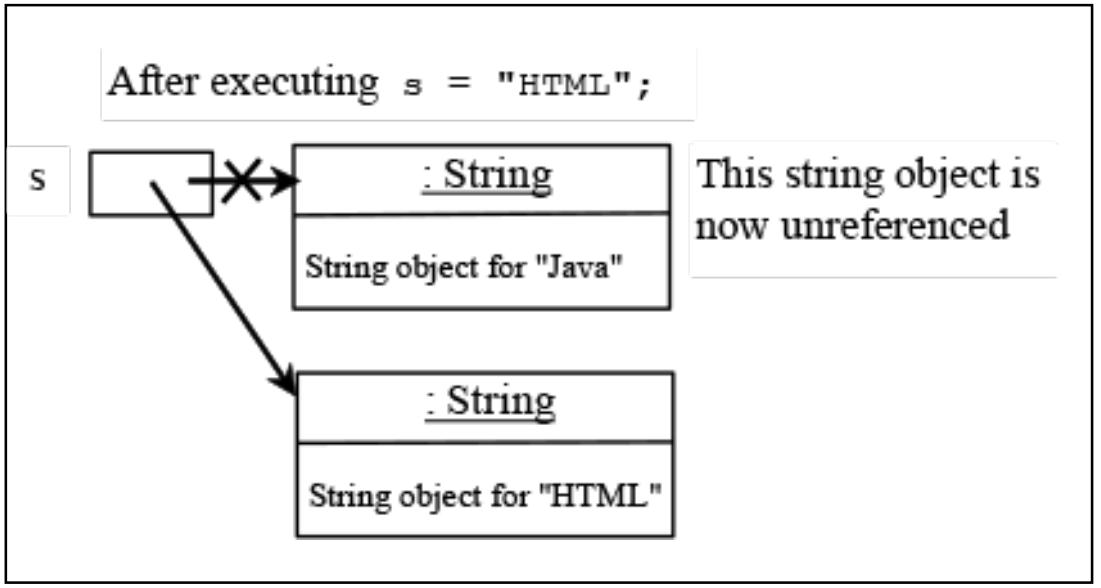


Trace Code

String s = "Java";

s = "HTML";





Interned Strings

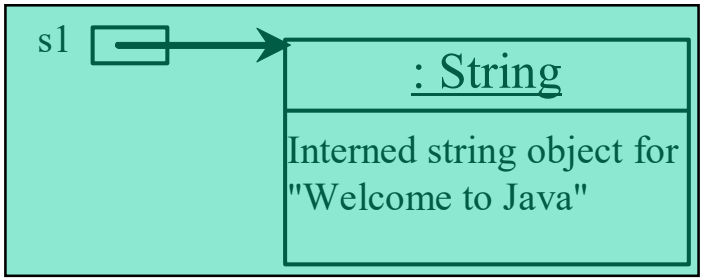
- Since strings are immutable and are frequently used, to improve efficiency and save memory, the JVM uses a unique instance for string literals with the same character sequence.
- Such an instance is called *interned*.

Example

```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
```

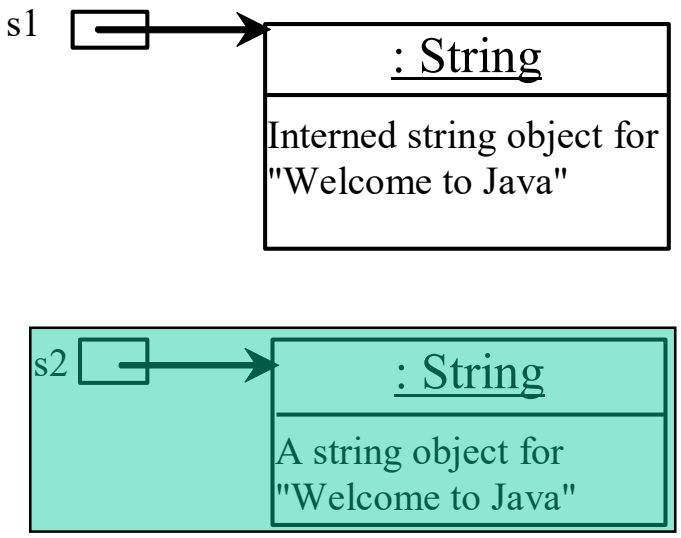
Example - Trace Code

```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
```



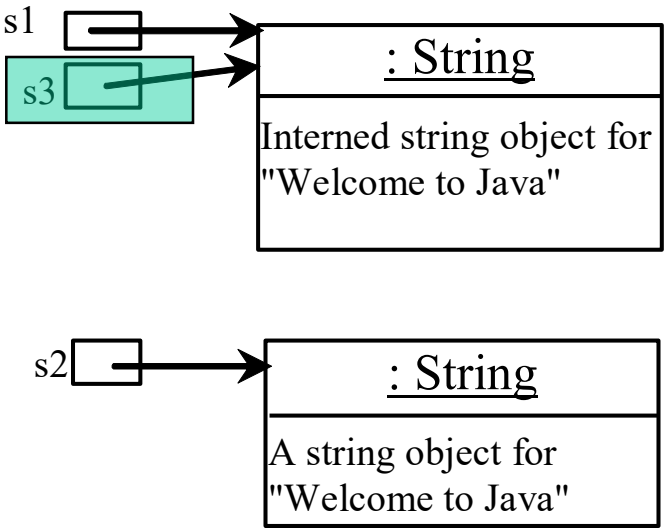
Example - Trace Code

```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
```



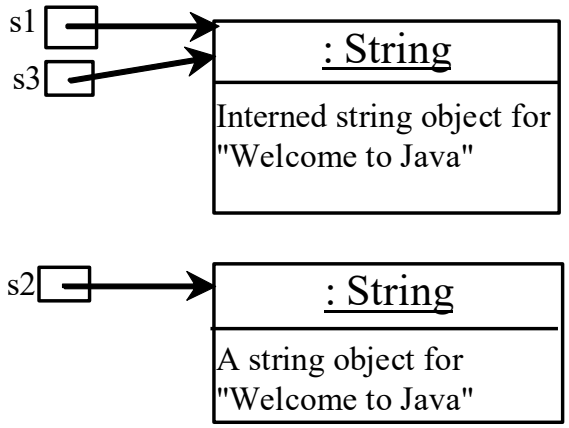
Example - Trace Code

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```



Example (cont.)

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



display

s1 == s2 is false

s1 == s3 is true

A new object is created if you use the new operator.

When you use the string initializer, no new object is created if the interned object is already created.

Replacing and Splitting Strings

java.lang.String	
+replace(oldChar: char, newChar: char): String	Returns a new string that replaces all matching character in this string with the new character.
+replaceFirst(oldString: String, newString: String): String	Returns a new string that replaces the first matching substring in this string with the new substring.
+replaceAll(oldString: String, newString: String): String	Returns a new string that replace all matching substrings in this string with the new substring.
+split(delimiter: String): String[]	Returns an array of strings consisting of the substrings split by the delimiter.

Example

```
"Welcome".replace('e', 'A')          // returns a new string, WAlcomA.

"Welcome".replaceFirst("e", "AB")    // returns a new string, WABlcome.

"Welcome".replaceAll("e", "AB")      // returns a new string, WABlcomAB.

"Welcome".replace("el", "AB")        // returns a new string, WABcome.
```

Splitting a String

```
String[] tokens = "Java#HTML#Perl".split("#");
for (int i = 0; i < tokens.length; i++)
  System.out.print(tokens[i] + " ");
```

Output:

Java HTML Perl

Matching, Replacing and Splitting by Patterns

- You can match, replace, or split a string by specifying a pattern. This is an extremely useful and powerful feature, commonly known as *regular expression*.
- Regular expression is complex to beginning students. For this reason, two simple patterns are used in this section.

```
"Java".matches("Java");          //true
"Java".equals("Java");           //true

"Java is fun".matches("Java.*");   //true
"Java is cool".matches("Java.*");  //true
```

Java Regular Expressions (Regex) Cheat Sheet: https://www.jrebel.com/blog/java-regular-expressions-cheat-sheet

Matching, Replacing and Splitting by Patterns

- The replaceAll, replaceFirst, and split methods can be used with a regular expression.
- For example, the following statement returns a new string that replaces $, +, or # in "a+b$#c" by the string NNN.

```
String s = "a+b$#c".replaceAll("[$+#]", "NNN");
System.out.println(s);
```

- Here the regular expression [$+#] specifies a pattern that matches $, +, or #. So, the output is aNNNbNNNNNNc.

Matching, Replacing and Splitting by Patterns

- The following statement splits the string into an array of strings delimited by some punctuation marks.

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]");

for (int i = 0; i < tokens.length; i++)
  System.out.println(tokens[i]);
```

Output:

Java
C
C#
C++

Convert Character and Numbers to Strings

- The String class provides several **static** valueOf methods for converting a character, an array of characters, and numeric values to strings.
- These methods have the same name valueOf with different argument types char, char[], double, long, int, and float.
- For example, to convert a double value to a string, use String.valueOf(5.44). The return value is string consists of characters ‘5’, ‘.’, ‘4’, and ‘4’.

Exercise (10 mins)

- Given two strings, write a method to decide if one is a permutation of the other.

```
hello vs. elolh
```

Answer

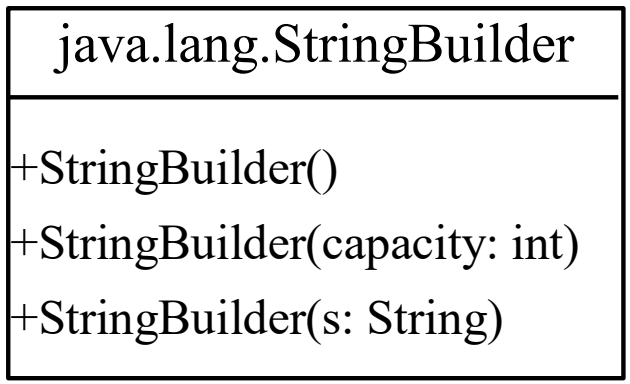
```java
import java.util.Scanner;

public class ClassExercise {

	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);

		System.out.print("1st string: ");
		String string1 = input.next();

		System.out.print("2nd string: ");
		String string2 = input.next();

		boolean result = PermutationCheck(string1, string2);

		if(result) {
			System.out.println("Permutation: Yes");
		}else {
			System.out.println("Permutation: No");
		}

		input.close();
	}

	public static String sort(String s) {
		char[] content = s.toCharArray();
		java.util.Arrays.sort(content);
		return new String(content);
	}

	public static boolean PermutationCheck(String a, String b) {

		if(a.length() != b.length()) {
			return false;
		}

		return sort(a).equals(sort(b));
	}
}
```

The `StringBuilder` and `StringBuffer` Classes

`StringBuilder` and `StringBuffer`

- The `StringBuilder`/`StringBuffer` class is an alternative to the String class.
- In general, a `StringBuilder`/`StringBuffer` can be used wherever a string is used.
- However, `StringBuilder`/`StringBuffer` is more flexible than String.
- You can add, insert, or append new contents into a string buffer, whereas the value of a `String` object is fixed once the string is created.

`StringBuilder` Constructors



Constructs an empty string builder with capacity 16.

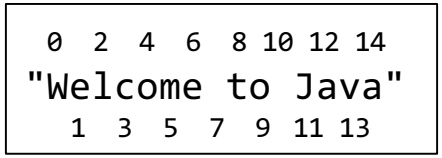
Constructs a string builder with the specified capacity.

Constructs a string builder with the specified string.

java.lang.StringBuilder	
+append(data: char[]): StringBuilder	Appends a char array into this string builder.
+append(data: char[], offset: int, len: int): StringBuilder	Appends a subarray in data into this string builder.
+append(v: *aPrimitiveType*): StringBuilder	Appends a primitive type value as a string to this builder.
+append(s: String): StringBuilder	Appends a string to this string builder.
+delete(startIndex: int, endIndex: int): StringBuilder	Deletes characters from startIndex to endIndex.
+deleteCharAt(index: int): StringBuilder	Deletes a character at the specified index.
+insert(index: int, data: char[], offset: int, len: int): StringBuilder	Inserts a subarray of the data in the array to the builder at the specified index.
+insert(offset: int, data: char[]): StringBuilder	Inserts data into this builder at the position offset.
+insert(offset: int, b: *aPrimitiveType*): StringBuilder	Inserts a value converted to a string into this builder.
+insert(offset: int, s: String): StringBuilder	Inserts a string into this builder at the position offset.
+replace(startIndex: int, endIndex: int, s: String): StringBuilder	Replaces the characters in this builder from startIndex to endIndex with the specified string.
+reverse(): StringBuilder	Reverses the characters in the builder.
+setCharAt(index: int, ch: char): void	Sets a new character at the specified index in this builder.

Exercise

```
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Welcome”);
stringBuilder.append(' ');
stringBuilder.append("to");
stringBuilder.append(' ');
stringBuilder.append("Java");   // "Welcome to Java"
```




```
stringBuilder.insert(11, "HTML and ")     // "Welcome to HTML and Java”
stringBuilder.delete(8, 11)               // "Welcome HTML and Java”

stringBuilder.deleteCharAt(8)             // "Welcome o Java”
stringBuilder.reverse()                   // "avaJ ot emocleW”
stringBuilder.replace(11, 15, "HTML")     // "Welcome to HTML”
stringBuilder.setCharAt(0, 'w')           // "welcome to Java"
```

Exercise (10 mins)

- Given a String[] of words: Dog, Cat, Fish, Bird, Horse

```
String[] myWords = {"Dog", "Cat", "Fish", "Bird", "Horse"};
```

- Please write a method `makeSentence()` that takes this String[] as an input parameter.
- Inside the method, please use `StringBuffer` to concatenate all words as a String and return it back.
- Print out the concatenated string

Answer

```
public class ClassExercise {

    public static void main(String[] args){
        String[] myWords = {"Dog", "Cat", "Fish", "Bird", "Horse"};
        System.out.println(makeSentence(myWords));
    }

    public static String makeSentence(String[] words){

        //You can replace StringBuilder with StringBuffer
        StringBuffer sentence = new StringBuffer();

        for(String w: words){
            sentence.append(w);
        }

        return sentence.toString();
    }
}
```

Exercise (offline)

- Implement a method to perform basic string compression using the counts of repeated characters.
- For example, the string aabccccbbb would become a2b1c4b3.
- If the “compressed” string would not become smaller than the original string, your method should return the original string.
- You can assume the string has only uppercase and lowercase letters (a-z).