**Strings in JAVA**



**String in Java:**

In Java a *string* is a sequence of characters. But, unlike many other languages that implement strings as character arrays, Java implements strings as objects of type **String**.

Implementing strings as built-in objects allows Java to provide a full complement of features that make string handling convenient. For example, Java has methods to compare two strings, search for a substring, concatenate two strings, and change the case of letters within a string. Also, String objects can be constructed a number of ways, making it easy to obtain a string when needed.

The **String**, **StringBuffer**, and **StringBuilder** classes are defined in **java.lang**. Thus, they are available to all programs automatically. All are declared final, which means that none of these classes may be subclassed. This allows certain optimizations that increase performance to take place on common string operations. All three implement the **CharSequence** interface.

**The String Constructors:**

The String class supports several constructors. To create an empty String, you call the default constructor. For example,

**String s = new String ();**

will create an instance of String with no characters in it.

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**String Length:**

The length of a string is the number of characters that it contains. To obtain this value, call the

**length( )** method, shown here:

**int length( );**

The following fragment prints “3”, since there are three characters in the string **s**:

char chars[] = { 'a', 'b', 'c' };

// Initializes the s String with abc, a way to initialize the String Object

String s = new String(chars);

System.out.println(s.length());

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**String Concatenation:**

In Java, + operator is used to concatenate two strings, producing a String object as theresult. This allows you to chain together a series of + operations. For example, the followingfragment concatenates three strings:

String age = "9";

String s = "He is " + age + " years old.";

System.out.println(s)

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**Character Extraction:**

The String class provides a number of ways in which characters can be extracted from aString object. Although the characters that comprise a string withina String object cannot be indexed as if they were a character array, many of the String methodsemploy an index (or offset) into the string for their operation. Like arrays, the string indexesbegin at zero.

**charAt( )**

To extract a single character from a String, you can refer directly to an individual charactervia the charAt( ) method. It has this general form:

charcharAt(int where)

Here, where is the index of the character that you want to obtain. The value of where must benonnegative and specify a location within the string. charAt( ) returns the character at thespecified location. For example,

charch;

ch = "abc".charAt(1);

**Output will be:????**

**getChars( )**

If you need to extract more than one character at a time, you can use the getChars( ) method.

It has this general form:

voidgetChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)

Here, sourceStart specifies the index of the beginning of the substring, and sourceEndspecifies an index that is one past the end of the desired substring. Thus, the substring contains the characters from sourceStart through sourceEnd–1. The array that will receive the charactersis specified by target. The index within target at which the substring will be copied is passedin targetStart. Care must be taken to assure that the target array is large enough to hold thenumber of characters in the specified substring.

String s = "This is a demo of the getChars method.";

int start = 10;

int end = 14;

charbuf[] = new char[end - start];

s.getChars(start, end, buf, 0);

System.out.println(buf);

**Output will be:????**

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**String Comparison:**

The String class includes several methods that compare strings or substrings within strings.

Each is examined here.

equals( ) and equalsIgnoreCase( )

To compare two strings for equality, use equals( ). It has this general form:

boolean equals(Object str)

Here, str is the String object being compared with the invoking String object. It returnstrue if the strings contain the same characters in the same order, and false otherwise. Thecomparison is case-sensitive.

To perform a comparison that ignores case differences, call equalsIgnoreCase( ). Whenit compares two strings, it considers A-Z to be the same as a-z. It has this general form:boolean

equalsIgnoreCase(String str)

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**substring( ):**

You can extract a substring using substring( ). It has two forms. The first is

String substring(int startIndex)

Here, startIndex specifies the index at which the substring will begin. This form returns a copy

of the substring that begins at startIndex and runs to the end of the invoking string.

The second form of substring( ) allows you to specify both the beginning and endingindex of the substring:

String substring(int startIndex, int endIndex)

Here, startIndex specifies the beginning index, and endIndex specifies the stopping point. The string returned contains all the characters from the beginning index, up to, but notincluding, the ending index.

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**startsWith( ) and endsWith( ):**

The startsWith( ) method determines whether a given String begins with a specified string.Conversely, endsWith( ) determines whether the String in question ends with a specifiedstring. They have the following general forms:

boolean**startsWith**(String str)

boolean**endsWith**(String str)

Here, str is the String being tested. If the string matches, true is returned. Otherwise, falseis returned. For example,

Here, str is the String being tested. If the string matches, true is returned. Otherwise, false is returned. For example,

"Foobar".endsWith("bar")

And

"Foobar".startsWith("Foo")

are both true.

Asecond form of startsWith( ), shown here, lets you specify a starting point:boolean

startsWith(String str, int startIndex)

Here, startIndex specifies the index into the invoking string at which point the search willbegin. For example,

"Foobar".startsWith("bar", 3)

returns true.

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**equals( ) Versus ==**

It is important to understand that the **equals( )** method and the **== operator** perform two different operations.

The **equals( ) method** compares the characters insidea String object.

The **== operator** compares two object references to see whether they referto the same instance.

The following program shows how two different String objects cancontain the same characters, but references to these objects will not compare as equal:

|  |
| --- |
| String s1 = "Hello";  String s2 = new String(s1);  System.out.println(s1 + " equals " + s2 + " -> " +  s1.equals(s2));  System.out.println(s1 + " == " + s2 + " -> " + (s1 == s2)); |

The variable s1 refers to the String instance created by “Hello”. The object referred to bys2 is created with s1 as an initializer. Thus, the contents of the two String objects are identical,but they are distinct objects. This means that s1 and s2 do not refer to the same objects andare, therefore, not ==, as is shown here by the output of the preceding example:

**Output:**

Hello equals Hello -> true

Hello == Hello -> false

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**compareTo( ):**

This method is simply used to know whether two strings are identical. For sortingapplications, you need to know which is less than, equal to, or greater than the next. A stringis less than another if it comes before the other in dictionary order. A string is greater thananother if it comes after the other in dictionary order. The String method compareTo( ) serves this purpose. It has this general form:

intcompareTo(String str)

Here, str is the String being compared with the

If you want to ignore case differences when comparing two strings, usecompareToIgnoreCase( ), as shown here:

intcompareToIgnoreCase(String str)

This method returns the same results as compareTo( ), except that case differences are ignored.

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**Searching Strings:**

The String class provides two methods that allow you to search a string for a specifiedcharacter or substring:

• indexOf( ) Searches for the first occurrence of a character or substring.

• lastIndexOf( ) Searches for the last occurrence of a character or substring.

To search for the first occurrence of a character, use

intindexOf(int ch)

To search for the last occurrence of a character, use

intlastIndexOf(int ch)

Here, ch is the character being sought.

To search for the first or last occurrence of a substring, use

intindexOf(String str)

intlastIndexOf(String str)

Here, str specifies the substring.You can specify a starting point for the search using these forms:

intindexOf(int ch, int startIndex)

intlastIndexOf(int ch, int startIndex)

intindexOf(String str, int startIndex)

intlastIndexOf(String str, int startIndex)

Here, startIndex specifies the index at which point the search begins. For indexOf( ), thesearch runs from startIndex to the end of the string. For lastIndexOf( ), the search runs fromstartIndex to zero.

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**concat( ):**

You can concatenate two strings using concat( ), shown here:

String concat(String str)

This method creates a new object that contains the invoking string with the contentsof str appended to the end. concat( ) performs the same function as +. For example,

String s1 = "one";

String s2 = s1.concat("two");

**OutPut:???????????**

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**replace();**

The replace( ) method has two forms. The first replaces all occurrences of one character inthe invoking string with another character. It has the following general form:

String replace(char original, char replacement)

Here, original specifies the character to be replaced by the character specified by replacement.

The resulting string is returned. For example,

String s = "Hello".replace('l', 'w');

**OutPut:???????????**

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**trim( )**

The trim( ) method returns a copy of the invoking string from which any leading and trailingwhitespace has been removed. It has this general form:

String trim( )

Here is an example:

String s = " Hello World ".trim();

This puts the string “Hello World” into s.

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**Changing the Case of Characters within a String**

The method **toLowerCase**( ) converts all the characters in a string from uppercase tolowercase. The **toUpperCase**( ) method converts all the characters in a string from lowercaseto uppercase. Non-alphabetical characters, such as digits, are unaffected. Here are the generalforms of these methods:

String toLowerCase( )

String toUpperCase( )

………………………..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………..

**Assignment:**

**Note:** You can use Internet; any reference Book etc to complete this assignment

1. Study StringBuffer Class (same as String Class) and implement all the methods of Buffer StringBuffer class
2. Implement all the Methods of Math Class
   1. Transcendental Functions
   2. Exponential Functions
   3. Rounding Functions
3. Implementation of Collections in Java
   1. Stack
   2. Queue
   3. Linked List
   4. Priority Queue
4. Email me the Screen Shots of the Java Code as well as output in MS Word/PDF format at

**[saif@uaar.edu.pk](mailto:saif@uaar.edu.pk)**