**1. String Basics**

As we know Strings are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

The Java platform provides the  **java.lang.String** class to create and manipulate strings.

**1.1 Create a String object**

There are two ways to create a String object:

* By string literal
* By new keyword

**Using String Literal**

Java String literal is created by using double-quotes.

For Example:

String s="javaguides";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If a string doesn't exist in the pool, a new string instance is created and placed in the pool.

For example:

String s1="javaguides";

String s2="javaguides";

//will not create new instance

Now the question is why java uses a concept of a string literal?

It's simple, to make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

**Using a new Keyword**

Let's create a simple example to demonstrate by creating String objects using the *new*keyword.

public static void main(String[] args) {

String str = new String("Java Guides");

// create String object using new Keyword

int length = str.length();

System.out.println(" length of the string '" + str + "' is :: " + length);

}

Output:

length of the string 'Java Guides' is:: 11

From the above example, JVM will create a new string object in normal(non-pool) heap memory and the literal "Java Guides" will be placed in the string constant pool. The variable *str* will refer to the object in heap(non-pool).

To create a String initialized by an array of characters, Here is an example:

char chars[] = {

'a',

'b',

'c'

}

;

String s = new String(chars);

Let's learn the String in depth by exploring all the String APIs with examples.

**1.2 String Class Hierarchy in Java**

String class implemented interfaces:  *Serializable, CharSequence, Comparable< String> .*Refer more detail on**java.lang.String**  JavaDoc.

|  |
| --- |
| **https://3.bp.blogspot.com/-KDjjgTAJKRY/W2UtFK6qGAI/AAAAAAAAC8A/lsnm_43KTyknark9wBAm8tm0h9Iahll6ACLcBGAs/s1600/string-hierachy.png** |
| *String class implemented interfaces* |

**2. The String Constructors**

The String class supports several constructors.

* *String()*  - Initializes a newly created String object so that it represents an empty character sequence.
* *String(byte[] bytes)*  - Constructs a new String by decoding the specified array of bytes using the platform's default charset.
* *String(byte[] bytes, Charset charset)*  - Constructs a new String by decoding the specified array of bytes using the specified charset.
* *String(byte[] bytes, int offset, int length)*  - Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.
* *String(byte[] bytes, int offset, int length, Charset charset)*  - Constructs a new String by decoding the specified subarray of bytes using the specified charset.
* *String(byte[] bytes, int offset, int length, String charsetName)*  - Constructs a new String by decoding the specified subarray of bytes using the specified charset.
* *String(byte[] bytes, String charsetName)*  - Constructs a new String by decoding the specified array of bytes using the specified charset.
* *String(char[] value)*  - Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.
* *String(char[] value, int offset, int count)*  - Allocates a new String that contains characters from a subarray of the character array argument.
* *String(int[] codePoints, int offset, int count)*  - Allocates a new String that contains characters from a subarray of the Unicode code point array argument.
* *String(String original)*  - Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
* *String(StringBuffer buffer)*  - Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.
* *String(StringBuilder builder)*  - Allocates a new string that contains the sequence of characters currently contained in the string builder argument.

Let's learn a few important String offered Constructors with examples.

1. **To create an empty String, call the default constructor.**  For example,

String s = new String();

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

1. **To create a String initialized by an array of characters.**  Here is an example:

char chars[] = {

'a',

'b',

'c'

}

;

String s = new String(chars);

This constructor initializes s with the string "abc".

1. **We can specify a subrange of a character array as an initializer using the following constructor:**

String(char chars[], int startIndex, int numChars)

Here, startIndex specifies the index at which the subrange begins, and numChars specifies the number of characters to use. Here is an example:

char chars[] = {

'a',

'b',

'c',

'd',

'e',

'f'

}

;

String s = new String(chars, 2, 3);

This initializes s with the character's *cde*.

1. **We can construct a String object that contains the same character sequence as another String object using this constructor:**

String(String strObj)

Here, *strObj* is a String object. Consider this example:

// Construct one String from another.

class MakeString {

public static void main(String args[]) {

char c[] = {

'J',

'a',

'v',

'a'

}

;

String s1 = new String(c);

String s2 = new String(s1);

System.out.println(s1);

System.out.println(s2);

}

}

The output from this program is as follows:

Java Java

As you can see, *s1* and *s2* contain the same string.

1. **String class provides constructors that initialize a string when given a byte array.**Two forms are shown here:

String(byte chrs[]) String(byte chrs[], int startIndex, int numChars)

Here, *chrs*specifies the array of bytes. The second form allows you to specify a subrange. In each of these constructors, the byte-to-character conversion is done by using the default character encoding of the platform. The following program illustrates these constructors:

// Construct string from subset of char array.

class SubStringCons {

public static void main(String args[]) {

byte ascii[] = {

65,

66,

67,

68,

69,

70

}

;

String s1 = new String(ascii);

System.out.println(s1);

String s2 = new String(ascii, 2, 3);

System.out.println(s2);

}

}

This program generates the following output:

ABCDEF CDE

1. **We can construct a  *String*  from a  *StringBuffer*  and StringBuilder using String constructors**

String(StringBuffer strBufObj) String(StringBuilder strBuildObj)

Examples:

String string = new String(new StringBuffer("JavaGuides"));

System.out.println(string);

String string2 = new String(new StringBuilder("JavaGuides"));

System.out.println(string2);

Output:

JavaGuides JavaGuides

**3. All String APIs/ Methods with Examples**

**charAt(int index)**

To extract a single character from a  *String*, you can refer directly to an individual character via the *charAt()*  method.

**Example 1:**  Returns the char value at the specified index of this string. The first char value is at index 0.

String str = "Welcome to string handling guide";

char ch1 = str.charAt(0);

char ch2 = str.charAt(5);

char ch3 = str.charAt(11);

char ch4 = str.charAt(20);

System.out.println("Character at 0 index is: " + ch1);

System.out.println("Character at 5th index is: " + ch2);

System.out.println("Character at 11th index is: " + ch3);

System.out.println("Character at 20th index is: " + ch4);

Output:

Character at 0 index is: W Character at 5th index is: m Character at 11th index is: s Character at 20th index is: n

**Example 2:**  Throws  *IndexOutOfBoundsException*  - if the index argument is negative or not less than the length of this string.

String str = "Java Guides";

char ch1 = str.charAt(str.length() + 1);

System.out.println("character :: " + ch1);

Output:

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 12 at java.lang.String.charAt(String.java:658) at com.javaguides.strings.methods.ChatAtExample.charAtExample2(ChatAtExample.java:26) at com.javaguides.strings.methods.ChatAtExample.main(ChatAtExample.java:6)

**Example 3:**  How to get the first and last character of the string

String str = "Java Guides";

int strLength = str.length();

// Fetching first character

System.out.println("Character at 0 index is: " + str.charAt(0));

// The last Character is present at the string length-1 index

System.out.println("Character at last index is: " + str.charAt(strLength - 1));

Output;

Character at 0 index is: J Character at last index is: s

**codePointAt(int index)**

This method returns the character (Unicode code point) at the specified index. The index refers to char values (Unicode code units) and ranges from 0 to length()- 1.

This method throws the  *IndexOutOfBoundsException*  - if the index argument is negative or not less than the length of this string.

**Example:**

public class CodePointAtExample {

public static void main(String[] args) {

String str = "javaguides";

int unicode = str.codePointAt(0);

System.out.println("the character (Unicode code point) at the specified index is :: " + unicode);

}

}

**Output:**

the character (Unicode code point) at the specified index is:: 106

**codePointBefore(int index)**

This method returns the character (Unicode code point) before the specified index. The index refers to char values (Unicode code units) and ranges from 1 to length.

This method throws the  *IndexOutOfBoundsException*  - if the index argument is negative or not less than the length of this string.

**Example:**

public class CodePointBeforeExample {

public static void main(String[] args) {

String str = "javaguides";

int unicode = str.codePointBefore(1);

System.out.println("the character (Unicode code point)" + " at the before specified index is :: " + unicode);

}

}

**Output:**

the character (Unicode code point) at the before specified index is:: 106

This method returns the number of Unicode code points in the specified text range of this String. The text range begins at the specified *beginIndex*and extends to the char at index *endIndex - 1*.

This method throws  *IndexOutOfBoundsException*  - if the beginIndex is negative, or endIndex is larger than the length of this String, or beginIndex is larger than endIndex.

Example:

public class CodePointCountExample {

public static void main(String[] args) {

String str = "javaguides";

System.out.println("length of the string :: " + str.length());

int unicode = str.codePointCount(0, str.length());

System.out.println("the character (Unicode code point) " + " at the specified index is :: " + unicode);

}

}

**Output:**

length of the string:: 10 the character (Unicode code point) at the specified index is:: 10

**compareTo(String anotherString)**

Often, it is not enough to simply know whether two strings are identical. For sorting applications, you need to know which is less than, equal to, or greater than the next. A string is less than another if it comes before the other in the dictionary order.  
  
A string is greater than another if it comes after the other in the dictionary order. The method *compareTo ()*  serves this purpose. It is specified by the **Comparable interface**, which String implements. It has this general form:

int compareTo(String str)

Here, str is the String being compared with the invoking String. The result of the comparison is returned as values meaning:

* Less than zero The invoking string is less than str.
* Greater than zero The invoking string is greater than str.
* Zero The two strings are equal.

**Example 1:** Here is a sample program that sorts an array of strings. The program uses  *compareTo()*  to determine sort ordering for a bubble sort:

// A bubble sort for Strings.

public class CompareToSecondExample {

static String arr[] = {

"Now",

"is",

"the",

"time",

"for",

"all",

"good",

"men",

"to",

"come",

"to",

"the",

"aid",

"of",

"their",

"country"

}

;

public static void main(String args[]) {

for (int j = 0; j < arr.length; j++) {

for (int i = j + 1; i < arr.length; i++) {

if (arr[i].compareTo(arr[j]) < 0) {

String t = arr[j];

arr[j] = arr[i];

arr[i] = t;

}

}

System.out.println(arr[j]);

}

}

}

The output of this program is the list of words:

Now aid all come country for good is men of the the their time to to

As you can see from the output of this example,   *compareTo()*takes into account uppercase and lowercase letters. The word "Now" came out before all the others because it begins with an uppercase letter, which means it has a lower value in the ASCII character set.

**Example 2:** compareTo method return different values example

String s1 = "Hello World";

String s2 = "Hello World";

String s3 = "Java";

String s4 = "Guides";

System.out.println(s1.compareTo(s2));

// 0 because both are equal

System.out.println(s1.compareTo(s3));

// -2 because "H" is 2 times lower than "J"

System.out.println(s1.compareTo(s4));

// 1 because "G" is 1 times greater than "H"

Output:

0 -2 1

**Example 3:** Compare the string with black or empty string using *compareTo()* method. Note that compare with an empty string returns length of the string.

String s1 = "hello";

String s2 = "";

String s3 = "me";

// compare with empty string, returns length of the string

System.out.println(s1.compareTo(s2));

// If first string is empty, result would be negative

System.out.println(s2.compareTo(s3));

Output:

5 -2

**compareToIgnoreCase(String str)**

Compares two strings lexicographically, ignoring case differences. This method returns an integer whose sign is that of calling compareTo with normalized versions of the strings where case differences have been eliminated by calling  *Character.toLowerCase(Character.toUpperCase(character))*on each character.

Example:

String s1="Hello World";

String s2="hello world";

String s3="Java";

String s4="java";

System.out.println(s1.compareToIgnoreCase(s2));

System.out.println(s3.compareToIgnoreCase(s4));

Output:

0 0

This method returns a negative integer, zero, or a positive integer as the specified String is greater than, equal to, or less than this String, ignoring case considerations.

**concat(String str)**

Concatenates the specified string to the end of this string.

This method creates a new object that contains the invoking string with the contents of str appended to the end. *concat()*  performs the same function as +.

**Example:**

public class ConcatExmaple {

public static void main(String[] args) {

String str = "javaguides";

str = str.concat(".net");

System.out.println("Concatenates the specified string to the end of this string : " + str);

System.out.println("cares".concat("s"));

System.out.println("to".concat("get"));

}

}

**Output:**

Concatenates the specified string to the end of this string: javaguides.net caress toget

**contains(CharSequence s)**

Returns *true* if and only if this string contains the specified sequence of char values.

**Example:**

public class ContainsExample {

public static void main(String[] args) {

String str = "javaguides";

boolean contains = str.contains("guides");

System.out.println("Contains : " + contains);

}

}

**Output:**

Contains: true

**contentEquals()**

There are two versions of contentEquals methods;

1. *contentEquals(CharSequence cs)*  - Compares this string to the specified  *CharSequence*.
2. *contentEquals(StringBuffer sb)*  - Compares this string to the specified  *StringBuffer*.

**Example:**

public class ContentEqualsExample {

public static void main(String[] args) {

String str = "javaguides";

String subStr = "javaguides";

boolean isContentEquals = str.contentEquals(subStr);

System.out.println("isContentEquals :: " + isContentEquals);

isContentEquals = str.contentEquals(new StringBuffer(subStr));

System.out.println("isContentEquals :: " + isContentEquals);

}

}

Output:

isContentEquals:: true isContentEquals:: true

**endsWith(String suffix)**

This method tests if this string ends with the specified suffix. Returns true if the character sequence represented by the argument is a suffix of the character sequence represented by this object; false otherwise.

Example:

public class EndsWithExample {

public static void main(String[] args) {

String str = "javaguides";

String subStr = "guides";

boolean endsWith = str.endsWith(subStr);

System.out.println(str + " endsWith " + subStr +" :: " + endsWith);

}

}

Output:

javaguides endsWith guides:: true

**equals(Object anObject) and equalsIgnoreCase(String anotherString)**

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

boolean equals(Object str)

Here, str is the String object is compared with the invoking String object. It returns true if the strings contain the same characters in the same order, and false otherwise. The comparison is case-sensitive.

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

boolean equalsIgnoreCase(String str)

Here, str is the String object is compared with the invoking String object. It, too, returns true if the strings contain the same characters in the same order, and false otherwise.

Here is an example that demonstrates  *equals()*  and  *equalsIgnoreCase():*

// Demonstrate equals() and equalsIgnoreCase().

class equalsDemo {

public static void main(String args[]) {

String s1 = "Hello";

String s2 = "Hello";

String s3 = "Good-bye";

String s4 = "HELLO";

System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));

System.out.println(s1 + " equals " + s3 + " -> " + s1.equals(s3));

System.out.println(s1 + " equals " + s4 + " -> " + s1.equals(s4));

System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " + s1.equalsIgnoreCase(s4));

}

}

The output from the program is shown here:

Hello equals Hello ->

true Hello equals Good-bye ->

false Hello equals HELLO ->

false Hello equalsIgnoreCase HELLO ->

true

Example 2:

public class EqualsExample {

public static void main(String[] args) {

String str = "javaguides";

String str1 = "javaguides";

String str3 = "javatutorial";

boolean equal = str.equals(str1);

System.out.println(" Is both string are equal :: " + equal);

}

}

Output:

Is both string are equal:: true

Example 3:

public class EqualsIgnoreCaseExample {

public static void main(String[] args) {

String str = "javaguides";

boolean equal = str.equalsIgnoreCase("JAVAguides");

System.out.println("Strings are equal :: " + equal);

}

}

Output:

Strings are equal:: true

**getBytes()**

There are four versions of *getBytes()*  methods. There is an alternative to *getChars()*  that stores the characters in an array of bytes.  *byte[] getBytes()*  - Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

*byte[] getBytes(Charset charset)*  - Encodes this String into a sequence of bytes using the given charset, storing the result into a new byte array.

*void getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin)*  - Deprecated.

*byte[] getBytes(String charsetName)*- Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

Let's write an example to demonstrate all the *getBytes()* methods.

public class GetBytesExamples {

public static void main(String[] args) throws UnsupportedEncodingException {

String str = "javaguides";

// Encodes this String into a sequence of bytes using the platform's

// default charset, storing the result into a new byte array.

byte[] bs = str.getBytes();

for (byte b : bs) {

System.out.println(b);

}

// Encodes this String into a sequence of bytes using the given charset,

// storing the result into a new byte array.

byte[] bs1 = str.getBytes(Charset.forName("UTF-8"));

for (byte b : bs1) {

System.out.println(b);

}

// Encodes this String into a sequence of bytes using the given charset,

// storing the result into a new byte array.

byte[] bs2 = str.getBytes("UTF-8");

for (byte b : bs2) {

System.out.println(b);

}

byte[] dest = new byte[str.length()];

str.getBytes(0, str.length(), dest, 0);

for (byte b : dest) {

System.out.println(b);

}

}

}

**getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)**

If you need to extract more than one character at a time, you can use the *getChars()* method. It has this general form:

void getChars(int sourceStart, int sourceEnd, char target[], int targetStart)

Here, *sourceStart* specifies the index of the beginning of the substring, and *sourceEnd* specifies 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 characters is specified by target. The index within the target at which the substring will be copied is passed in targetStart. Care must be taken to assure that the target array is large enough to hold the number of characters in the specified substring.

The following program demonstrates  *getChars():*

class getCharsDemo {

public static void main(String args[]) {

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

int start = 10;

int end = 14;

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

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

System.out.println(buf);

}

}

Here is the output of this program:

demo

**hashCode()**

Returns a hash code for this string. The hash code for a String object is computed as

s[0]\*31^(n-1) + s[1]\*31^(n-2) + ... + s[n-1]

using int arithmetic, where s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

Example:

public class HashcodeExample {

public static void main(String[] args) {

String str = "javaguides";

int hashcode = str.hashCode();

System.out.println("hashcode of " + str + " is :: " + hashcode);

}

}

Output:

hashcode of javaguides is:: -138203751

**indexOf()**

There are 4 types of an *indexOf* method in java. The signature of indexOf methods are given below:

* *indexOf(int ch)*  - Returns the index within this string of the first occurrence of the specified character.
* *indexOf(int ch, int fromIndex)*- Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
* *indexOf(String str) -*  Returns the index within this string of the first occurrence of the specified substring.
* *indexOf(String str, int fromIndex) -*  Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Example: This program demonstrates the example of all the 4  *indexOf()*  methods.

public class IndexOfExample {

public static void main(String[] args) {

String str = "javaguides";

// method 1

int index = str.indexOf("java");

System.out.println(index);

// Remember index starts with 0 so count from 0

System.out.println("index of guides :: " + str.indexOf("guides"));

System.out.println(" index of des :: " + str.indexOf("des"));

// method 2

System.out.println(str.indexOf('s'));

// method 3

System.out.println(str.indexOf('g', 0));

// method 4

System.out.println(str.indexOf("guides", 3));

}

}

Output:

0 index of guides:: 4 index of des:: 7 9 4 4

**intern()**

Returns a canonical representation for the string object.

A pool of strings, initially empty, is maintained privately by the class String.

Example:

public class InternExample {

public static void main(String[] args) {

String str = "javaguides";

String newStr = new String("javaguides");

System.out.println(newStr.intern().equals(str));

System.out.println(newStr.equals(str));

newStr.intern();

str.intern();

}

}

Output:

true true

**lastIndexOf() methods**

There are 4 types of a *lastIndexOf* method in java. The signature of *lastIndexOf* methods are given below:

* *lastIndexOf(int ch)*  - Returns the index within this string of the last occurrence of the specified character.
* *lastIndexOf(int ch, int fromIndex)*  - Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
* *lastIndexOf(String str)*- Returns the index within this string of the last occurrence of the specified substring.
* *lastIndexOf(String str, int fromIndex)*  - Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

The main usage of *lastIndexOf()*  - Searches for the last occurrence of a character or substring.

Example: This program demonstrates the usage of all 4  *lastIndexOf()*  methods.

public class LastIndexOfExample {

public static void main(String[] args) {

String str = "javaguides";

// method1

int lastIndexOf = str.lastIndexOf('s');

System.out.println(" last index of given character 's' in' " + " "+ str+"' :: " + lastIndexOf);

// method 2

lastIndexOf = str.lastIndexOf("guides");

System.out.println(" last index of given string 'guides' in' " + " "+ str+"' :: " + lastIndexOf);

// method 3

lastIndexOf = str.lastIndexOf("guides", 4);

System.out.println(" last index of guides in given string " + " "+ str+" and from index " + lastIndexOf);

// method 4

lastIndexOf = str.lastIndexOf('g', str.length());

System.out.println(" last index of given char :: " + lastIndexOf);

}

}

Output:

last index of given character 's' in' javaguides':: 9 last index of given string 'guides' in' javaguides':: 4 last index of guides in given string javaguides and from index 4 last index of given char:: 4

**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()

Example: Example to print the length of string "Java Guides".

public class LengthExample {

public static void main(String[] args) {

String str = new String("Java Guides");

int length = str.length();

System.out.println(" length of the string '" + str + "' is :: " + length);

}

}

Output:

length of the string 'Java Guides' is:: 11

**regionMatches()  methods**

There are two types of *regionMatches()*  methods.

* *regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)*- Tests if two string regions are equal.
* *regionMatches(int toffset, String other, int ooffset, int len)*  - Tests if two string regions are equal.

Example: Example to tests if two string regions are equal.

public class RegionMatchesExample {

public static void main(String[] args) {

String str = "javaguides";

String subStr = "guides";

boolean b = str.regionMatches(0, subStr, str.length(), str.length());

boolean b1 = str.regionMatches(true, 0, str, 0, str.length());

System.out.println(b);

System.out.println(b1);

}

}

Output:

false true

**replace()  methods**

The *replace()*  method has two forms. The first replaces all occurrences of one character in the 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');

puts the string "Hewwo" into s.

The second form of*replace()* replaces one character sequence with another. It has this general form:

String replace(CharSequence original, CharSequence replacement)

Example: This is a complete example to demonstrate the usage of*replace()* methods.

public class ReplaceExample {

public static void main(String[] args) {

String str = "javaguides";

String subStr = str.replace('a', 'b');

System.out.println("replace char 'a' with char 'b' from given string : " + subStr);

subStr = str.replace("guides", "tutorials");

System.out.println("replace guides with tutorials from given string : " + subStr);

subStr = str.replaceAll("[a-z]", "java");

System.out.println(subStr);

subStr = str.replaceFirst("[a-z]", "java");

System.out.println(subStr);

}

}

Output:

replace char 'a' with char 'b' from given string: jbvbguides replace guides with tutorials from given string: javatutorials javajavajavajavajavajavajavajavajavajava javaavaguides

**replaceAll(String regex, String replacement)**

Replaces each substring of this string that matches the given regular expression with the given replacement.

Example: This is a complete example to demonstrate the usage of *replaceAll()*  methods.

public class ReplaceExample {

public static void main(String[] args) {

String str = "javaguides";

String subStr = str.replaceAll("[a-z]", "java");

System.out.println(subStr);

}

}

Output:

javajavajavajavajavajavajavajavajavajava

**replaceFirst(String regex, String replacement)**

Replaces the first substring of this string that matches the given regular expression with the given replacement.

Example: This is a complete example to demonstrate the usage of*replaceFirst()*methods.

public class ReplaceExample {

public static void main(String[] args) {

String str = "javaguides";

String subStr = str.replaceFirst("[a-z]", "java");

System.out.println(subStr);

}

}

Output:

javaavaguides

**split() methods**

There are two form of split() methods.

* *split(String regex)*  Splits this string around matches of the given regular expression.
* *split(String regex, int limit)*  - Splits this string around matches of the given regular expression.

Example: This is a complete example to demonstrate the usage of *split()*  methods.

public class SplitExample {

public static void main(String[] args) {

String str = "java,guides.net";

String[] strArray = str.split(",");

for (int i = 0;

i <

strArray.length;

i++) {

System.out.println(strArray[i]);

}

strArray = str.split(",", 0);

for (int i = 0;

i <

strArray.length;

i++) {

System.out.println(strArray[i]);

}

}

}

Output:

java guides.net java guides.net

**startsWith() methods**

There are two forms of*startsWith()*methods.

* *startsWith(String prefix)*  - Tests if this string starts with the specified prefix.
* *boolean startsWith(String prefix, int toffset)*  - Tests if the substring of this string beginning at the specified index starts with the specified prefix.

The *startsWith()*  method determines whether a given String begins with a specified string.

Example: This is a complete example to demonstrate the usage of *startsWith()*  methods.

public class StartsWithExample {

public static void main(String[] args) {

String str = "javaguides";

boolean startWith = str.startsWith("ja");

System.out.println("startWith :: " +startWith);

// Remember index starts from 0

boolean startWithOffset = str.startsWith("guides", 4);

System.out.println("startWithOffset :: " + startWithOffset);

}

}

Output:

startWith:: true startWithOffset:: true

**subSequence(int beginIndex, int endIndex)**

Returns a character sequence that is a subsequence of this sequence.

An invocation of this method of the form

str.subSequence(begin, end)

behaves in exactly the same way as the invocation

str.substring(begin, end)

Example: This is a complete example to demonstrate the usage of *subSequence()* method.

public class SubSequenceExample {

public static void main(String[] args) {

String str = "javaguides";

CharSequence subStr = str.subSequence(0, str.length());

System.out.println("Returns a character sequence that " + " is a subsequence of this sequence : " + subStr);

}

}

Output:

Returns a character sequence that is a subsequence of this sequence: javaguides

**substring()  methods**

There are two forms of *substring()*  methods.

* *String substring(int beginIndex) -*  Returns a string that is a substring of this string.
* *String substring(int beginIndex, int endIndex)*  - Returns a string that is a substring of this string.

These methods throw *IndexOutOfBoundsException*- if beginIndex is negative or larger than the length of this String object.

Example: This is a complete example to demonstrate the usage of both *substring()*  methods.

public class SubStringExample {

public static void main(String[] args) {

String str = "javaguides";

// substring from start to end

String subStr = str.substring(0, str.length());

System.out.println("substring from 0 to length of the string : " + subStr);

subStr = str.substring(4);

System.out.println("Sub string starts from index 4 : " + subStr);

// Remember index starts from 0

System.out.println(str.substring(1));

System.out.println("unhappy".substring(2));

System.out.println("Harbison".substring(3));

System.out.println("emptiness".substring(8));

}

}

Output:

substring from 0 to length of the string: javaguides Sub string starts from index 4: guides avaguides happy bison s

**char[] java.lang.String.toCharArray()**

Converts this string to a new character array.

Example: This is a complete example to demonstrate the usage of *toCharArray()*  method.

public class ToCharArrayExample {

public static void main(String[] args) {

String str = "javaguides";

char[] characters = str.toCharArray();

for (char c : characters) {

System.out.println(c);

}

}

}

Output:

j a v a g u i d e s

**toLowerCase()  methods**

There are two forms of*toLowerCase()*methods.

* *toLowerCase()*  - Converts all of the characters in this String to lower case using the rules of the default locale.
* *String toLowerCase(Locale locale)*  - Converts all of the characters in this String to lower case using the rules of the given Locale.

Example: This is a complete example to demonstrate the usage of both *toLowerCase()* methods.

public class ToLowerCaseExample {

public static void main(String[] args) {

String str = "JAVAGUIDES";

String subStr = str.toLowerCase();

System.out.println(subStr);

subStr = str.toLowerCase(Locale.ENGLISH);

System.out.println(subStr);

}

}

Output:

javaguides javaguides

**toString()**

This object (which is already a string!) is itself returned. Example:

public class ToStringExample {

public static void main(String[] args) {

String str = "javaguides";

System.out.println(str.toString());

}

}

Output:

javaguides

**toUpperCase() methods**

There are two forms of toUpperCase() methods.

* *toUpperCase()*  - Converts all of the characters in this String to upper case using the rules of the default locale.
* *String toUpperCase(Locale locale)*- Converts all of the characters in this String to upper case using the rules of the given Locale.

Example: This is a complete example to demonstrate the usage of both *toUpperCase()* methods.

public class ToUpperCaseExample {

public static void main(String[] args) {

String str = "javaguides";

String subStr = str.toUpperCase();

System.out.println(subStr);

subStr = str.toUpperCase(Locale.ENGLISH);

System.out.println(subStr);

}

}

Output:

JAVAGUIDES JAVAGUIDES

**trim()**

Returns a string whose value is this string, with any leading and trailing whitespace removed.

Example: This is a complete example to demonstrate the usage of *trim()*method.

public class TrimExample {

public static void main(String[] args) {

String str = "javaguides ";

String subStr = str.trim();

System.out.println("trim the space from given string : " + subStr);

}

}

Output:

trim the space from given string: javaguides

**valueOf()**

The  *valueOf()*  method converts data from its internal format into a human-readable form.

The Java string valueOf *()*  method converts different types of values into a string. By the help of string valueOf *()*  method, you can convert *int* to *string*, *long* to *string*, *boolean* to *string*, *character* to *string*, *float* to a *string*, *double* to a *string*, *object* to *string* and *char* array to *string*.

Here are a few of its forms and the signature or syntax of string *valueOf()*method is given below:

public static String valueOf(boolean b) public static String valueOf(char c) public static String valueOf(char[] c) public static String valueOf(int i) public static String valueOf(long l) public static String valueOf(float f) public static String valueOf(double d) public static String valueOf(Object o)

Example: Let's see an example where we are converting all primitives and objects into strings.

public class StringValueOfExample5 {

public static void main(String[] args) {

boolean b1=true;

byte b2=11;

short sh = 12;

int i = 13;

long l = 14L;

float f = 15.5f;

double d = 16.5d;

char chr[]= {

'j',

'a',

'v',

'a'

}

;

StringValueOfExample5 obj=new StringValueOfExample5();

String s1 = String.valueOf(b1);

String s2 = String.valueOf(b2);

String s3 = String.valueOf(sh);

String s4 = String.valueOf(i);

String s5 = String.valueOf(l);

String s6 = String.valueOf(f);

String s7 = String.valueOf(d);

String s8 = String.valueOf(chr);

String s9 = String.valueOf(obj);

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

System.out.println(s4);

System.out.println(s5);

System.out.println(s6);

System.out.println(s7);

System.out.println(s8);

System.out.println(s9);

}

}

Output:

true 11 12 13 14 15.5 16.5 java StringValueOfExample5@2a139a55