|  |  |
| --- | --- |
| **Topic** | **Strings in Java** |
| **Learning Outcomes** | Strings are among the most common types of objects in Java. Throughout this chapter are various techniques for working with strings. We have seen how to create string variables, how to concatenate strings, and how to compare strings, but so far, we have only scratched the surface of what you can do with strings. In this chapter, We will dive deeper into what Java can do with stringAt the end of the lesson, The Students will be able to:   1. Understand the nature of Strings in Java 2. Work with substrings 3. Use StringBuilder and StringBuffer Class 4. Apply different methods of String class in writing a program 5. Do String Manipulation |
| **References** | Lemay, L., Perkins, C. L. (1996), Teach Yourself JAVA in 21 days,  Indianapolis, Indiana, Sams.net.  Doug Lowe, Java® All-in-One For Dummies®, 4th Edition), New Jersey,  John Wiley & Sons, Inc.  Barry Burd Ph.D., Java® For Dummies®, 7th Edition, New Jersey,  John Wiley & Sons, Inc.  David J. Eck, Introduction to Programming Using Java Version 5.0, December 2006  Geneva, NY, Hobart and William Smith Colleges  Jester Lhee I. Pandio, Computer Programming 1, Global Port Taguig City, STI College |

**Discussion**

**Java Strings**

String manipulation is arguably one of the most common activities in computer programming. This is especially true in Web systems, where Java is heavily used. In this chapter, we will look more deeply at what is certainly the most commonly used class in the language, String, along with some of its associated classes and utilities.

Java provides the String class from the java.lang package to create and manipulate strings. String has its own methods for working with strings.

**String**

A string is a sequence of characters. Every character in a string has a specific position in the string, and the position of the first character starts at index 0. The length of a string is the number of characters in it.

**Characteristics of Strings**

1. Strings are reference types, not value types, such as int or boolean. As a result, a string variable holds a reference to an object created from the String class, not the value of the string itself.
2. String class are immutable. Method in the class that appears to modify a String actually creates and returns a brand new String object containing the modification. The original String is left untouched.
3. Strings can include escape sequences that consist of a slash followed by another character. The most common escape sequences are \n for new line and \t for tab. If you want to include a slash in a string, you must use the escape sequence \\.

|  |  |
| --- | --- |
| **Table 6.1: Strings Escape Sequences** | |
| **Escape Sequence** | **Explanation** |
| \n | Newline |
| \t | Tab |
| \b | Backspace |
| \r | Carriage Return |
| \f | Form Feed |
| \’ | Apostrophe |
| \” | Quotation Mark |
| \\ | Backslash |

**Two ways on how to create Strings**

1. Directly assigning a string literal to a String object;

String strName = “Hidilyn Diaz”

1. Using the new keyword and String constructor to create a String object.

String sports= new String (“Weightlifting”);

**Overloading ‘+’ vs. StringBuilder**

Since String objects are immutable, you can alias to a particular String as many times as you want. Because a String is read-only, there is no possibility that one reference will change something that will affect the other references. Immutability can have efficiency issues. A case in point is the operator ‘+’ that has been overloaded for String objects. Overloading means that an operation has been given an extra meaning when used with a particular class. (The ‘+’ and ‘+=‘for String are the only operators that are overloaded in Java, and Java does not allow the programmer to overload any others.)1 The’+’ operator allows you to concatenate Strings:

**Example**

**Overloading**

String name = "Hidilyn Diaz";

String sports = "Weightlifting";

String achievement = "Olympic Gold Medalist";

String strFinal = name + " " + sports + " " + achievement;

// or String strFinal = "Hidilyn Diaz" + " " + "Weightlifting"+ " " + "Olympic Gold Medalist";

System.out.println(strFinal);

Note: You can also use the += to concatenate string

**String Builder**

StringBuilder strFinal = new StringBuilder();

strFinal.append("Hidilyn Diaz ");

strFinal.append("Weightlifting ");

strFinal.append("Olympic Gold Medalist");

System.out.println(strFinal);

**String Buffer**

StringBuffer bufferedString = new StringBuffer();

bufferedString.append("The ");

bufferedString.append("Quick ");

bufferedString.append("Brown ");

bufferedString.append("Fox ");

bufferedString.append("Jumps ");

bufferedString.append("Over ");

bufferedString.append("The ");

bufferedString.append("Lazy ");

bufferedString.append("Dog ");

System.out.println(bufferedString);

The difference between StringBuilder and StringBuffer is that StringBuffer is Synchronized also String Buffer is older than StringBuilder. StringBuilder is meant as a replacement to StringBuffer where synchronization is not necessary. We will discuss more about synchronization on the topic of Threads

**The String Methods**

The String class provides methods to perform operations on strings. Table 6.2 shows the list of some commonly used String methods in Java. These methods can only create and return a new string that contains the result of the operation.

Assume the following statement for the example of the table below

String string = “Java Programming”;

|  |  |  |
| --- | --- | --- |
| **Table 6.2: List of the most commonly used methods in class strings** | | |
| Method | Description | Example |
| charAt(index) | Returns the character of a string based on the specified index | string.charAt(2);//returns char ‘v’ |
| compareTo(string) | Compares this string to another string, using alphabetical order. Returns -1 if this string comes before the other string, 0 if the strings are the same, and 1 if this string comes after the other string. | String.compareTo(“JaVa Programming”)  /\*returns an integer value, the operation’s result is a Positive integer 32 \*/  // ASCII value 0f v – ASCII value 0f V |
| compareToIgnoreCase(String) | Similar to compareTo but ignores case. | String.compareToIgnoreCase(“JaVa Programming”)  // Returns 0 for equality |
| concat(string) | Returns a new string concatenated with the value of the parameter | string.concat(“ Techniques”);/\*returns a new string “Java programming Techniques”\*/ |
| contains(CharSequence) | Returns true if this string contains the parameter value. parameter can be a String, StringBuilder, or StringBuffer. | System.out.println(string.contains("Java"));  // Returns true – Note this is case sensitive |
| boolean endsWith(String) | Returns true if this string ends with the parameter string. | System.out.println(string.endsWith("ming"));  // Returns true |
| boolean equals(String) | Returns true if this string has the same value as the parameter string. | System.out.println(string.equals("JAva Programming"));  //Returns false – case sensitive |
| boolean equalsIgnoreCase(String) | Similar to equals but ignores case. | System.out.println(string.equals("JAva Programming"));  //Returns true |
| int indexOf(char) | Returns the index of the first occurrence of the char parameter in this string. Returns -1 if the character is not in the string. | System.out.println(string.indexOf("v"));  // Returns 2 – J = 0, a = 1, v = 2 |
| int indexOf(String) | Returns the index of the first occurrence of the String parameter in this string. Returns -1 if the string isn’t in this string | System.out.println(string.indexOf("gram"));  // Returns 8 |
| int indexOf(String, int start) | Similar to indexOf, but starts the search at the specified position in the string. | System.out.println(string.indexOf("gram", 5));  // Returns 8 – there is no much different between the previous one except that it starts at a designated position in the string. |
| int lastIndexOf(char) | Returns the index of the last occurrence of the char parameter in this string. Returns -1 if the character is not in the string. | System.out.println(string.lastIndexOf("a" ));  //returns 3 |
| int lastIndexOf(String) | Returns the index of the last occurrence of the String parameter in this string. Returns -1 if the string is not in this string. | System.out.println(string.lastIndexOf("a" )); |
| int lastIndexOf (String, int) | Similar to lastIndexOf, but starts the search at the specified position in the string. | System.out.println(string.lastIndexOf("Pro",5));  //returns 5 |
| int length() | Returns the length of this string. | System.out.println(string.length());  //returns 16 |
| String replace(char, char) | Returns a new string that is based on the original string, but with every occurrence of the first parameter replaced by the second parameter. | System.out.println(string.replace(‘a’, ‘i’));  // Returns Jivi Progrimming |
| String replaceAll(String old, String new) | Returns a new string that is based on the original string, but with every occurrence of the first string replaced by the second parameter. Note that the first parameter can be a regular expression. | System.out.println(string.replaceAll("Java", "C-Sharp"));  // C-Sharp Programming  Note : this method will replace all occurrence |
| String replaceFirst(String old, String new) | Returns a new string that is based on the original string, but with the first occurrence of the first string replaced by the second parameter. Note that the first parameter can be a regular expression. | System.out.println(string.replaceFirst("Prog", "Log"));  //Returns Java Logramming  Note: this method will only replace a single occurrence of the string. |
| String[] split(String) | Splits the string into an array of strings, using the string parameter as a pattern to determine where to split the strings. boolean | for(String n:string.split(" "))  System.out.println(n);  // returns Java  Programming  Note: the String is temporarily place in an array and access by the n string variable. |
|  |  |  |
| boolean startsWith(String) | Returns true if this string starts with the parameter string. boolean | System.out.println(string.startsWith("rog",));  //returns false |
| startsWith(String, int) | Returns true if this string contains the parameter string at the position indicated by the int parameter. | System.out.println(string.startsWith("gram",8));  //Returns true |
| String substring(int) | Extracts a substring from this string, beginning at the position indicated by the int parameter and continuing to the end of the string. | System.out.println(string.substring(4));  //return “ Programming”  Note to emphasize the space I used the “ “ |
| String substring(int, int) | Extracts a substring from this string, beginning at the position indicated by the first parameter and ending at the position one character before the value of the second parameter. | System.out.println(string.substring(8,12));  //returns gram  Note 12 is not the length of the string to be extracted but the position of the element or character in the string |
| char[] toCharArray() | Converts the string to an array of individual characters. | char[] characters = string.toCharArray();  //returns a character array  //{‘J’, ‘a’, ‘v’, ‘a’, ‘ ‘, ‘P’, ‘r’, ‘o’, ‘g’, ‘r’, ‘a’, ‘m’, ‘m’, ‘i’, ‘n’, ‘g’} |
| String toLowerCase() | Converts the string to lowercase. | System.out.println(string.toLowerCase());  // returns java programming |
| String toUpperCase() | Converts the string to uppercase. | System.out.println(string.toUpperCase());  // returns JAVA PROGRAMMING |
| String trim() | Returns a copy of the string with all leading and trailing white spaces removed. | System.out.println(string.trim());  // It will just remove the space in the beginning and in the end of the string. No middle spaces are remove |
| String valueOf(primitiveType) | Returns a string representation of any primitive type. | System.out.println(string.valueOf(6));  // returns a numeric string “6” |
|  |  |  |