Java means DURGA SOFT..

CORE JAVA

Material

Java™

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

String manipulations

1) ***Java.lang.String***
2) ***Java.lang.StringBuffer***
3) ***Java.lang.StringBuilder***
4) ***Java.util.StringTokenizer***

Java.lang.String:-

String is used to represent group of characters or character array enclosed with in the double quotes.

```
class Test
{       public static void main(String[] args)
        {       String str="ratan";
                System.out.println(str);

                String str1=new String("ratan");
                System.out.println(str1);

                char[] ch={'r','a','t','a','n'};
                String str3=new String(ch);
                System.out.println(str3);

                char[] ch1={'a','r','a','t','a','n','a'};
                String str4=new String(ch1,1,5);
                System.out.println(str4);

                byte[] b={65,66,67,68,69,70};
                String str5=new String(b);
                System.out.println(str5);

                byte[] b1={65,66,67,68,69,70};
                String str6=new String(b1,2,4);
                System.out.println(str6);
        }
}
```

Case 1:-String vsStringBuffe

String &StringBuffer both classes are final classes present in java.lang package.

Case 2:-String vsStringBuffer

We are able to create String object in two ways.

1) ***Without using new operator*** ***String str="ratan";***
2) ***By using new operator*** ***String str = new String("ratan");***

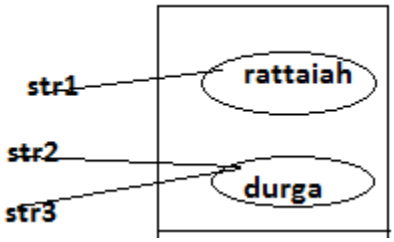
We are able to create StringBuffer object only one approach by using new operator.

StringBuffersb = new StringBuffer("sravyainfotech");

Creating a string object without using new operator :-

- When we create String object without using new operator the objects are created in SCP (String constant pool) area.

- String str1="rattaiah";
  String str2="Sravya";
  String str3="Sravya";



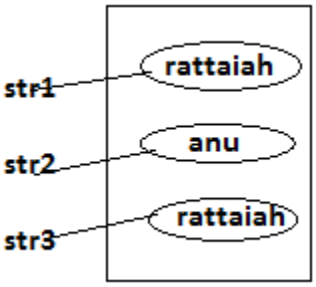
SCP area

- When we create object in SCP area then just before object creation it is always checking previous objects.
  - If the previous object is available with the same content then it won't create new object that reference variable pointing to existing object.
  - If the previous objects are not available then JVM will create new object.
- SCP area does not allow duplicate objects.

Creating a string object by using new operator

- Whenever we are creating String object by using new operator the object created in heap area.

- String str1=new String("rattaiah");
  String str2 = new String("anu");
  String str3 = new String("rattaiah");



Heap area

- When we create object in Heap area instead of checking previous objects it directly creates objects.

- Heap memory allows duplicate objects.

Example:-

```
class Test
{       public static void main(String[] args)
        {       //two approaches to create a String object
                String str1 = "ratan";
                System.out.println(str1);
                String str2 = new String("anu");
                System.out.println(str2);

                //one approach to create StringBuffer Object (by using new operator)
                StringBuffersb = new StringBuffer("ratansoft");
                System.out.println(sb);
        }
}
```

==*operator :-*

- *It is comparing reference type and it returns Boolean value as a return value.*
- *If two reference variables are pointing to same object then it returns true otherwise false.*

Example:-

```
class Test
{       public static void main(String[] args)
        {       Test t1 = new Test();
                Test t2 = new Test();
                Test t3 = t1;
                System.out.println(t1==t2);       //false
                System.out.println(t1==t3);       //true

                String str1="ratan";
                String str2="ratan";
                System.out.println(str1==str2);  //true

                String s1 = new String("anu");
                String s2 = new String("anu");
                System.out.println(s1==s2);       //false

                StringBuffer sb1 = new StringBuffer("sravya");
                StringBuffer sb2 = new StringBuffer("sravya");
                System.out.println(sb1==sb2);    //flase
        }
}
```

Case 3:- String

Java.lang.Stringvsjava.lang.StringBuffer:-

*String is **immutability** class it means once we are creating String objects it is not possible to perform modifications on existing object. (String object is fixed object)*

*StringBuffer is a **mutability** class it means once we are creating StringBuffer objects on that existing object it is possible to perform modification.*

Example :-

```
class Test
{       public static void main(String[] args)
        {       //immutability class (modifications on existing content not allowed)
                String str="ratan";
                str.concat("soft");
                System.out.println(str);

                //mutability class (modifications on existing content possible)
                StringBuffersb = new StringBuffer("anu");
                sb.append("soft");
                System.out.println(sb);
        }
}
```

Concat( ) :-

- *Concat() method is combining two String objects and it is returning new String object.*

  publicjava.lang.Stringconcat(java.lang.String);

Example :-

```
class Test
{       public static void main(String[] args)
        {       String str="ratan";
                String str1 = str.concat("soft");//concat() method return String object.
                System.out.println(str);
                System.out.println(str1);
        }
}
```


- One java class method is able to return same class object or different class object that method is called factory method.

- In java if the method is returning some class object that method is called factory method.
- There are two types of factory methods in java
  - Instance factory method
  - Static factory method

Instance factory method:-

- Concat() is factory method because it is present in String class and able to return String class object only.

```
String str="ratan";
String str1 = str.concat("soft");
System.out.println(str1);
```

- toString() is factory method because StringBuffer class toString() method is returning String class object.

```
StringBuffersb = new StringBuffer("anu");
String sss = sb.toString();
```

Static factory method:-

- if the factory method is called by using class name that method is called static factory method.

```
Integer i = Integer.valueOf(100);
System.out.println(i);
```

Example :-

```
class Test
{       public static void main(String[] args)
        {       //instance factory method   [fctory method is called ]
                String str="ratan";
                String str1 = str.concat("soft");
                System.out.println(str1);
                StringBuffersb = new StringBuffer("anu");
                String sss = sb.toString();

                //static factory method
                Integer i = Integer.valueOf(100);
                System.out.println(i);
        }
}
```


Java.lang.Stringvsjava.lang.StringBuffer:-

Internal implementation equals() method:-

- equals() method present in object used for reference comparison & return Boolean value.
  - If two reference variables are pointing to same object returns true otherwise false.
- String is child class of object and it is overriding equals( ) methods used for content comparison.
  - If two objects content is same then returns true otherwise false.
- StringBuffer class is child class of object and it is not overriding equals() method hence it is using parent class(Object) equals() method used for reference comparison.
  - If two reference variables are pointing to same object returns true otherwise false.

```
class Object
{   publicboolean equals(java.lang.Object)
        {       // reference comparison;
        }
};
class String extends Object
{       //String class is overriding equals() method
        publicboolean equals(java.lang.Object);
        {       //content comparison;
        }
};
classStringBuffer extends Object
{       //not overriding hence it is using parent class equals() method
        //reference comparison;
};
```

Example :-

```
class  Test
{       Test(String str)  {    }
        public static void main(String[] args)
        {       Test t1 = new Test("ratan");
                Test t2 = new Test("ratan");
                //Object class equals() method executed (reference comparison)
                System.out.println(t1.equals(t2));

                String str1 = new String("Sravya");
                String str2 = new String("Sravya");
                //String class equals() method executed (content comparison)
                System.out.println(str1.equals(str2));

                StringBuffer sb1 = new StringBuffer("anu");
                StringBuffer sb2 = new StringBuffer("anu");
                //StringBuffer class equals() executed (reference comparison)
                System.out.println(sb1.equals(sb2));
        }
}
```

Java.lang.Stringvsjava.lang.StringBuffer:-

Internal implementation of toString method:-

- toString( ) method Returns a string representation of the object and it is present in java.lang.Object class.
- String is child class of Object and String is overriding toString() used to return content of the String.
- StringBuffer is child class of Object and StringBuffer is overriding toString() used to return content of the StringBuffer.

Note :- whenever we are printing reference variable internally it is calling toString() method

In java when we print any type of reference variables internally it calls toString() method.

```
class Object
{       publicjava.lang.StringtoString()
        {       returngetClass().getName() + '@' + Integer.toHexString(hashCode());     }
}
class String extends Object
{       //overriding method
        publicjava.lang.StringtoString()
        {       return "content of String";       }
};
classStringBuffer extends Object
{       //overriding method
        publicjava.lang.StringtoString()
        {       return "content of String";       }
};
```

Example:-

```
class Test
{       public static void main(String[] args)
        {       Test t = new Test();
//the below two lines are same (if we are printing reference variables it's calling toString() method)
                System.out.println(t);              //object class toString() executed
                System.out.println(t.toString());//object class toString() executed

                String str="ratan";
```

```
                System.out.println(str);  //String  class toString() executed
                System.out.println(str.toString());//String  class toString() executed

                StringBuffersb = new StringBuffer("anu");
                System.out.println(sb);                 //StringBuffer class toString() executed
                System.out.println(sb.toString());      //StringBuffer class toString() executed
        }
};
```

D:\>java Test
Test@530daaTest@530daaRatan ratanAnu anu

In above example when we call **t.toString()** JVM searching toString() in Test class since not there then parent class(Object) toString() method is executed.

== operatorvs equals() :-

- In above example we are completed equals() method.
- == operator used to check reference variables & returns boolean ,if two reference variables are pointing to same object returns true otherwise false.

```
class  Test
{       Test(String str){}
        public static void main(String[] args)
        {       Test t1 = new Test("ratan");
                Test t2 = new Test("ratan");
                System.out.println(t1==t2);//reference comparison    false
                System.out.println(t1.equals(t2));//reference comparison    false

                String str1="anu";
                String str2="anu";
                System.out.println(str1==str2);  //reference comparison    true
                System.out.println(str1.equals(str2));//content comparison    true

                String str3 = new String("Sravya");
                String str4 = new String("Sravya");
                System.out.println(str3==str4);         //reference comparison   false
                System.out.println(str3.equals(str4));  //content comparison    true

                StringBuffer sb1 = new StringBuffer("students");
                StringBuffer sb2 = new StringBuffer("students");
                System.out.println(sb1==sb2);           //reference comparison    false
                System.out.println(sb1.equals(sb2));    //reference comparison    false
        }
}


class Test extends Object
{       Test(String str){}
        public static void main(String[] args)
```

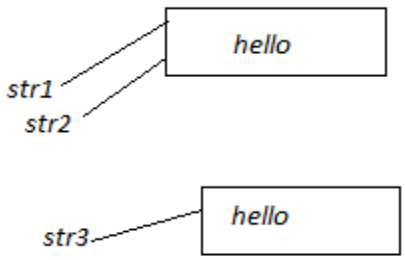
```
{       Test t1 = new Test("ratan");
        Test t2 = new Test("anu");
        Test t3 = t2;
        Test t4 = new Test("ratan");
        System.out.println(t1==t2);//false
        System.out.println(t1==t3);//false
        System.out.println(t3==t2);//true
        System.out.println(t1==t4);//false
        //object class equals() executed reference comparison
        System.out.println(t1.equals(t2));//false
        System.out.println(t3.equals(t2));//true

        String str1 = "ratan";
        String str2="ratan";
        String str3=str2;
        System.out.println(str1==str2);//true
        System.out.println(str3==str2);//true
        System.out.println(str1==str3);//true
        //String class equals() executed content comparison
        System.out.println(str1.equals(str2));//true

        String s1= new String("ratan");
        String s2= new String("ratan");
        String s3=s2;
        System.out.println(s1==s2);//false
        System.out.println(s2==s3);//true
        //String class equals() executed content comparison
        System.out.println(s1.equals(s2));//true

        StringBuffer sb1 = new StringBuffer("anu");
        StringBuffer sb2 = new StringBuffer("anu");
        StringBuffer sb3 = sb1;
        System.out.println(sb1==sb2);//false
        System.out.println(sb1==sb3);//true
        //StringBuffer class equals() executed reference comparison
        System.out.println(sb1.equals(sb3));//true
    }
}
```

Example :- String identity vs String equality



```
class Test
{       public static void main(String[] args)
        {       String str1 = "hello";
                String str2 = "hello";
                String str3= new String("hello");
                System.out.println(str1==str2);          //true
                System.out.println(str1==str3);          //false
                System.out.println(str1==str3);          //false
                System.out.println(str1.equals(str2));   //true
                System.out.println(str1.equals(str3));   //true
                System.out.println(str2.equals(str3));   //true
        }
}
```


Java.lang.String class methods:-

1) CompareTo() &compareToIgnoreCase():-

- By using compareTo() we are comparing two strings character by character, such type of checking is called lexicographically checking or dictionary checking.
- compareTo() is return type is integer and it returns three values
  a. zero ----> if both String are equal
  b. positive --->if first string first character Unicode value is bigger than second String first character Unicode value then it returns positive.
  c. Negative ---> if first string first character Unicode value is smaller than second string first character Unicode value then it returns negative.
- compareTo() method comparing two string with case sensitive.
- compareToIgnoreCase() method comparing two strings character by character by ignoring case.

```
class Test
{       public static void main(String... ratan)
        {       String str1="ratan";
                String str2="Sravya";
                String str3="ratan";
                System.out.println(str1.compareTo(str2));//14
```

```
                System.out.println(str1.compareTo(str3));//0
                System.out.println(str2.compareTo(str1));//-13
                System.out.println("ratan".compareTo("RATAN"));//+ve
                System.out.println("ratan".compareToIgnoreCase("RATAN"));//0
        }
};
```

Difference between length() method and length variable:-

- ***length*** *variable used to find length of the Array.*
- ***length()*** *is method used to find length of the String.*

Example :-

```
                int [] a={10,20,30};
                System.out.println(a.length);     //3

                String str="rattaiah";
                System.out.println(str.length());          //8
```

cahrAt(int) & split() & trim():-

charAt(int):-By using above method we are able to extract the character from particular index position.

public char charAt(int);

Split(String):- By using split() method we are dividing string into number of tokens.

publicjava.lang.String[] split(java.lang.String);

trim():- trim() is used to remove the trail and leading spacesthis method always used for memory saver.

publicjava.lang.String trim();

```
class Test
{       public static void main(String[] args)
        {       //cahrAt() method
                String str="ratan";
                System.out.println(str.charAt(1));
                //System.out.println(str.charAt(10));StringIndexOutOfBoundsException
                charch="ratan".charAt(2);
                System.out.println(ch);
                //split() method
                String s="hi rattaiah how r u";
                String[] str1=s.split(" ");
                for(String str2 : str1)
                {       System.out.println(str2);
                }
                //trim()
                String ss="       ratan    ";
                System.out.println(ss.length());//7
                System.out.println(ss.trim());//ratan
                System.out.println(ss.trim().length());//5
        }
}
```

replace() &toUpperCase() &toLowerCase():-

publicjava.lang.String replace(Stirngstr,Stringstr1):-

publicjava.lang.String replace(char, char);

replace() method used to replace the String or character.

publicjava.lang.StringtoLowerCase();

publicjava.lang.StringtoUpperCase();

The above methods are used to convert lower case to upper case & upper case to lower case.

Example:-

```
class Test
{       public static void main(String[] args)
        {       String str="rattaiah how r u";
                System.out.println(str.replace('a','A'));          //rAttAiAh
                System.out.println(str.replace("how","who"));      //rattaiah how r u

                String str1="Sravya software solutions";
                System.out.println(str1);
                System.out.println(str1.replace("software","hardware"));// Sravya hardware solutions

                String str="ratan HOW R U";
                System.out.println(str.toUpperCase());
                System.out.println(str.toLowerCase());
                System.out.println("RATAN".toLowerCase());
                System.out.println("soft".toUpperCase());
        }
}
```

endsWith()&startsWith()& substring():-

- ***endsWith()*** is used to find out if the string is ending with particular character/string or not.
- ***startsWith()*** used to find out the particular String starting with particular character/string or not.

```
publicbooleanstartsWith(java.lang.String);
publicbooleanendsWith(java.lang.String);
```

- *substring() used to find substring in main String.*

*publicjava.lang.String substring(int);**int = starting index***

*publicjava.lang.String substring(int, int);**int=starting index to int =ending index***

while printing substring() it includes starting index & it excludes ending index.

Example:-

```
class Test
{       public static void main(String[] args)
        {       String str="rattaiah how r u";
                System.out.println(str.endsWith("u"));          //true
                System.out.println(str.endsWith("how"));        //false
                System.out.println(str.startsWith("d"));        //false
                System.out.println(str.startsWith("r"));        //true
                String s="ratan how r u";
                System.out.println(s.substring(2));             //tan how r u
                System.out.println(s.substring(1,7));           //atan h
                System.out.println("ratansoft".substring(2,5)); //tan
        }
}
```

StringBuffer class methods:-

reverse():-

```
class Test
{       public static void main(String[] args)
        {       StringBuffersb=new StringBuffer("rattaiah");
                System.out.println(sb);
                System.out.println(sb.delete(1,3));
                System.out.println(sb);
                System.out.println(sb.deleteCharAt(1));
                System.out.println(sb.reverse());
        }
}
```

Append():-

By using this method we can append the any values at the end of the string

Ex:-

```
class Test
{       public static void main(String[] args)
        {       StringBuffersb=new StringBuffer("rattaiah");
                String str=" salary ";
                int a=60000;
                sb.append(str);
                sb.append(a);
                System.out.println(sb);
        }
};
```

Insert():-

By using above method we are able to insert the string any location of the existing string.

```
class Test
{       public static void main(String[] args)
        {       StringBuffersb=new StringBuffer("ratan");
                sb.insert(0,"hi ");
                System.out.println(sb);
        }
}
```

indexOf() and lastIndexOf():-

Ex:-

```
class Test
{       public static void main(String[] args)
        {       StringBuffersb=new StringBuffer("hi ratan hi");
                int i;
                i=sb.indexOf("hi");
                System.out.println(i);
                i=sb.lastIndexOf("hi");
                System.out.println(i);
        }
}
```

replace():-

```
class Test
{       public static void main(String[] args)
        {       StringBuffersb=new StringBuffer("hi ratan hi");
                sb.replace(0,2,"oy");
                System.out.println("after replaceing the string:-"+sb);
        }
}
```

Java.lang.StringBuilder:-

1) Introduced in jdk1.5 version.
2) StringBuilder is identical to StringBuffer except for one important difference.
3) Every method present in the StringBuilder is not Synchronized means that is not thread safe.
4) multiple threads are allow to operate on StringBuilder methods hence the performance of the application is increased.

Cloneable:-

1) The process of creating exactly duplicate object is called cloning.
2) We can create a duplicate object only for the cloneableclasses .
3) We can create cloned object by using clone()
4) The main purpose of the cloning is to maintain backup.

```
class Test implements Cloneable
{       int a=10,b=20;
        public static void main(String[] args) throws CloneNotSupportedException
        {       Test t1 = new Test();//creates object of Test class
                Test t2 = (Test)t1.clone();//duplicate object of Test class
                System.out.println(t1.a);
                System.out.println(t1.b);
                t1.b=555;
                t1.a=444;
                System.out.println(t1.a);
                t1.b=333;
                System.out.println(t1.a);
                System.out.println(t1.b);
                //if we want initial values use duplicate object
                System.out.println(t2.a);//10
                System.out.println(t2.b);//20
        }
}
```

```
importjava.util.*;
class Test
{
        public static void main(String[] args)
        {
```

```
		String str="hi ratan w r u wt bout anushka";
StringTokenizerst = new StringTokenizer(str);//split the string with by default (space symbol)
		while (st.hasMoreElements())
		{
			System.out.println(st.nextElement());
		}

		//used our string to split giver String
		String str1 = "hi,rata,mf,sdfsdf,ara";  StringTokenizer st1 = new
StringTokenizer(str1,",");
		while (st1.hasMoreElements())
		{
			System.out.println(st1.nextElement());
		}


	}
}
```