CHP-13
Java Strings

Strings which are widely used in Java programming are a sequence of characters. In the Java programming language, strings are objects.
The Java platform provides the String class to create and manipulate strings.

Creating Strings:

The most direct way to create a string is to write:

```
String greeting ="Hello world!";
```

Whenever it encounters a string literal in your code, the compiler creates a String object with its value, in this case, "Hello world!'.
As with any other object, you can create String objects by using the new keyword and a constructor. The String class has eleven constructors that allow you to provide the initial value of the string using different sources, such as an array of characters:

```
public class StringDemo{
public static void main(String args[]){
char[] helloArray ={'h','e','l','l','o','.'};
String helloString =new String(helloArray);
System.out.println(helloString);
}
}
```

This would produce the following result:

```
hello.
```

Note: The String class is immutable, so that once it is created a String object cannot be changed. If there is a necessity to make alot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes.

String Length:

Methods used to obtain information about an object are known as accessor methods. One accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object.

CHAPTER

After the following two lines of code have been executed, len equals 17:

```
public class StringDemo{
public static void main(String args[]){
String palindrome ="Dot saw I was Tod";
int len = palindrome.length();
System.out.println("String Length is : "+ len );
}
}
```

This would produce the following result:

```
StringLengthis:17
```

Concatenating Strings:

The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

This returns a new string that is string1 with string2 added to it at the end. You can also use the concat() method with string literals, as in:

```
"My name is ".concat("Zara");
```

Strings are more commonly concatenated with the + operator, as in:

```
"Hello,"+" world"+"!"
```

which results in:

```
"Hello, world!"
```

Let us look at the following example:

```
public class StringDemo{
public static void main(String args[]){
String string1 ="saw I was ";
System.out.println("Dot "+ string1 +"Tod");
}
}
```

This would produce the following result:

```
Dot saw I was Tod
```

Creating Format Strings:

You have printf() and format() methods to print output with formatted numbers. The String class has an equivalent class method, format(), that returns a String object rather than a PrintStream object.

Using String's static format() method allows you to create a formatted string that you can reuse, as opposed to a one-time print statement. For example, instead of:

```
System.out.printf("The value of the float variable is "+
```



```
"%f, while the value of the integer "+
"variable is %d, and the string "+
"is %s", floatVar, intVar, stringVar);
```

you can write:

```
String fs;
fs =String.format("The value of the float variable is "+
"%f, while the value of the integer "+
"variable is %d, and the string "+
"is %s", floatVar, intVar, stringVar);
System.out.println(fs);
```

String Methods:

Here is the list of methods supported by String class:

SN Methods with Description

1 char charAt(int index)
Returns the character at the specified index.

2 int compareTo(Object o)
Compares this String to another Object.

3 int compareTo(String anotherString)
Compares two strings lexicographically.

4 int compareToIgnoreCase(String str)
Compares two strings lexicographically, ignoring case differences.

5 String concat(String str)
Concatenates the specified string to the end of this string.

6
boolean contentEquals(StringBuffer sb)
Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.

7 static String copyValueOf(char[] data)
Returns a String that represents the character sequence in the array specified.

8 static String copyValueOf(char[] data, int offset, int count)
Returns a String that represents the character sequence in the array specified.

9 boolean endsWith(String suffix)
Tests if this string ends with the specified suffix.
10 boolean equals(Object anObject)
Compares this string to the specified object.
11 boolean equalsIgnoreCase(String anotherString)
Compares this String to another String, ignoring case considerations.
12
byte getBytes()
Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13
byte[] getBytes(String charsetName
Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.

14 void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
Copies characters from this string into the destination character array.
15 int hashCode()
Returns a hash code for this string.
16 int indexOf(int ch)
Returns the index within this string of the first occurrence of the specified character.
17
int 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.
18 int indexOf(String str)
Returns the index within this string of the first occurrence of the specified substring.
19
int indexOf(String str, int fromIndex)
Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
20 String intern()
Returns a canonical representation for the string object.
21 int lastIndexOf(int ch)
Returns the index within this string of the last occurrence of the specified character.
22
int 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.
23 int lastIndexOf(String str)
Returns the index within this string of the rightmost occurrence of the specified substring.
24
int 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.
25 int length()
Returns the length of this string.
26 boolean matches(String regex)
Tells whether or not this string matches the given regular expression.
27 boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
Tests if two string regions are equal.
28 boolean regionMatches(int toffset, String other, int ooffset, int len)
Tests if two string regions are equal.
29
String replace(char oldChar, char newChar)
Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
30
String replaceAll(String regex, String replacement
Replaces each substring of this string that matches the given regular expression with the given replacement.
31

String replaceFirst(String regex, String replacement)
Replaces the first substring of this string that matches the given regular expression with the given replacement.

32 String[] split(String regex)
Splits this string around matches of the given regular expression.
33 String[] split(String regex, int limit)
Splits this string around matches of the given regular expression.
34 boolean startsWith(String prefix)
Tests if this string starts with the specified prefix.
35 boolean startsWith(String prefix, int toffset)
Tests if this string starts with the specified prefix beginning a specified index.
36 CharSequence subSequence(int beginIndex, int endIndex)
Returns a new character sequence that is a subsequence of this sequence.
37 String substring(int beginIndex)
Returns a new string that is a substring of this string.
38 String substring(int beginIndex, int endIndex)
Returns a new string that is a substring of this string.
39 char[] toCharArray()
Converts this string to a new character array.
40 String toLowerCase()
Converts all of the characters in this String to lower case using the rules of the default locale.
41 String toLowerCase(Locale locale)
Converts all of the characters in this String to lower case using the rules of the given Locale.
42 String toString()
This object (which is already a string!) is itself returned.
43 String toUpperCase()
Converts all of the characters in this String to upper case using the rules of the default locale.
44 String toUpperCase(Locale locale)
Converts all of the characters in this String to upper case using the rules of the given Locale.
45 String trim()
Returns a copy of the string, with leading and trailing whitespace omitted.
46 static String valueOf(primitive data type x)
Returns the string representation of the passed data type argument.
The above mentioned methods are explained here:

char charAt(int index)

Description:

This method returns the character located at the String's specified index. The string indexes start from zero.

Syntax:

Here is the syntax of this method:

```
public char charAt(int index)
```

TUTORIALS POINT Simply Easy Learning

Parameters:

Here is the detail of parameters:

- **index** -- Index of the character to be returned.

Return Value:

- This method Returns a char at the specified index.

Example:

```
public class Test{
public static void main(String args[]){
String s ="Strings are immutable";
char result = s.charAt(8);
System.out.println(result);
}
```

```
}
```
This produces the following result:
```
a
```
int compareTo(Object o)

Description:

There are two variants of this method. First method compares this String to another Object and second method compares two strings lexicographically.

Syntax:

Here is the syntax of this method:
```
int compareTo(Object o)
or
int compareTo(String anotherString)
```
Parameters:

Here is the detail of parameters:

- **o** -- the Object to be compared.
- **anotherString** -- the String to be compared.

Return Value :

- The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

Example:

```
public class Test{
public static void main(String args[]){
String str1 ="Strings are immutable";
String str2 ="Strings are immutable";
String str3 ="Integers are not immutable";
int result = str1.compareTo( str2 );
System.out.println(result);
result = str2.compareTo( str3 );
System.out.println(result);
result = str3.compareTo( str1 );
System.out.println(result);
}
}
```
This produces the following result:
```
0
10
-10
```
int compareTo(String anotherString)

Description:

There are two variants of this method. First method compares this String to another Object and second method compares two strings lexicographically.

Syntax:

Here is the syntax of this method:
```
int compareTo(Object o)
or
int compareTo(String anotherString)
```

Parameters:

Here is the detail of parameters:

- **o** -- the Object to be compared.
- **anotherString** -- the String to be compared.

Return Value :

- The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

Example:

```
public class Test{
public static void main(String args[]){
String str1 ="Strings are immutable";
String str2 ="Strings are immutable";
String str3 ="Integers are not immutable";
int result = str1.compareTo( str2 );
System.out.println(result);
result = str2.compareTo( str3 );
System.out.println(result);
result = str3.compareTo( str1 );
System.out.println(result);
}
}
```

This produces the following result:

```
0
10
-10
```

int compareToIgnoreCase(String str)

Description:

This method compares two strings lexicographically, ignoring case differences.

Syntax:

Here is the syntax of this method:

```
int compareToIgnoreCase(String str)
```

Parameters:

Here is the detail of parameters:

- **str** -- the String to be compared.

Return Value:

- 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.

Example:

```
public class Test{
public static void main(String args[]){
String str1 ="Strings are immutable";
String str2 ="Strings are immutable";
String str3 ="Integers are not immutable";
int result = str1.compareToIgnoreCase( str2 );
System.out.println(result);
result = str2.compareToIgnoreCase( str3 );
```

```
System.out.println(result);
result = str3.compareToIgnoreCase( str1 );
System.out.println(result);
}
}
```

This produces the following result:

```
0
10
-10
```

String concat(String str)

Description:

This method appends one String to the end of another. The method returns a String with the value of the String passed in to the method appended to the end of the String used to invoke this method.

Syntax:

Here is the syntax of this method:

```
public String concat(String s)
```

Parameters:

Here is the detail of parameters:

- **s** -- the String that is concatenated to the end of this String.

Return Value :

- This methods returns a string that represents the concatenation of this object's characters followed by the string argument's characters.

Example:

```
public class Test{
public static void main(String args[]){
String s ="Strings are immutable";
s = s.concat(" all the time");
System.out.println(s);
}
}
```

This produces the following result:

```
Strings are immutable all the time
```

boolean contentEquals(StringBuffer sb)

Description:

This method returns true if and only if this String represents the same sequence of characters as the specified in StringBuffer.

Syntax:

Here is the syntax of this method:

```
public boolean contentEquals(StringBuffer sb)
```

Parameters:

Here is the detail of parameters:

- **sb** -- the StringBuffer to compare.

Return Value:

- This method returns true if and only if this String represents the same sequence of characters as the specified in StringBuffer, otherwise false.

Example:

```
public class Test{
public static void main(String args[]){
String str1 ="Not immutable";
String str2 ="Strings are immutable";
StringBuffer str3 =new StringBuffer("Not immutable");
```



```
boolean result = str1.contentEquals( str3 );
System.out.println(result);
result = str2.contentEquals( str3 );
System.out.println(result);
}
}
```

This produces the following result:

```
true
false
```

static String copyValueOf(char[] data)

Description:

This method has two different forms:

- **public static String copyValueOf(char[] data):** Returns a String that represents the character sequence in the array specified.
- **public static String copyValueOf(char[] data, int offset, int count):** Returns a String that represents the character sequence in the array specified.

Syntax:

Here is the syntax of this method:

```
public staticString copyValueOf(char[] data)
or
public staticString copyValueOf(char[] data,int offset,int count)
```

Parameters:

Here is the detail of parameters:

- **data** -- the character array.
- **offset** -- initial offset of the subarray.
- **count** -- length of the subarray.

Return Value :

- This method returns a String that contains the characters of the character array.

Example:

```
public class Test{
public static void main(String args[]){
char[]Str1={'h','e','l','l','o',' ','w','o','r','l','d'};
```



```
String Str2="";
Str2=Str2.copyValueOf(Str1);
System.out.println("Returned String: "+Str2);
Str2=Str2.copyValueOf(Str1,2,6);
System.out.println("Returned String: "+Str2);
}
}
```

This produces the following result:

```
Returned String: hello world
Returned String: llo wo
```

boolean endsWith(String suffix)

Description:

This method tests if this string ends with the specified suffix.

Syntax:

Here is the syntax of this method:

```
public boolean endsWith(String suffix)
```

Parameters:

Here is the detail of parameters:

- **suffix** -- the suffix.

Return Value:

- This method returns true if the character sequence represented by the argument is a suffix of the character sequence represented by this object; false otherwise. Note that the result will be true if the argument is the empty string or is equal to this String object as determined by the equals(Object) method.

Example:

```
public class Test{
public static void main(String args[]){
String Str=new String("This is really not immutable!!");
boolean retVal;
retVal =Str.endsWith("immutable!!");
System.out.println("Returned Value = "+ retVal );
retVal =Str.endsWith("immu");
```



```
System.out.println("Returned Value = "+ retVal );
}
}
```

This produces the following result:

```
Returned Value = true
Returned Value = false
```

boolean equals(Object anObject)

Description:

This method compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.

Syntax:

Here is the syntax of this method:

```
public boolean equals(Object anObject)
```

Parameters:

Here is the detail of parameters:

- **anObject** -- the object to compare this String against.

Return Value :

- This method returns true if the String are equal; false otherwise.

Example:

```
public class Test{
public static void main(String args[]){
String Str1=new String("This is really not immutable!!");
String Str2=Str1;
String Str3=new String("This is really not immutable!!");
```

```
boolean retVal;
retVal =Str1.equals(Str2);
System.out.println("Returned Value = "+ retVal );
retVal =Str1.equals(Str3);
System.out.println("Returned Value = "+ retVal );
}
}
```

This produces the following result:

TUTORIALS POINT Simply Easy Learning

```
Returned Value = true
Returned Value = true
```

boolean equalsIgnoreCase(String anotherString)

Description:

This method compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

Syntax:

Here is the syntax of this method:

```
public boolean equalsIgnoreCase(String anotherString)
```

Parameters:

Here is the detail of parameters:

- **anotherString** -- the String to compare this String against

Return Value:

- This method returns true if the argument is not null and the Strings are equal, ignoring case; false otherwise.

Example:

```
public class Test{
public static void main(String args[]){
String Str1=new String("This is really not immutable!!");
String Str2=Str1;
String Str3=new String("This is really not immutable!!");
String Str4=new String("This IS REALLY NOT IMMUTABLE!!");
boolean retVal;
retVal =Str1.equals(Str2);
System.out.println("Returned Value = "+ retVal );
retVal =Str1.equals(Str3);
System.out.println("Returned Value = "+ retVal );
retVal =Str1.equalsIgnoreCase(Str4);
System.out.println("Returned Value = "+ retVal );
}
}
```

This produces the following result:

TUTORIALS POINT Simply Easy Learning

```
Returned Value = true
Returned Value = true
Returned Value = true
```

byte getBytes()

Description:

This method has following two forms:

- **getBytes(String charsetName):** Encodes this String into a sequence of bytes using the named charset,

storing the result into a new byte array.

 getBytes(): Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

Syntax:

Here is the syntax of this method:

```
public byte[] getBytes(String charsetName)
throwsUnsupportedEncodingException
or
public byte[] getBytes()
```

Parameters:

Here is the detail of parameters:

 charsetName -- the name of a supported charset.

Return Value:

 This method returns the resultant byte array

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str1=new String("Welcome to Tutorialspoint.com");
try{
byte[]Str2=Str1.getBytes();
System.out.println("Returned Value "+Str2);
Str2=Str1.getBytes("UTF-8");
System.out.println("Returned Value "+Str2);
```



```
Str2=Str1.getBytes("ISO-8859-1");
System.out.println("Returned Value "+Str2);
}catch(UnsupportedEncodingException e){
System.out.println("Unsupported character set");
}
}
}
```

This produces the following result:

```
Returned Value [B@192d342
Returned Value [B@15ff48b
Returned Value [B@1b90b39
```

byte[] getBytes(String charsetName)

Description:

This method has following two forms:

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

 getBytes(): Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

Syntax:

Here is the syntax of this method:

```
public byte[] getBytes(String charsetName)
throws UnsupportedEncodingException
or
public byte[] getBytes()
```

Parameters:

Here is the detail of parameters:

- **charsetName** -- the name of a supported charset.

Return Value:

- This method returns the resultant byte array

Example:

```
import java.io.*;
public class Test{
```



```
public static void main(String args[]){
String Str1=new String("Welcome to Tutorialspoint.com");
try{
byte[]Str2=Str1.getBytes();
System.out.println("Returned Value "+Str2);
Str2=Str1.getBytes("UTF-8");
System.out.println("Returned Value "+Str2);
Str2=Str1.getBytes("ISO-8859-1");
System.out.println("Returned Value "+Str2);
}catch(UnsupportedEncodingException e){
System.out.println("Unsupported character set");
}
}
}
```

This produces the following result:

```
Returned Value [B@192d342
Returned Value [B@15ff48b
Returned Value [B@1b90b39
```

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

Description:

This method copies characters from this string into the destination character array.

Syntax:

Here is the syntax of this method:

```
public void getChars(int srcBegin,
int srcEnd,
char[] dst,
int dstBegin)
```

Parameters:

Here is the detail of parameters:

- **srcBegin** -- index of the first character in the string to copy.
- **srcEnd** -- index after the last character in the string to copy.
- **dst** -- the destination array.
- **dstBegin** -- the start offset in the destination array.

Return Value:

- It does not return any value but throws IndexOutOfBoundsException.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str1=new String("Welcome to Tutorialspoint.com");
char[]Str2=newchar[7];
try{
Str1.getChars(2,9,Str2,0);
System.out.print("Copied Value = ");
System.out.println(Str2);
}
catch(Exception ex){
System.out.println("Raised exception...");
}
}
}
```

This produces the following result:

```
Copied Value = lcome t
```

int hashCode()

Description:

This method 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.)

Syntax:

Here is the syntax of this method:

```
public int hashCode()
```

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- This method returns a hash code value for this object.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.println("Hashcode for Str :"+Str.hashCode());
}
}
```

This produces the following result:

```
Hashcode for Str :1186874997
```

int indexOf(int ch)

Description:

This method has following different variants:

- **public int indexOf(int ch):** Returns the index within this string of the first occurrence of the specified character or -1 if the character does not occur.
- **public int 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 or -1 if the character does not occur.
- **int indexOf(String str):** Returns the index within this string of the first occurrence of the specified

substring. If it does not occur as a substring, -1 is returned.

- **int indexOf(String str, int fromIndex):** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If it does not occur, -1 is returned.

Syntax:

Here is the syntax of this method:

```
public int indexOf(int ch )
or
public int indexOf(int ch,int fromIndex)
or
int indexOf(String str)
or
int indexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:

- **ch** -- a character.
- **fromIndex** -- the index to start the search from.
- **str** -- a string.

Return Value:

- See the description.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
StringSubStr1=new String("Tutorials");
StringSubStr2=new String("Sutorials");
System.out.print("Found Index :");
System.out.println(Str.indexOf('o'));
System.out.print("Found Index :");
System.out.println(Str.indexOf('o',5));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1,15));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr2));
}
}
```

This produces the following result:

```
Found Index :4
Found Index :9
Found Index :11
Found Index :-1
Found Index :-1
```

int indexOf(int ch, int fromIndex)

Description:

This method has following different variants:

- **public int indexOf(int ch):** Returns the index within this string of the first occurrence of the specified character or -1 if the character does not occur.
- **public int 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 or -1 if the character does not occur.

 int indexOf(String str): Returns the index within this string of the first occurrence of the specified substring. If it does not occur as a substring, -1 is returned.

 int indexOf(String str, int fromIndex): Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If it does not occur, -1 is returned.

Syntax:

Here is the syntax of this method:

```
public int indexOf(int ch )
or
public int indexOf(int ch,int fromIndex)
or
int indexOf(String str)
or
int indexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:

 ch -- a character.

 fromIndex -- the index to start the search from.

 str -- a string.

Return Value:

 See the description.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
String SubStr1=new String("Tutorials");
String SubStr2=new String("Sutorials");
System.out.print("Found Index :");
System.out.println(Str.indexOf('o'));
System.out.print("Found Index :");
System.out.println(Str.indexOf('o',5));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1,15));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr2));
```



```
}
}
```

This produces the following result:

```
Found Index :4
Found Index :9
Found Index :11
Found Index :-1
Found Index :-1
```

int indexOf(String str)

Description:

This method has following different variants:

 public int indexOf(int ch): Returns the index within this string of the first occurrence of the specified character or -1 if the character does not occur.

 public int 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 or -1 if the character does not occur.
 int indexOf(String str): Returns the index within this string of the first occurrence of the specified substring. If it does not occur as a substring, -1 is returned.
 int indexOf(String str, int fromIndex): Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If it does not occur, -1 is returned.

Syntax:

Here is the syntax of this method:

```
public int indexOf(int ch )
or
public int indexOf(int ch,int fromIndex)
or
int indexOf(String str)
or
int indexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:
 ch -- a character.
 fromIndex -- the index to start the search from.
 str -- a string.

Return Value:

 See the description.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
StringSubStr1=new String("Tutorials");
StringSubStr2=new String("Sutorials");
System.out.print("Found Index :");
System.out.println(Str.indexOf('o'));
System.out.print("Found Index :");
System.out.println(Str.indexOf('o',5));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1,15));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr2));
}
}
```

This produces the following result:

```
Found Index :4
Found Index :9
Found Index :11
Found Index :-1
Found Index :-1
```

int indexOf(String str, int fromIndex)

Description:

This method has following different variants:
 public int indexOf(int ch): Returns the index within this string of the first occurrence of the specified

character or -1 if the character does not occur.

- **public int 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 or -1 if the character does not occur.
- **int indexOf(String str):** Returns the index within this string of the first occurrence of the specified substring. If it does not occur as a substring, -1 is returned.
- **int indexOf(String str, int fromIndex):** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. If it does not occur, -1 is returned.

Syntax:

Here is the syntax of this method:

```
public int indexOf(int ch )
or
public int indexOf(int ch,int fromIndex)
or
int indexOf(String str)
or
int indexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:

- **ch** -- a character.
- **fromIndex** -- the index to start the search from.
- **str** -- a string.

Return Value:

- See the description.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
String SubStr1=new String("Tutorials");
String SubStr2=new String("Sutorials");
System.out.print("Found Index :");
System.out.println(Str.indexOf('o'));
System.out.print("Found Index :");
System.out.println(Str.indexOf('o',5));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr1,15));
System.out.print("Found Index :");
System.out.println(Str.indexOf(SubStr2));
}
}
```

This produces the following result:

```
Found Index :4
```



```
Found Index :9
Found Index :11
Found Index :-1
Found Index :-1
```

String intern()

Description:

This method returns a canonical representation for the string object. It follows that for any two strings s and t, s.intern() == t.intern() is true if and only if s.equals(t) is true.

Syntax:

Here is the syntax of this method:

```
public String intern()
```

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- This method returns a canonical representation for the string object.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str1=new String("Welcome to Tutorialspoint.com");
String Str2=new String("WELCOME TO SUTORIALSPOINT.COM");
System.out.print("Canonical representation:");
System.out.println(Str1.intern());
System.out.print("Canonical representation:");
System.out.println(Str2.intern());
}
}
```

This produces the following result:

```
Canonical representation: Welcome to Tutorialspoint.com
Canonical representation: WELCOME TO SUTORIALSPOINT.COM
```


int lastIndexOf(int ch)

Description:

This method has the following variants:

- **int lastIndexOf(int ch):** Returns the index within this string of the last occurrence of the specified character or -1 if the character does not occur.
- **public int lastIndexOf(int ch, int fromIndex):** Returns the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to fromIndex, or -1 if the character does not occur before that point.
- **public int lastIndexOf(String str):** If the string argument occurs one or more times as a substring within this object, then it returns the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned.
- **public int 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.

Syntax:

Here is the syntax of this method:

```
int lastIndexOf(int ch)
or
public int lastIndexOf(int ch,int fromIndex)
or
public int lastIndexOf(String str)
or
public int lastIndexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:

- **ch** -- a character.
- **fromIndex** -- the index to start the search from.
- **str** -- A string.

Return Value:

- This method returns the index.

Example:

```
import java.io.*;
public class Test{
```



```
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
String SubStr1=new String("Tutorials");
String SubStr2=new String("Sutorials");
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o'));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o',5));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1,15));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr2));
}
}
```

This produces the following result:

```
Found Last Index :27
Found Last Index :4
Found Last Index :11
Found Last Index :11
Found Last Index :-1
```

int lastIndexOf(int ch, int fromIndex)

Description:

This method has the following variants:

- **int lastIndexOf(int ch):** Returns the index within this string of the last occurrence of the specified character or -1 if the character does not occur.
- **public int lastIndexOf(int ch, int fromIndex):** Returns the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to fromIndex, or -1 if the character does not occur before that point.
- **public int lastIndexOf(String str):** If the string argument occurs one or more times as a substring within this object, then it returns the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned.
- **public int 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.

Syntax:

Here is the syntax of this method:

```
int lastIndexOf(int ch)
or
public int lastIndexOf(int ch,int fromIndex)
```



```
or
public int lastIndexOf(String str)
```

```
or
public int lastIndexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:

- **ch** -- a character.
- **fromIndex** -- the index to start the search from.
- **str** -- A string.

Return Value:

- This method returns the index.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
String SubStr1=new String("Tutorials");
String SubStr2=new String("Sutorials");
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o'));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o',5));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1,15));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr2));
}
}
```

This produces the following result:

```
Found Last Index :27
Found Last Index :4
Found Last Index :11
Found Last Index :11
Found Last Index :-1
```


int lastIndexOf(String str)

Description:

This method has the following variants:

- **int lastIndexOf(int ch):** Returns the index within this string of the last occurrence of the specified character or -1 if the character does not occur.
- **public int lastIndexOf(int ch, int fromIndex):** Returns the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to fromIndex, or -1 if the character does not occur before that point.
- **public int lastIndexOf(String str):** If the string argument occurs one or more times as a substring within this object, then it returns the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned.
- **public int 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.

Syntax:

Here is the syntax of this method:

```
int lastIndexOf(int ch)
or
```

```
public int lastIndexOf(int ch,int fromIndex)
or
public int lastIndexOf(String str)
or
public int lastIndexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:

- **ch** -- a character.
- **fromIndex** -- the index to start the search from.
- **str** -- A string.

Return Value:

- This method returns the index.

Example:

```
import java.io.*;
public class Test{
```



```
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
String SubStr1=new String("Tutorials");
String SubStr2=new String("Sutorials");
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o'));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o',5));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1,15));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr2));
}
}
```

This produces the following result:

```
Found Last Index :27
Found Last Index :4
Found Last Index :11
Found Last Index :11
Found Last Index :-1
```

int lastIndexOf(String str, int fromIndex)

Description:

This method has the following variants:

- **int lastIndexOf(int ch):** Returns the index within this string of the last occurrence of the specified character or -1 if the character does not occur.
- **public int lastIndexOf(int ch, int fromIndex):** Returns the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to fromIndex, or -1 if the character does not occur before that point.
- **public int lastIndexOf(String str):** If the string argument occurs one or more times as a substring within this object, then it returns the index of the first character of the last such substring is returned. If it does not occur as a substring, -1 is returned.
- **public int 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.

Syntax:

Here is the syntax of this method:

```
int lastIndexOf(int ch)
or
public int lastIndexOf(int ch,int fromIndex)
```



```
or
public int lastIndexOf(String str)
or
public int lastIndexOf(String str,int fromIndex)
```

Parameters:

Here is the detail of parameters:

- **ch** -- a character.
- **fromIndex** -- the index to start the search from.
- **str** -- A string.

Return Value:

- This method returns the index.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
String SubStr1=new String("Tutorials");
String SubStr2=new String("Sutorials");
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o'));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf('o',5));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr1,15));
System.out.print("Found Last Index :");
System.out.println(Str.lastIndexOf(SubStr2));
}
}
```

This produces the following result:

```
Found Last Index :27
Found Last Index :4
Found Last Index :11
Found Last Index :11
Found Last Index :-1
```


int length()

Description:

This method returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

Syntax:

Here is the syntax of this method:

```
public int length()
```

Parameters:

Here is the detail of parameters:

 NA

Return Value:

 This method returns the the length of the sequence of characters represented by this object.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str1=new String("Welcome to Tutorialspoint.com");
String Str2=new String("Tutorials");
System.out.print("String Length :");
System.out.println(Str1.length());
System.out.print("String Length :");
System.out.println(Str2.length());
}
}
```

This produces the following result:

```
String Length :29
String Length :9
```

boolean matches(String regex)

Description:

TUTORIALS POINT Simply Easy Learning

This method tells whether or not this string matches the given regular expression. An invocation of this method of the form str.matches(regex) yields exactly the same result as the expression Pattern.matches(regex, str).

Syntax:

Here is the syntax of this method:

```
public boolean matches(String regex)
```

Parameters:

Here is the detail of parameters:

 regex -- the regular expression to which this string is to be matched.

Return Value:

 This method returns true if, and only if, this string matches the given regular expression.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.matches("(.*)Tutorials(.*)"));
System.out.print("Return Value :");
System.out.println(Str.matches("Tutorials"));
System.out.print("Return Value :");
System.out.println(Str.matches("Welcome(.*)"));
}
}
```

This produces the following result:

```
Return Value :true
Return Value :false
Return Value :true
```

boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)

Description:

This method has two variants which can be used to test if two string regions are equal.

Syntax:

Here is the syntax of this method:

```
public boolean regionMatches(int toffset,
String other,
int ooffset,
int len)
or
public boolean regionMatches(boolean ignoreCase,
int toffset,
String other,
int ooffset,
int len)
```

Parameters:

Here is the detail of parameters:

- **toffset** -- the starting offset of the subregion in this string.
- **other** -- the string argument.
- **ooffset** -- the starting offset of the subregion in the string argument.
- **len** -- the number of characters to compare.
- **ignoreCase** -- if true, ignore case when comparing characters.

Return Value:

It returns true if the specified subregion of this string matches the specified subregion of the string argument; false otherwise. Whether the matching is exact or case insensitive depends on the ignoreCase argument.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str1=new String("Welcome to Tutorialspoint.com");
String Str2=new String("Tutorials");
String Str3=new String("TUTORIALS");
System.out.print("Return Value :");
System.out.println(Str1.regionMatches(11,Str2,0,9));
System.out.print("Return Value :");
System.out.println(Str1.regionMatches(11,Str3,0,9));
System.out.print("Return Value :");
System.out.println(Str1.regionMatches(true,11,Str3,0,9));
}
```



```
}
```

This produces the following result:

```
Return Value :true
Return Value :false
Return Value :true
```

boolean regionMatches(int toffset, String other, int ooffset, int len)

Description:

This method has two variants which can be used to test if two string regions are equal.

Syntax:

Here is the syntax of this method:

```
public boolean regionMatches(int toffset,
String other,
int ooffset,
int len)
or
public boolean regionMatches(boolean ignoreCase,
int toffset,
String other,
int ooffset,
int len)
```

Parameters:

Here is the detail of parameters:

- **toffset** -- the starting offset of the subregion in this string.
- **other** -- the string argument.
- **ooffset** -- the starting offset of the subregion in the string argument.
- **len** -- the number of characters to compare.
- **ignoreCase** -- if true, ignore case when comparing characters.

Return Value:

It returns true if the specified subregion of this string matches the specified subregion of the string argument; false otherwise. Whether the matching is exact or case insensitive depends on the ignoreCase argument.

Example:

TUTORIALS POINT Simply Easy Learning

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str1=new String("Welcome to Tutorialspoint.com");
String Str2=new String("Tutorials");
String Str3=new String("TUTORIALS");
System.out.print("Return Value :");
System.out.println(Str1.regionMatches(11,Str2,0,9));
System.out.print("Return Value :");
System.out.println(Str1.regionMatches(11,Str3,0,9));
System.out.print("Return Value :");
System.out.println(Str1.regionMatches(true,11,Str3,0,9));
}
}
```

This produces the following result:

```
Return Value :true
Return Value :false
Return Value :true
```

String replace(char oldChar, char newChar)

Description:

This method returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

Syntax:

Here is the syntax of this method:

```
public String replace(char oldChar,char newChar)
```

Parameters:

Here is the detail of parameters:

- **oldChar** -- the old character.
- **newChar** -- the new character.

Return Value:

- It returns a string derived from this string by replacing every occurrence of oldChar with newChar.

Example:

```
import java.io.*;
```



```
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.replace('o','T'));
System.out.print("Return Value :");
System.out.println(Str.replace('l','D'));
}
}
```

This produces the following result:

```
Return Value :WelcTme tT TutTrialspTint.cTm
Return Value :WeDcome to TutoriaDspoint.com
```

String replaceAll(String regex, String replacement)

Description:

This method replaces each substring of this string that matches the given regular expression with the given replacement.

Syntax:

Here is the syntax of this method:

```
public String replaceAll(String regex,String replacement)
```

Parameters:

Here is the detail of parameters:

- **regex** -- the regular expression to which this string is to be matched.
- **replacement** -- the string which would replace found expression.

Return Value:

- This method returns the resulting String.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
```



```
System.out.println(Str.replaceAll("(.*)Tutorials(.*)",
"AMROOD"));
}
}
```

This produces the following result:

```
Return Value :AMROOD
```

String replaceFirst(String regex, String replacement)

Description:

This method replaces the first substring of this string that matches the given regular expression with the given replacement.

Syntax:

Here is the syntax of this method:

```
public String replaceFirst(String regex,String replacement)
```

Parameters:

Here is the detail of parameters:

- **regex** -- the regular expression to which this string is to be matched.
- **replacement** -- the string which would replace found expression.

Return Value :

- This method returns a resulting String.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.replaceFirst("(.*)Tutorials(.*)",
"AMROOD"));
System.out.print("Return Value :");
System.out.println(Str.replaceFirst("Tutorials","AMROOD"));
}
}
```

This produces the following result:


```
Return Value :AMROOD
Return Value :Welcome to AMROODpoint.com
```

String[] split(String regex)

Description:

This method has two variants and splits this string around matches of the given regular expression.

Syntax:

Here is the syntax of this method:

```
public String[] split(String regex,int limit)
or
public String[] split(String regex)
```

Parameters:

Here is the detail of parameters:

- **regex** -- the delimiting regular expression.
- **limit** -- the result threshold which means how many strings to be returned.

Return Value:

- It returns the array of strings computed by splitting this string around matches of the given regular expression.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome-to-Tutorialspoint.com");
System.out.println("Return Value :");
for(String retval:Str.split("-",2)){
System.out.println(retval);
}
System.out.println("");
System.out.println("Return Value :");
for(String retval:Str.split("-",3)){
System.out.println(retval);
}
System.out.println("");
System.out.println("Return Value :");
for(String retval:Str.split("-",0)){
System.out.println(retval);
```



```
}
System.out.println("");
System.out.println("Return Value :");
for(String retval:Str.split("-")){
System.out.println(retval);
}
}
}
```

This produces the following result:

```
Return Value :
Welcome
to-Tutorialspoint.com
Return Value :
Welcome
to
Tutorialspoint.com
Return Value:
Welcome
to
Tutorialspoint.com
Return Value :
Welcome
to
Tutorialspoint.com
```

String[] split(String regex, int limit)

Description:

This method has two variants and splits this string around matches of the given regular expression.

Syntax:

Here is the syntax of this method:

```
public String[] split(String regex,int limit)
or
public String[] split(String regex)
```

Parameters:

Here is the detail of parameters:

- **regex** -- the delimiting regular expression.
- **limit** -- the result threshold which means how many strings to be returned.

Return Value:

- It returns the array of strings computed by splitting this string around matches of the given regular expression.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome-to-Tutorialspoint.com");
System.out.println("Return Value :");
for(String retval:Str.split("-",2)){
System.out.println(retval);
}
System.out.println("");
System.out.println("Return Value :");
for(String retval:Str.split("-",3)){
System.out.println(retval);
}
System.out.println("");
System.out.println("Return Value :");
for(String retval:Str.split("-",0)){
System.out.println(retval);
}
System.out.println("");
System.out.println("Return Value :");
for(String retval:Str.split("-")){
System.out.println(retval);
}
}
}
```

This produces the following result:

```
Return Value :
Welcome
to-Tutorialspoint.com
Return Value :
Welcome
to
Tutorialspoint.com
Return Value:
Welcome
```

```
to
Tutorialspoint.com
Return Value :
Welcome
to
Tutorialspoint.com
```


boolean startsWith(String prefix)

Description:

This method has two variants and tests if a string starts with the specified prefix beginning a specified index or by default at the beginning.

Syntax:

Here is the syntax of this method:

```
public boolean startsWith(String prefix,int toffset)
or
public boolean startsWith(String prefix)
```

Parameters:

Here is the detail of parameters:

- **prefix** -- the prefix to be matched.
- **toffset** -- where to begin looking in the string.

Return Value:

- It returns true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.startsWith("Welcome"));
System.out.print("Return Value :");
System.out.println(Str.startsWith("Tutorials"));
System.out.print("Return Value :");
System.out.println(Str.startsWith("Tutorials",11));
}
}
```

This produces the following result:

```
Return Value :true
Return Value :false
```



```
Return Value :true
```

boolean startsWith(String prefix, int toffset)

Description:

This method has two variants and tests if a string starts with the specified prefix beginning a specified index or by default at the beginning.

Syntax:

Here is the syntax of this method:

```
public boolean startsWith(String prefix,int toffset)
or
```

```
public boolean startsWith(String prefix)
```

Parameters:

Here is the detail of parameters:

- **prefix** -- the prefix to be matched.
- **toffset** -- where to begin looking in the string.

Return Value:

- It returns true if the character sequence represented by the argument is a prefix of the character sequence represented by this string; false otherwise.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.startsWith("Welcome"));
System.out.print("Return Value :");
System.out.println(Str.startsWith("Tutorials"));
System.out.print("Return Value :");
System.out.println(Str.startsWith("Tutorials",11));
}
}
```

This produces the following result:


```
Return Value :true
Return Value :false
Return Value :true
```

CharSequence subSequence(int beginIndex, int endIndex)

Description:

This method returns a new character sequence that is a subsequence of this sequence.

Syntax:

Here is the syntax of this method:

```
public CharSequence subSequence(int beginIndex,int endIndex)
```

Parameters:

Here is the detail of parameters:

- **beginIndex** -- the begin index, inclusive.
- **endIndex** -- the end index, exclusive.

Return Value:

- This method returns the specified subsequence.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.subSequence(0,10));
System.out.print("Return Value :");
System.out.println(Str.subSequence(10,15));
```

```
}
}
```
This produces the following result:
```
Return Value :Welcome to
Return Value : Tuto
```
TUTORIALS POINT Simply Easy Learning

String substring(int beginIndex)

Description:

This method has two variants and returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string or up to endIndex - 1 if second argument is given.

Syntax:

Here is the syntax of this method:
```
public String substring(int beginIndex)
or
public String substring(int beginIndex,int endIndex)
```
Parameters:

Here is the detail of parameters:

- **beginIndex** -- the begin index, inclusive.
- **endIndex** -- the end index, exclusive.

Return Value:

- The specified substring.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.substring(10));
System.out.print("Return Value :");
System.out.println(Str.substring(10,15));
}
}
```
This produces the following result:
```
Return Value : Tutorialspoint.com
Return Value : Tuto
```
TUTORIALS POINT Simply Easy Learning

String substring(int beginIndex, int endIndex)

Description:

This method has two variants and returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string or up to endIndex - 1 if second argument is given.

Syntax:

Here is the syntax of this method:
```
public String substring(int beginIndex)
or
```

```
public String substring(int beginIndex,int endIndex)
```

Parameters:

Here is the detail of parameters:

- **beginIndex** -- the begin index, inclusive.
- **endIndex** -- the end index, exclusive.

Return Value:

- The specified substring.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.substring(10));
System.out.print("Return Value :");
System.out.println(Str.substring(10,15));
}
}
```

This produces the following result:

```
Return Value : Tutorialspoint.com
Return Value : Tuto
```


char[] toCharArray()

Description:

This method converts this string to a new character array.

Syntax:

Here is the syntax of this method:

```
public char[] toCharArray()
```

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- It returns a newly allocated character array, whose length is the length of this string and whose contents are initialized to contain the character sequence represented by this string.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.toCharArray());
}
}
```

This produces the following result:

```
Return Value :Welcome to Tutorialspoint.com
```

String toLowerCase()

Description:

This method has two variants. First variant converts all of the characters in this String to lower case using the rules of the given Locale. This is equivalent to calling toLowerCase(Locale.getDefault()).
Second variant takes locale as an argument to be used while converting into lower case.

Syntax:

Here is the syntax of this method:

```
public String toLowerCase()
or
public String toLowerCase(Locale locale)
```

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- It returns the String, converted to lowercase.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.toLowerCase());
}
}
```

This produces the following result:

```
Return Value :welcome to tutorialspoint.com
```

String toLowerCase(Locale locale)

Description:

This method has two variants. First variant converts all of the characters in this String to lower case using the rules of the given Locale. This is equivalent to calling toLowerCase(Locale.getDefault()).
Second variant takes locale as an argument to be used while converting into lower case.

Syntax:

Here is the syntax of this method:

```
public String toLowerCase()
```



```
or
public String toLowerCase(Locale locale)
```

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- It returns the String, converted to lowercase.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.toLowerCase());
```

```
}
}
```
This produces the following result:
```
Return Value :welcome to tutorialspoint.com
```
String toString()
Description:
This method returns itself a string
Syntax:
Here is the syntax of this method:
```
public String toString()
```
Parameters:
Here is the detail of parameters:
- **NA**

Return Value:
- This method returns the string itself.

Example:
```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.toString());
}
}
```
This produces the following result:
```
Return Value :Welcome to Tutorialspoint.com
```
String toUpperCase()
Description:
This method has two variants. First variant converts all of the characters in this String to upper case using the rules of the given Locale. This is equivalent to calling toUpperCase(Locale.getDefault()).
Second variant takes locale as an argument to be used while converting into upper case.
Syntax:
Here is the syntax of this method:
```
public String toUpperCase()
or
public String toUpperCase(Locale locale)
```
Parameters:
Here is the detail of parameters:
- **NA**

Return Value:
- It returns the String, converted to uppercase.

Example:
```
import java.io.*;
```

```
public class Test{
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.toUpperCase());
}
}
```

This produces the following result:

```
Return Value :WELCOME TO TUTORIALSPOINT.COM
```

String toUpperCase(Locale locale)

Description:

This method has two variants. First variant converts all of the characters in this String to upper case using the rules of the given Locale. This is equivalent to calling toUpperCase(Locale.getDefault()).
Second variant takes locale as an argument to be used while converting into upper case.

Syntax:

Here is the syntax of this method:

```
public String toUpperCase()
or
public String toUpperCase(Locale locale)
```

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- It returns the String, converted to uppercase.

Example:

```
import java.io.*;
public class Test{
```



```
public static void main(String args[]){
String Str=new String("Welcome to Tutorialspoint.com");
System.out.print("Return Value :");
System.out.println(Str.toUpperCase());
}
}
```

This produces the following result:

```
Return Value :WELCOME TO TUTORIALSPOINT.COM
```

String trim()

Description:

This method returns a copy of the string, with leading and trailing whitespace omitted.

Syntax:

Here is the syntax of this method:

```
publicString trim()
```

Parameters:

Here is the detail of parameters:

- **NA**

Return Value:

- It returns a copy of this string with leading and trailing white space removed, or this string if it has no

leading or trailing white space.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
String Str=new String(" Welcome to Tutorialspoint.com ");
System.out.print("Return Value :");
System.out.println(Str.trim());
}
}
```

This produces the following result:

```
Return Value :Welcome to Tutorialspoint.com
```


static String valueOf(primitive data type x)

Description:

This method has followings variants, which depend on the passed parameters. This method returns the string representation of the passed argument.

- **valueOf(boolean b):** Returns the string representation of the boolean argument.
- **valueOf(char c):** Returns the string representation of the char argument.
- **valueOf(char[] data):** Returns the string representation of the char array argument.
- **valueOf(char[] data, int offset, int count):** Returns the string representation of a specific subarray of the char array argument.
- **valueOf(double d):** Returns the string representation of the double argument.
- **valueOf(float f):** Returns the string representation of the float argument.
- **valueOf(int i):** Returns the string representation of the int argument.
- **valueOf(long l):** Returns the string representation of the long argument.
- **valueOf(Object obj):** Returns the string representation of the Object argument.

Syntax:

Here is the syntax of this method:

```
static String valueOf(boolean b)
or
static String valueOf(char c)
or
static String valueOf(char[] data)
or
static String valueOf(char[] data,int offset,int count)
or
static String valueOf(double d)
or
static String valueOf(float f)
or
static String valueOf(int i)
or
static String valueOf(long l)
or
```



```
static String valueOf(Object obj)
```

Parameters:

Here is the detail of parameters:

- **See the description.**

Return Value :

 This method returns the string representation.

Example:

```
import java.io.*;
public class Test{
public static void main(String args[]){
double d =102939939.939;
boolean b =true;
long l =1232874;
char[] arr ={'a','b','c','d','e','f','g'};
System.out.println("Return Value : "+String.valueOf(d));
System.out.println("Return Value : "+String.valueOf(b));
System.out.println("Return Value : "+String.valueOf(l));
System.out.println("Return Value : "+String.valueOf(arr));
}
}
```

This produces the following result:

```
Return Value : 1.02939939939E8
Return Value : true
Return Value : 1232874
Return Value : abcdefg
```

CHP-14
Java Arrays

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar;// preferred way.
or
dataType arrayRefVar[];// works but not preferred way.
```

Note: The style **dataType[] arrayRefVar** is preferred. The style **dataType arrayRefVar[]** comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

Example:

The following code snippets are examples of this syntax:

```
double[] myList;// preferred way.
or
double myList[];// works but not preferred way.
```

Creating Arrays:

You can create an array by using the new operator with the following syntax:

```
arrayRefVar =new dataType[arraySize];
```

The above statement does two things:

CHAPTER

- It creates an array using new dataType[arraySize];
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

```
dataType[] arrayRefVar =new dataType[arraySize];
```

Alternatively you can create arrays as follows:

```
dataType[] arrayRefVar ={value0, value1,..., valuek};
```

The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0

to **arrayRefVar.length-1**.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

```
double[] myList =new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.

Processing Arrays:

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Example:

Here is a complete example of showing how to create, initialize and process arrays:

```
public class TestArray{
public static void main(String[] args){
double[] myList ={1.9,2.9,3.4,3.5};
// Print all the array elements
for(int i =0; i < myList.length; i++){
```



```
System.out.println(myList[i]+" ");
}
// Summing all elements
double total =0;
for(int i =0; i < myList.length; i++){
total += myList[i];
}
System.out.println("Total is "+ total);
// Finding the largest element
double max = myList[0];
for(int i =1; i < myList.length; i++){
if(myList[i]> max) max = myList[i];
}
System.out.println("Max is "+ max);
}
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
Totalis11.7
Maxis3.5
```

The foreach Loops:

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

Example:

The following code displays all the elements in the array myList:

```
public class TestArray{
public static void main(String[] args){
double[] myList ={1.9,2.9,3.4,3.5};
// Print all the array elements
for(double element: myList){
System.out.println(element);
}
}
}
```

This would produce the following result:

```
1.9
2.9
```

3.4
3.5

Passing Arrays to Methods:

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array:


```
public static void printArray(int[] array){
for(int i =0; i < array.length; i++){
System.out.print(array[i]+" ");
}
}
```

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2:

```
printArray(newint[]{3,1,2,6,4,2});
```

Returning an Array from a Method:

A method may also return an array. For example, the method shown below returns an array that is the reversal of another array:

```
publicstaticint[] reverse(int[] list){
int[] result =newint[list.length];
for(int i =0, j = result.length -1; i < list.length; i++, j--){
result[j]= list[i];
}
return result;
}
```

The Arrays Class:

The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements. These methods are overloaded for all primitive types.

SN Methods with Description

1

public static int binarySearch(Object[] a, Object key)

Searches the specified array of Object (Byte, Int , double, etc.) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, (-(insertion point + 1).

2

public static boolean equals(long[] a, long[] a2)

Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. This returns true if the two arrays are equal. Same method could be used by all other primitive data types (Byte, short, Int, etc.)

3

public static void fill(int[] a, int val)

Assigns the specified int value to each element of the specified array of ints. Same method could be used by all other primitive data types (Byte, short, Int, etc.)

4

public static void sort(Object[] a)

Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. Same method could be used by all other primitive data types (Byte, short, Int, etc.)