STUDENT OUTLINE

Lesson 13 – String Class

INTRODUCTION: Strings are needed in many large programming tasks. Much of the information that identifies a person must be stored as a string: name, address, city, social security number, etc.

Although the `String` class is technically not an actual part of the Java language, a standard `String` class is provided with Java by Sun Microsystems, Inc.

Your job in this lesson is to learn the specifications of the `String` class and use it to solve string-processing questions.

The key topics for this lesson are:

A. The String Class
B. String Constructors
C. String Query Methods
D. String Translate Methods
E. Comparing Strings
F. Strings and Characters

VOCABULARY: String Class

DISCUSSION: A. <u>The `String` Class</u>

1. Character strings in Java are not represented by primitive types as are integers (**`int`**) or single characters (**`char`**). Strings are represented as objects of the `String` class. The `String` class is defined in `java.lang.String`, which is automatically imported for use in every program you write. We've used character string literals, such as `"Enter a value"` in earlier examples. Now we can begin to explore the `String` class and the capabilities that it offers.

2. A `String` is the only Java reference data type that has a built-in syntax for constants. These constants, referred to as string literals, consist of any sequence of characters enclosed within double quotations. For example:

```
"This is a string"
"Hello World!"
"\tHello World!\n"
```

The characters that a `String` object contains can include control characters. The last example contains a tab (`\t`) and a linefeed (`\n`) character, specified with escape sequences.

3. A second unique characteristic of the `String` type is that it supports the `"+"` operator to concatenate two `String` expressions. For example:

```
sentence = "I " + "want " + "to be a " + "Java programmer.";
```

The `"+"` operator can be used to combine a `String` expression with any other expression of primitive type. When this occurs, the primitive expression is converted to a `String` representation and concatenated with the string. For example, consider the following instruction sequence:

```
PI = 3.14159;
System.out.prinln("The value of PI is " + PI);
```

Run output:

```
The value of PI is 3.14159
```

4. `String` shares some of the characteristics with the primitive types, however, `String` *is a reference type, not a primitive type*. As a result, assigning one `String` to another copies the reference to the same `String` object. Similarly, each `String` method returns a new `String` object.

5. The rest of the student outline details a partial list of the methods provided by the Java `String` class. For a detailed listing of all of the capabilities of the class, please refer to Sun's Java documentation at:

   http://java.sun.com/j2se/1.4.1/docs/api/java/lang/String.html.

B. `String` Constructors

1. You create a `String` object by using the keyword **`new`** and the `String` constructor method, just as you would create an object of any other type.

Constructor	**Sample syntax**

`String();`	`// emptyString is a reference to an empty string`
	`String emptyString = new String();`

`String(String value);`	`String aGreeting;`
	
	`// aGreeting is reference to a String`
	`aGreeting = new String("Hello world");`
	
	`// Shortcut for constructing String objects.`
	`aGreeting = "Hello world";`
	
	`// anotherGreeting is a copy of aGreeting`
	`String anotherGreeting = new String(aGreeting);`

2. Though they are not primitive types, strings are so important and frequently used that Java provides additional syntax for declaration:

```
String aGreeting = "Hello world";
```

A `String` created in this short-cut way is called a *String literal.* Only `Strings` have a short-cut like this. All other objects are constructed by using the **`new`** operator.

C. String Query Methods

Query Method	Sample syntax

`int` `length();`	`String str1 = "Hello!"`
	`int` `len = str1.length();  //len == 6`

`char` `charAt(`**`int`** `index);`	`String str1 = "Hello!"`
	`char` `ch = str1.charAt(0); // ch == 'H'`

`int` `indexOf(String str);`	`String str2 = "Hi World!"`
	`int` `n = str2.indexOf("World"); // n == 3`
	`int` `n = str2.indexOf("Sun");   // n == -1`

`int` `indexOf(`**`int`** `ch);`	`String str2 = "Hi World!"`
	`int` `n = str2.indexOf('!');  // n == 8`
	`int` `n = str2.indexOf('T');  // n == -1`

1. The **`int`** `length()` method returns the number of characters in the `String` object.

2. The `charAt` method is a tool for extracting a character from within a `String`. The `charAt` parameter specifies the position of the desired character (0 for the leftmost character, 1 for the second from the left, etc.). For example, executing the following two instructions, prints the char value `'X'`.

```
String stringVar = "VWXYZ"
System.out.println(stringVar.charAt(2));
```

3. The **`int`** `indexOf(String str)` method will find the first occurrence of `str` within this `String` and return the index of the first character. If `str` does not occur in this `String`, the method returns -1.

4. The **`int`** `indexOf(`**`int`** `ch)` method is identical in function and output to the other `indexOf` function except it is looking for a single character.

D. String Translate Methods

Translate Method	**Sample syntax**

`String toLowerCase();`	`String greeting = "Hi World!";`
	`greeting.toLowerCase();`
	`// returns "hi world!"`

`String toUpperCase();`	`String greeting = "Hi World!";`
	`greeting.toUpperCase();`
	`// returns "HI WORLD!"`

`String trim();`	`String needsTrim = "  trim me!  ";`
	`needsTrim.trim();`
	`// returns "trim me!"`

`String substring(`**`int`**` beginIndex)`	`String sample = "hamburger";`
	`sample.substring(3);`
	`// returns "burger"`

`String substring(`**`int`**` beginIndex, `**`int`**` endIndex)`	`String sample = "hamburger";`
	`sample.substring(4, 8);`
	`// returns "urge"`

1. `toLowerCase()` returns a `String` with the same characters as the `String` object, but with all characters converted to lowercase.

2. `toUpperCase()` returns a `String` with the same characters as the string object, but with all characters converted to uppercase.

3. `trim()` returns a `String` with the same characters as the string object, but with the leading and trailing whitespace removed.

4. `substring(`**`int`**` beginIndex)` returns the substring of the string object starting from `beginIndex` through to the end of the `String` object.

5. `substring(`**`int`**` beginIndex, `**`int`**` endIndex)` returns the substring of the `String` object starting from `beginIndex` through, but not including, position `endIndex` of the `String` object. That is, the new string contains characters numbered `beginIndex` to `endIndex-1` in the original string.

E. Comparing Strings

1. The following methods should be used when comparing String objects:

Comparison Method	Sample Syntax

`boolean equals(String anotherString);`	`String aName = "Mat";`
	`String anotherName = "Mat";`
	`if (aName.equals(anotherName))`
	`  System.out.println("the same");`

`boolean equalsIgnoreCase(String anotherString);`	`String aName = "Mat";`
	`if (aName.equalsIgnoreCase("MAT"))`
	`  System.out.println("the same");`

`int compareTo(String anotherString)`	`String aName = "Mat"`
	`n = aName.compareTo("Rob"); // n < 0`
	`n = aName.compareTo("Mat"); // n == 0`
	`n = aName.compareTo("Amy"); // n > 0`

2. The `equals()` method evaluates the contents of two `String` objects to determine if they are equivalent. The method returns true if the objects have identical contents. For example, the code below shows two `String` objects and several comparisons. Each of the comparisons evaluate to **`true`**; each comparison results in printing the line `"Name's the same"`

```
String aName = "Mat";
String anotherName = "Mat";

if (aName.equals(anotherName))
  System.out.println("Name's the same");

if (anotherName.equals(aName))
  System.out.println("Name's the same");

if (aName.equals("Mat"))
  System.out.println("Name's the same");
```

Each `String` shown above, `aName` and `anotherName` is an object of type `String`, so each `String` has access to the `equals()` method. The `aName` object can call `equals()` with `aName.equals(anotherName)`, or the `anotherName` object can call `equals()` with `anotherName.equals(aName)`. The `equals()` method can take either a variable `String` object or a literal `String` as its argument.

3. The == operator can create some confusion when comparing `String` objects. Observe the following code segment and its output:

```
String aGreeting1 = new String("Hello");
String anotherGreeting1 = new String("Hello");

if (aGreeting1 == anotherGreeting1)
  System.out.println("This better not work!");
else
  System.out.println("This prints since each object " +
     "reference is different.");

String aGreeting2 = "Hello";
String anotherGreeting2 = "Hello";


if (aGreeting2 == anotherGreeting2)
  System.out.println("This prints since both " +
       "object references are the same!");
else
  System.out.println("This does not print.");
```

   Run Output:

```
This prints since each object reference is different.
This prints since both object references are the same!
```

   The objects `aGreeting1` and `anotherGreeting1` are each instantiated using the **`new`** command, which assigns a different reference to each object. The == operator compares the reference to each object, not their contents. Therefore the comparison `(aGreeting1 == anotherGreeting1)` returns **`false`** since the references are different.

   The objects `aGreeting2` and `anotherGreeting2` are `String` literals (created without the **`new`** command – i.e. using the short-cut instantiation process unique to `String`s). In this case, Java recognizes that the contents of the objects are the same and it creates only one instance, with `aGreeting2` and `anotherGreeting2` each referencing that instance. Since their references are the same, `(aGreeting2 == anotherGreeting2)` returns **`true`**.

   Because of potential problems like those described above, you should always use the `equals()` method to compare the contents of two `String` objects.

4. The `equalsIgnoreCase()` method is very similar to the `equals()` method. As its name implies, it ignores case when determining if two `Strings` are equivalent. This method is very useful when users type responses to prompts in your program. The `equalsIgnoreCase()` method allow you to test entered data without regard to capitalization.

5. The `compareTo()` method compares the calling `String` object and the `String` argument to see which comes first in the lexicographic ordering. Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase or all lowercase. If the calling string is first, it returns a negative value. If the two strings are equal, it returns zero. If the argument string is first, it returns a positive number.

E. Strings and Characters

1. It is natural to think of a **`char`** as a `String` of length 1. Unfortunately, in Java the **`char`** and `String` types are incompatible since a `String` is a reference type and a **`char`** is a primitive type. Some of the limitations of this incompatibility are:

   a. You *cannot* pass a **`char`** argument to a `String` parameter (nor the opposite).
   b. You *cannot* use a **`char`** constant in place of a `String` constant.
   c. You *cannot* assign a **`char`** expression to a `String` variable (nor the opposite).

   The last restriction is particularly troublesome, because there are many times in programs when **`char`** data and `String` data must be used cooperatively.

2. Extracting a **`char`** from within a `String` can be accomplished using the `charAt` method as previously described.

3. Conversion from **`char`** to `String` can be accomplished by using the `"+"` (concatenation) operator described previously. Concatenating any **`char`** with an *empty string* (`String` of length zero) results in a string that consists of that **`char`**. The java notation for an empty string is two consecutive double quotation marks. For example, the following expression

```
"" + 'X';
```

   evaluates to a `String` of length `1`, consisting of the letter `"X"`

```
// charVar is a variable of type char
// stringVar is a variable of type String
stringVar = "" + charVar;
```

   The execution of this instruction assigns to `stringVar` a `String` object that consist of the single character from `charVar`.

SUMMARY/ REVIEW:

The use of pre-existing code (classes) has helped reduce the time and cost of developing new programs. In this lesson you will use the `String` class without knowing its inner details. This is an excellent example of data type abstraction.

ASSIGNMENT: Lab Exercise, L.A.13.1, *Palindrome*
Lab Exercise, L.A.13.2, *Shorthand*
Lab Exercise, L.A.13.3, *Piglatinator*

LAB EXERCISE

Palindrome

“Madam, I’m Adam”
- Adam

“Men, I'm Eminem!"
- M. Salveit, 2000

"A man, a plan, a canal: Panama!"
- L. Mercer, 1914

"A man, a plan, a canoe, pasta, heros, rajahs, a coloratura, maps, snipe, percale, macaroni, a gag, a banana bag, a tan, a tag, a banana bag again (or a camel), a crepe, pins, Spam, a rut, a Rolo, cash, a jar, sore hats, a peon, a canal: Panama!"
- G. Jacobson, 1984

Background:

1. A word is a palindrome if it reads the same forwards and backwards. For example, the word

   ```
   level
   ```

   is a palindrome.

2. The idea of a palindrome can be extended to phrases or sentences if we ignore details like punctuation. Here are two familiar examples:

   ```
   Madam, I'm Adam
   A man, a plan, a canal: Panama
   ```

   We can recognize these more elaborate examples as palindromes by considering the text that is obtained by removing all spaces and punctuation marks and converting all letters to their lower-case form.

   ```
   Madam, I'm Adam                ==> madamimadam
   A man, a plan, a canal: Panama ==> amanaplanacanalpanama
   ```

3. If the "word" obtained from a phrase in this manner is a palindrome, then the phrase is a palindrome.

Assignment:

1. Write a program that inputs a string from the user, and then tell the user whether or not it is a palindrome.

2. Your program should ignore the case of the letter. In other words, an "a" and an "A" should be treated

the same.

3. A palindrome is determined by considering only alphabetic characters (a – z, A – Z) and numbers (0 – 9) as valid text.

4. An input string consisting of a single character is not considered to be a palindrome.

5. Your program should keep prompting the user for strings and printing whether or not the strings are palindromes until the user enters "Q" or "q" (which will signal termination of the program).

Instructions:

1. Use these sample phrases as inputs for your run outputs:

```
radar
J
Lewd did I live, & evil I did dwel.
I like Java
Straw? No, too stupid a fad, I put soot on warts.
```

2. Here are sample run outputs.

```
Welcome to the Palindrome Program!

Enter a string: radar
Yes, the string you entered is a palindrome.

Enter a string: J
No, the string you entered is NOT a palindrome.

Enter a string: Lewd did I live, & evil I did dwel.
Yes, the string you entered is a palindrome.

Enter a string: I like Java
No, the string you entered is NOT a palindrome.

Enter a string: Straw? No, too stupid a fad, I put soot on warts.
Yes, the string you entered is a palindrome.

Enter a string: Q
```

3. Turn in your source code and run outputs.

Note: The World’s Longest Palindrome, created at 8:02 PM on the 20th of February (a palindromic time/date - 20:02 02/20 2002) is reported at http://www.norvig.com/palindrome.html

LAB EXERCISE

Shorthand

"t s bttr 2 rmn slnt & thght fl, thn 2 spk p & rmv ll dbt." - A. Lincoln

Assignment:

1. Write a program that repeatedly accepts a line of input and produces the shorthand form of that line.

2. The shorthand form of a string is defined as follows:

   a. replace these four words: "and" with "&", "to" with "2", "you" with "U", and "for" with "4".
   b. remove all vowels ('a', 'e', 'i', 'o', 'u', whether lowercase or uppercase)

3. Your program should keep prompting the user for strings and printing out the shorthand version until the user enters "Q" or "q" (which will signal termination of the program).

Instructions:

1. Use these sample phrases as inputs for your run outputs:

```
You can pretend to be serious; you can't pretend to be witty
for For to you YOU and TO and
Humuhumunukunukuapua'a - Hawaiian state fish
2 + 2 = 4
This is FOR YOU TO try AND convert.
```

2. Here are sample run outputs.

```
Welcome to the Shorthand Message Generator!

Enter a message: You can pretend to be serious; you can't pretend to be witty
U cn prtnd 2 b srs; U cn't prtnd 2 b wtty

Enter a message: for For to you YOU and TO and
4 4 2 U U & 2 &

Enter a message: Humuhumunukunukuapua'a - Hawaiian state fish
Hmhmnknkp' - Hwn stt fsh

Enter a message: 2 + 2 = 4
2 + 2 = 4

Enter a message: This is FOR YOU TO try AND convert.
Ths s 4 U 2 try & cnvrt.

Enter a message: Q
```

3. Turn in your source code and run outputs.

LAB EXERCISE

Piglatinator

"Astahay alay istavay, abybay." - The Piglatinator

Assignment:

1. Your assignment is to create an English to Pig Latin translator.

2. Your program should begin by explaining its function to the user. It should then do the following until the user responds to the final question negatively:

   a. Ask the user to enter an English phrase.
   b. Translate the phrase word-by-word into Pig Latin. (Here, words are delineated by any combination of spaces, commas, periods, question marks, exclamation marks, semicolons, colons, hyphens, double quotes, or parentheses. A description of the algorithm for translating an English word into a Pig Latin word is given in part 3 below.)
   c. Tell the user the Pig Latin translation of the phrase they entered above.
   d. Ask the user if they would like to translate another English phrase.

3. Here's how to translate the English word `englishWord` into the Pig Latin word `pigLatinWord`:

   a. If there are no vowels in `englishWord`, then `pigLatinWord` is just `englishWord` + "ay". (There are ten vowels: 'a', 'e', 'i', 'o', and 'u', and their uppercase counterparts.)
   b. Else, if `englishWord` begins with a vowel, then `pigLatinWord` is just `englishWord` + "yay".
   c. Otherwise (if `englishWord` has a vowel in it and yet doesn't start with a vowel), then `pigLatinWord` is `end + start + "ay"`, where `end` and `start` are defined as follows:

      1. Let `start` be all of `englishWord` up to (but not including) its first vowel.
      2. Let `end` be all of `englishWord` from its first vowel on.
      3. But, if `englishWord` is capitalized, then capitalize `end` and "uncapitalize" `start`.

Instructions:

1. Use these sample phrases as inputs for your run outputs:

```
Hasta la vista baby. - the Terminator
System.out.println("I love Java");
I have never let my schooling interfere with my education. – Mark Twain
```

2. Here are sample run outputs.

```
I can translate English sentences and phrases into Pig Latin.
Please type an English sentence or phrase and then press <Enter>.
> Hasta la vista baby. - the Terminator

In Pig Latin that would be:
```

```
> Astahay alay istavay abybay. - ethay Erminatortay

Would you like to translate another phrase? y
```

```
Please type an English sentence or phrase and then press <Enter>.
> System.out.println("I love Java");

In Pig Latin that would be:
> Emsystay.outyay.intlnpray("Iyay ovelay Avajay");

Would you like to translate another phrase? y

Please type an English sentence or phrase and then press <Enter>.
> I have never let my schooling interfere with my education. - Mark Twain

In Pig Latin that would be:
> Iyay avehay evernay etlay myay oolingschay interfereyay ithway myay educationyay. -
Arkmay Aintway

Would you like to translate another phrase? N
```

3. Turn in your source code and run outputs.