STRING IN JAVA

JAVA STRING

in Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

char[] ch={'j','a','v','a'};

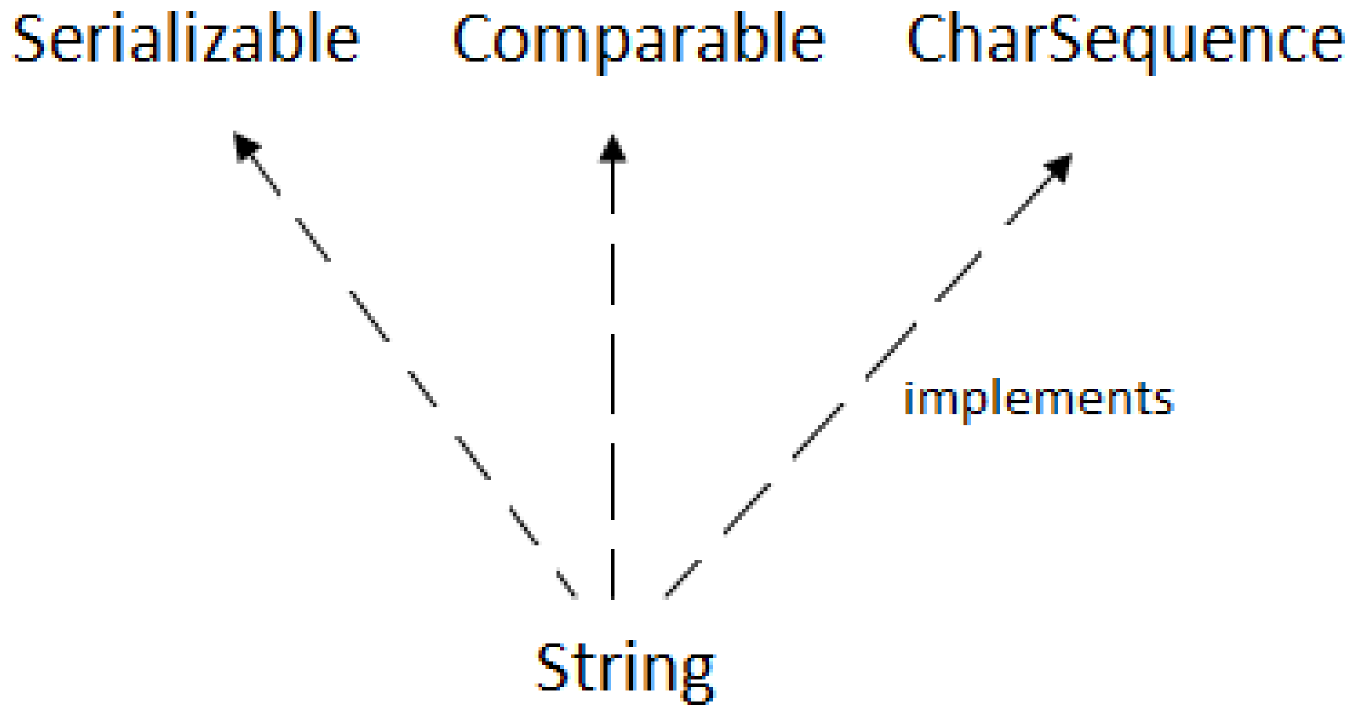
String s=**new** String(ch);

is same as:

String s="java";

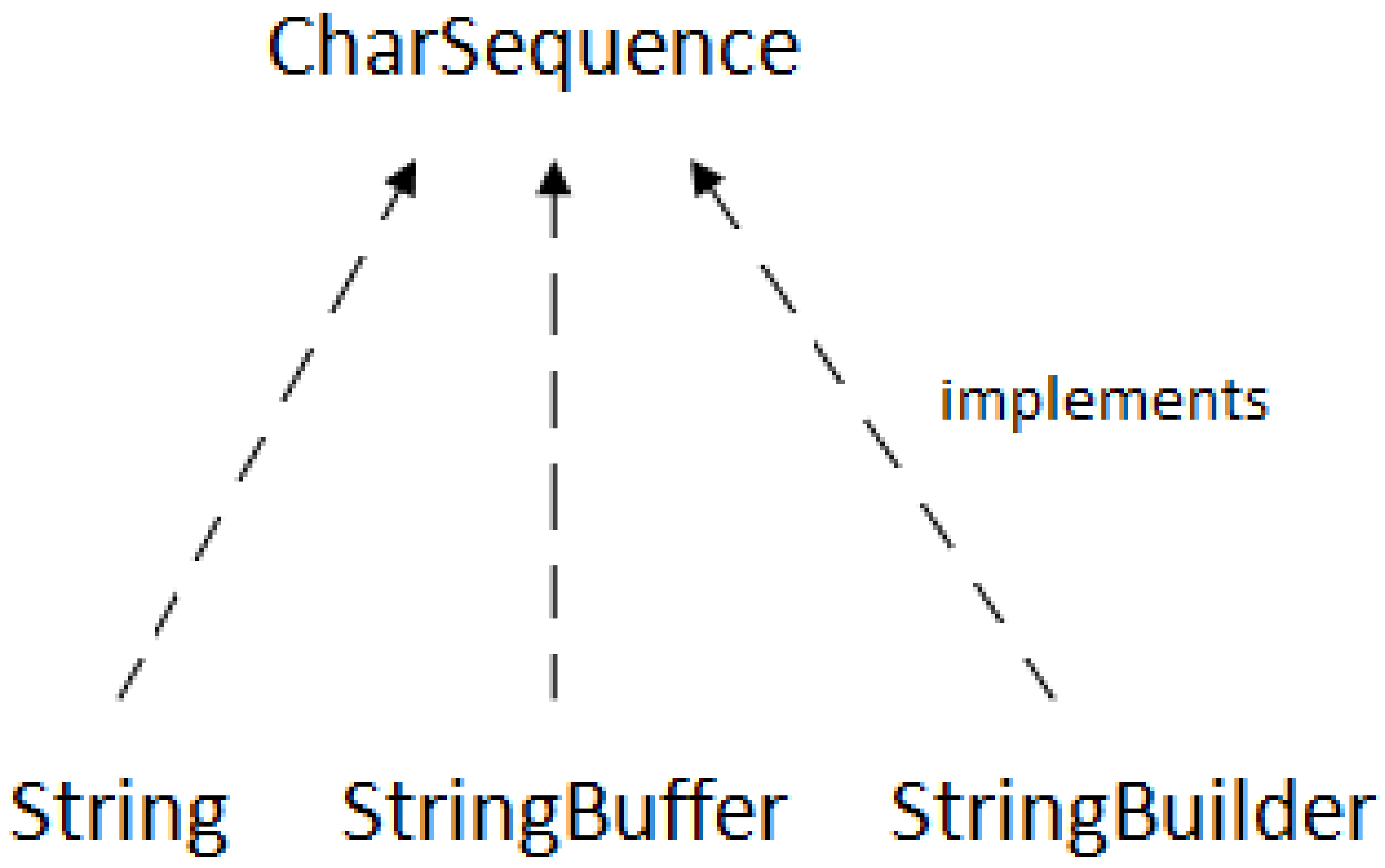
Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.


Serializable
Comparable
CharSequence
implements
String

CHARSEQUENCE INTERFACE

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.


CharSequence
implements
String
StringBuffer
StringBuilder

The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

We will discuss immutable string later. Let's first understand what is String in Java and how to create the String object.

WHAT IS STRING IN JAVA

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

By string literal

By new keyword

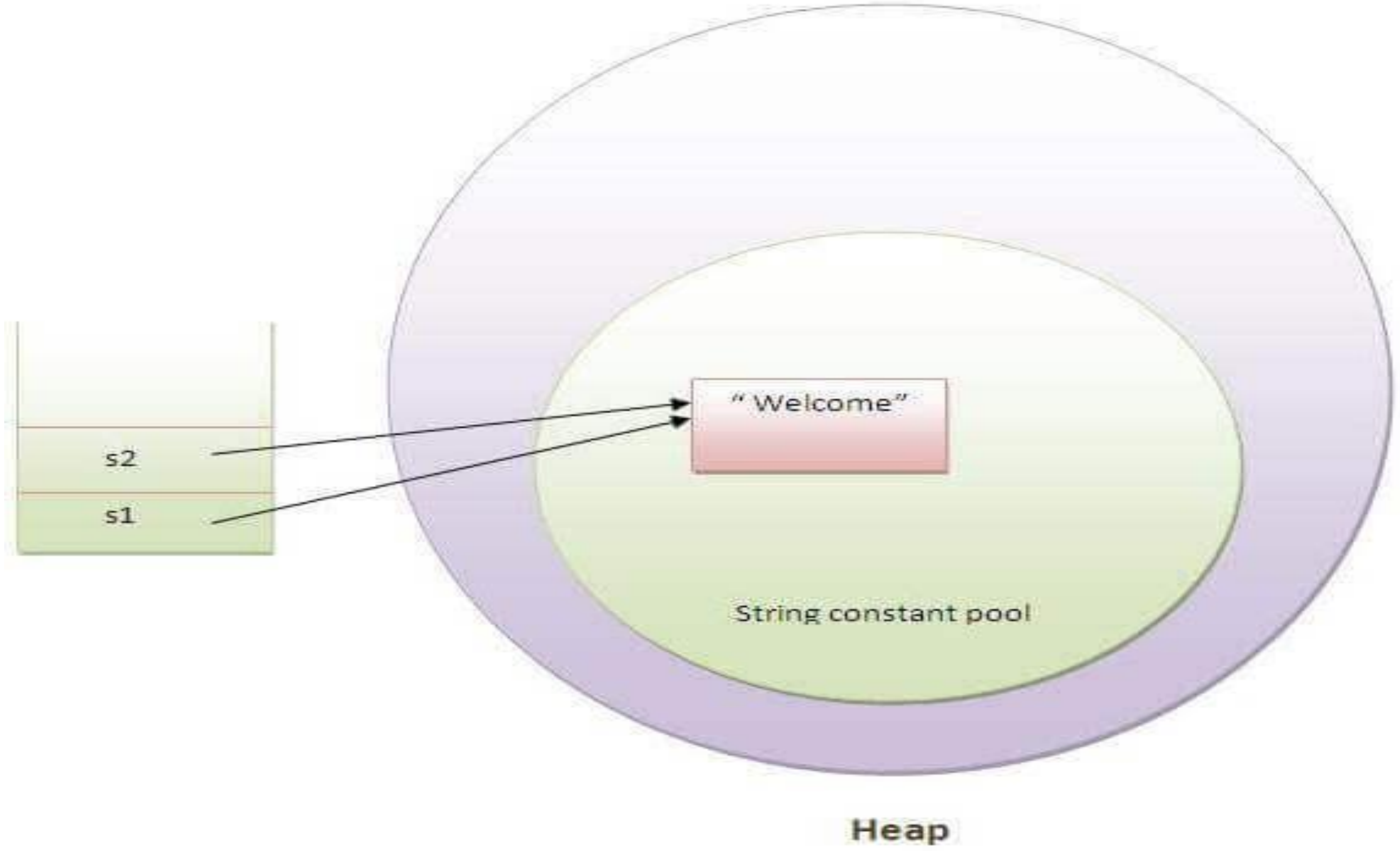
STRING LITERAL

Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

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

```
String s1="Welcome";
String s2="Welcome";//It doesn't create a new instance
```


s2
s1
" Welcome"
String constant pool
Heap

In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as the "string constant pool".

WHY JAVA USES THE CONCEPT OF STRING LITERAL?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

BY NEW KEYWORD

String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

JAVA STRING EXAMPLE

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

IMMUTABLE STRING IN JAVA

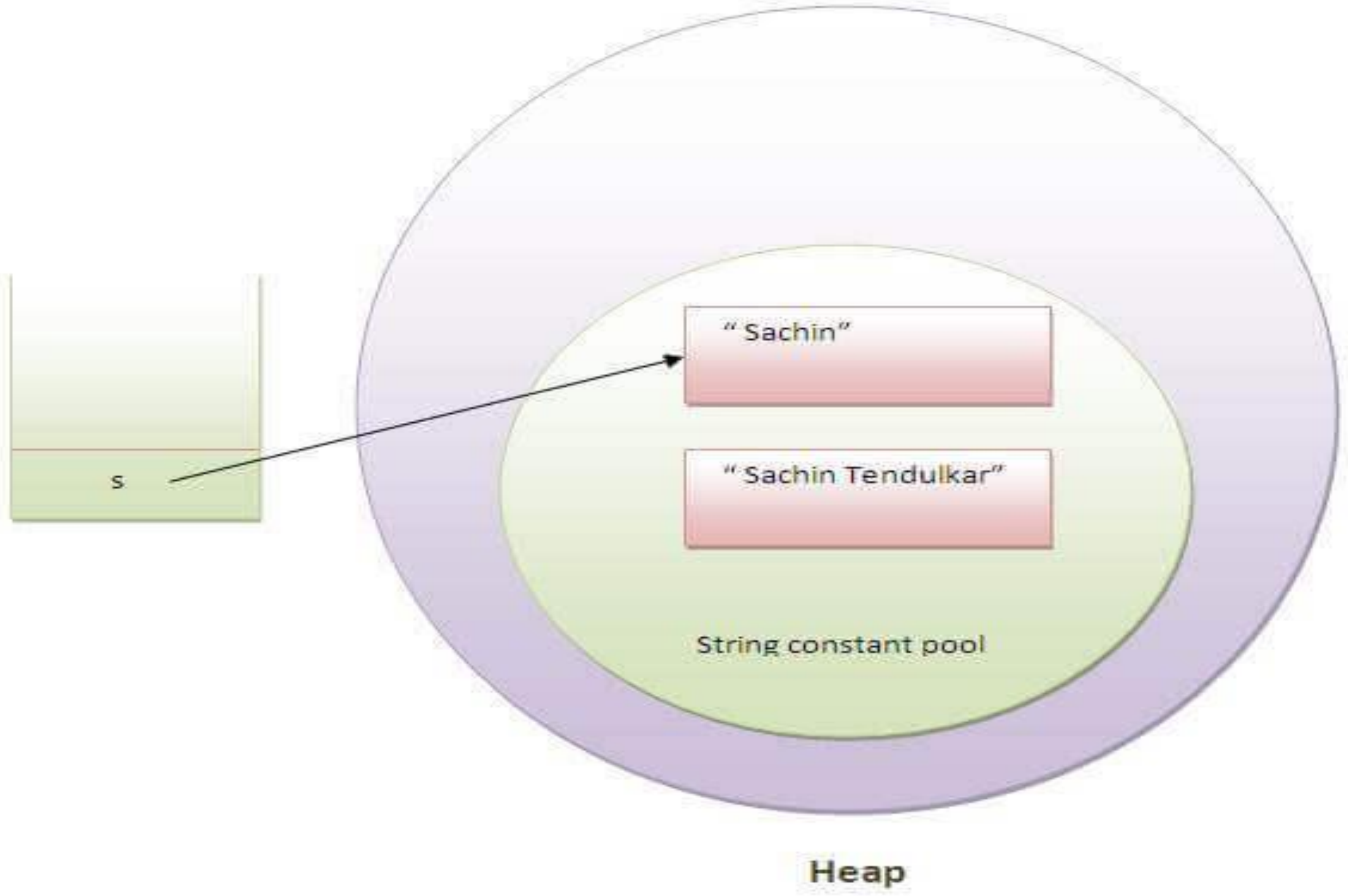
In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objects
 }
}
```

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.


s
" Sachin"
" Sachin Tendulkar"
String constant pool
Heap

As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

WHY STRING OBJECTS ARE IMMUTABLE IN JAVA?

Because java uses the concept of string literal.Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

JAVA STRING COMPARE

We can compare string in java on the basis of content and reference.

We can compare string in java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare string in java:

1. By equals() method
2. By = = operator
3. By compareTo() method

STRING COMPARE BY EQUALS() METHOD

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

public boolean equals(Object another) compares this string to the specified object.

public boolean equalsIgnoreCase(String another) compares this String to another string, ignoring case.

```
class Teststringcomparison1{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 }
}
```

```
class Teststringcomparison2{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";

   System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s2));//true
 }
}
```

STRING COMPARE BY == OPERATOR

The = = operator compares references not values.

```
class Teststringcomparison3{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   System.out.println(s1==s2);//true (because both refer to same instance)
   System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
 }
}
```

STRING COMPARE BY COMPARETO() METHOD

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

s1 == s2 :0

s1 > s2 :positive value

s1 < s2 :negative value

```
class Teststringcomparison4{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3="Ratan";
   System.out.println(s1.compareTo(s2));//0
   System.out.println(s1.compareTo(s3));//1(because s1>s3)
   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
 }
}
```

STRING CONCATENATION IN JAVA

In java, string concatenation forms a new string *that is* the combination of multiple strings. There are two ways to concat string in java:

1. By + (string concatenation) operator
2. By concat() method

STRING CONCATENATION BY + (STRING CONCATENATION) OPERATOR

Java string concatenation operator (+) is used to add strings. For Example:

```
class TestStringConcatenation1{
 public static void main(String args[]){
   String s="Sachin"+" Tendulkar";
   System.out.println(s);//Sachin Tendulkar
 }
}
```

The **Java compiler transforms** above code to this:

String s=(**new** StringBuilder()).append("Sachin").append(" Tendulkar).toString();

In java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also. For Example:

```
class TestStringConcatenation2{
 public static void main(String args[]){
   String s=50+30+"Sachin"+40+40;
   System.out.println(s);//80Sachin4040
 }
}
```

STRING CONCATENATION BY CONCAT() METHOD

The String concat() method concatenates the specified string to the end of current string. Syntax:

public String concat(String another)

Let's see the example of String concat() method.

```
class TestStringConcatenation3{
 public static void main(String args[]){
   String s1="Sachin ";
   String s2="Tendulkar";
   String s3=s1.concat(s2);
   System.out.println(s3);//Sachin Tendulkar
  }
}
```

SUBSTRING IN JAVA

A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.

Note: Index starts from 0.

You can get substring from the given string object by one of the two methods:

public String substring(int startIndex): This method returns new String object containing the substring of the given string from specified startIndex (inclusive).

public String substring(int startIndex, int endIndex): This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

In case of string:

startIndex: inclusive

endIndex: exclusive

Let's understand the startIndex and endIndex by the code given below.

```
String s="hello";
System.out.println(s.substring(0,2));//he
```

In the above substring, 0 points to h but 2 points to e (because end index is exclusive).

EXAMPLE OF JAVA SUBSTRING

```
public class TestSubstring{
 public static void main(String args[]){
   String s="SachinTendulkar";
   System.out.println(s.substring(6));//Tendulkar
   System.out.println(s.substring(0,6));//Sachin
 }
}
```

JAVA STRINGBUFFER CLASS

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

IMPORTANT CONSTRUCTORS OF STRINGBUFFER CLASS

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

WHAT IS MUTABLE STRING

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

STRINGBUFFER APPEND() METHOD

The append() method concatenates the given argument with this string.

```
class StringBufferExample{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

STRINGBUFFER INSERT() METHOD

The insert() method inserts the given string with this string at the given position.

```
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

STRINGBUFFER REPLACE() METHOD

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

STRINGBUFFER DELETE() METHOD

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

STRINGBUFFER REVERSE() METHOD

The reverse() method of StringBuilder class reverses the current string.

```
class StringBufferExample5{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

STRINGBUFFER CAPACITY() METHOD

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
class StringBufferExample6{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
```

STRINGBUFFER ENSURECAPACITY() METHOD

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
class StringBufferExample7{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
```

JAVA STRINGBUILDER CLASS

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

IMPORTANT CONSTRUCTORS OF STRINGBUILDER CLASS

Constructor	Description
StringBuilder()	creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str)	creates a string Builder with the specified string.
StringBuilder(int length)	creates an empty string Builder with the specified capacity as length.

STRINGBUILDER APPEND() METHOD

The StringBuilder append() method concatenates the given argument with this string.

```
class StringBuilderExample{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

STRINGBUILDER INSERT() METHOD

The StringBuilder insert() method inserts the given string with this string at the given position.

```
class StringBuilderExample2{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

STRINGBUILDER REPLACE() METHOD

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBuilderExample3{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

STRINGBUILDER DELETE() METHOD

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

```
class StringBuilderExample4{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

STRINGBUILDER REVERSE() METHOD

The reverse() method of StringBuilder class reverses the current string.

```
class StringBuilderExample5{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

STRINGBUILDER CAPACITY() METHOD

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
class StringBuilderExample6{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}
```

STRINGBUILDER ENSURECAPACITY() METHOD

The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

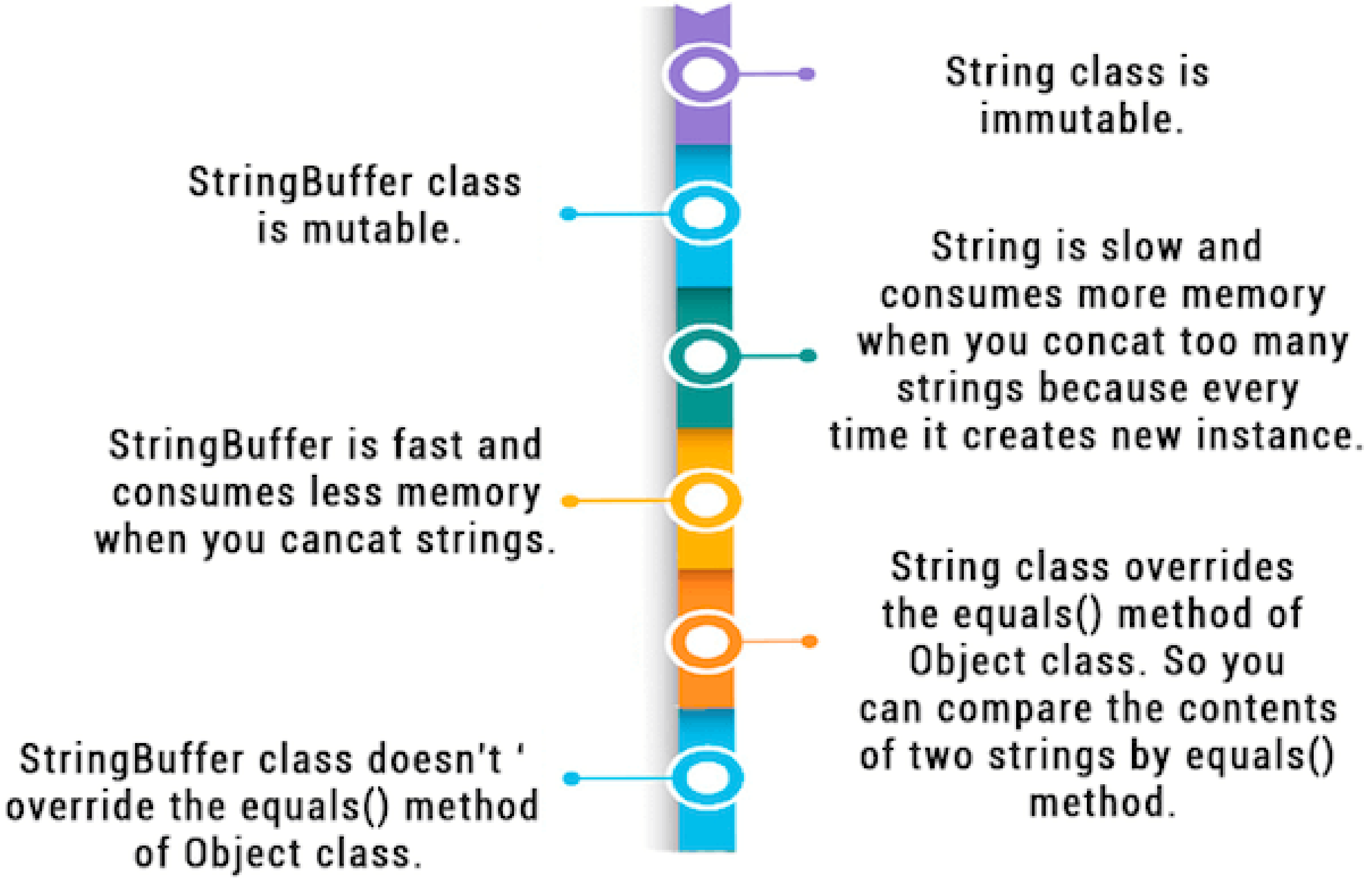
```
class StringBuilderExample7{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

DIFFERENCE BETWEEN STRING AND STRINGBUFFER

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

StringBuffer vs String



String class is immutable.

StringBuffer class is mutable.

String is slow and consumes more memory when you concat too many strings because every time it creates new instance.

StringBuffer is fast and consumes less memory when you cancat strings.

String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.

StringBuffer class doesn't ' override the equals() method of Object class.

PERFORMANCE TEST OF STRING AND STRINGBUFFER

```
public class ConcatTest{
    public static String concatWithString()    {
        String t = "Java";
        for (int i=0; i<10000; i++){
            t = t + "Tpoint";
        }
        return t;
    }
```

```
public static String concatWithStringBuffer(){
    StringBuffer sb = new StringBuffer("Java");
    for (int i=0; i<10000; i++){
        sb.append("Tpoint");
    }
    return sb.toString();
}
```

```
public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        concatWithString();
        System.out.println("Time taken by Concating with String: "+(System.currentTimeM
illis()-startTime)+"ms");
        startTime = System.currentTimeMillis();
        concatWithStringBuffer();
        System.out.println("Time taken by Concating with  StringBuffer: "+(System.curren
tTimeMillis()-startTime)+"ms");
    }
}
```

STRING AND STRINGBUFFER HASHCODE TEST

As you can see in the program given below, String returns new hashcode value when you concat string but StringBuffer returns same

```
public class InstanceTest{
    public static void main(String args[]){
        System.out.println("Hashcode test of String:");
        String str="java";
        System.out.println(str.hashCode());
        str=str+"tpoint";
        System.out.println(str.hashCode());
        System.out.println("Hashcode test of StringBuffer:");
        StringBuffer sb=new StringBuffer("java");
        System.out.println(sb.hashCode());
        sb.append("tpoint");
        System.out.println(sb.hashCode());
    }
}
```

DIFFERENCE BETWEEN STRINGBUFFER AND STRINGBUILDER

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.

A list of differences between StringBuffer and StringBuilder are given below:

No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.

STRINGBUFFER EXAMPLE

```
//Java Program to demonstrate the use of StringBuffer class.
public class BufferTest{
    public static void main(String[] args){
        StringBuffer buffer=new StringBuffer("hello");
        buffer.append("java");
        System.out.println(buffer);
    }
}
```

STRINGBUILDER EXAMPLE

```
//Java Program to demonstrate the use of StringBuilder class.
public class BuilderTest{
    public static void main(String[] args){
        StringBuilder builder=new StringBuilder("hello");
        builder.append("java");
        System.out.println(builder);
    }
}
```

PERFORMANCE TEST OF STRINGBUFFER AND STRINGBUILDER

Let's see the code to check the performance of StringBuffer and StringBuilder classes.

```
//Java Program to demonstrate the performance of StringBuffer and StringBuilder classes.
public class ConcatTest{
    public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++){
            sb.append("Tpoint");
        }
        System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");
        startTime = System.currentTimeMillis();
        StringBuilder sb2 = new StringBuilder("Java");
        for (int i=0; i<10000; i++){
            sb2.append("Tpoint");
                }
        System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");
    }
}
```

HOW TO CREATE IMMUTABLE CLASS?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

EXAMPLE TO CREATE IMMUTABLE CLASS

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

```
public final class Employee{
final String pancardNumber;

public Employee(String pancardNumber){
this.pancardNumber=pancardNumber;
}

public String getPancardNumber(){
return pancardNumber;
}

}
```

The above class is immutable because:

1. The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
2. The class is final so we cannot create the subclass.
3. There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

JAVA TOSTRING() METHOD

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

ADVANTAGE OF JAVA TOSTRING() METHOD

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

UNDERSTANDING PROBLEM WITHOUT TOSTRING() METHOD

Let's see the simple code that prints reference.

```
class Student{
 int rollno;
 String name;
 String city;
 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }
 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");
   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

As you can see in the above example, printing s1 and s2 prints the hashcode values of the objects but I want to print the values of these objects. Since java compiler internally calls toString() method, overriding this method will return the specified values. Let's understand it with the example given below:

EXAMPLE OF JAVA TOSTRING() METHOD

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }
public String toString(){//overriding the toString() method
  return rollno+" "+name+" "+city;
 }
 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");

   System.out.println(s1);//compiler writes here s1.toString()
   System.out.println(s2);//compiler writes here s2.toString()
 }
}
```

STRING METHODS

JAVA STRING CHARAT()

The **java string charAt()** method returns *a char value at the given index number*.

The index number starts from 0 and goes to n-1, where n is length of the string. It returns **StringIndexOutOfBoundsException** if given index number is greater than or equal to this string length or a negative number.

```
public class CharAtExample{
public static void main(String args[]){
String name=“javaLanguage";
char ch=name.charAt(4);//returns the char value at the 4th index
System.out.println(ch);
}}
```

JAVA STRING COMPARETO()

The **java string compareTo()** method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

It compares strings on the basis of Unicode value of each character in the strings.

If first string is lexicographically greater than second string, it returns positive number (difference of character value). If first string is less than second string lexicographically, it returns negative number and if first string is lexicographically equal to second string, it returns 0.

```
public class CompareToExample{
public static void main(String args[]){
String s1="hello";
String s2="hello";
String s3="meklo";
String s4="hemlo";
String s5="flag";
System.out.println(s1.compareTo(s2));//0 because both are equal
System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
}}
```

JAVA STRING CONCAT

The **java string concat()** method *combines specified string at the end of this string*. It returns combined string. It is like appending another string.

```
public class ConcatExample{
public static void main(String args[]){
String s1="java string";
s1.concat("is immutable");
System.out.println(s1);
s1=s1.concat(" is immutable so assign it explicitly");
System.out.println(s1);
}}
```

JAVA STRING CONTAINS()

The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.

```
class ContainsExample{
public static void main(String args[]){
String name="what do you know about me";
System.out.println(name.contains("do you know"));
System.out.println(name.contains("about"));
System.out.println(name.contains("hello"));
}}
```

JAVA STRING ENDSWITH()

The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

```
public class EndsWithExample{
public static void main(String args[]){
String s1="java by javatpoint";
System.out.println(s1.endsWith("t"));
System.out.println(s1.endsWith("point"));
}}
```

JAVA STRING EQUALS()

The **java string equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of Object class.

```
public class EqualsExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equals(s2));//true because content and case is same
System.out.println(s1.equals(s3));//false because case is not same
System.out.println(s1.equals(s4));//false because content is not same
}}
```

JAVA STRING EQUALSIGNORECASE()

The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string. It is like equals() method but doesn't check case. If any character is not matched, it returns false otherwise it returns true.

```
public class EqualsIgnoreCaseExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equalsIgnoreCase(s2));//true because content and case both ar
e same
System.out.println(s1.equalsIgnoreCase(s3));//true because case is ignored
System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same
}}
```

JAVA STRING FORMAT()

The **java string format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()* method.

The format() method of java language is like *sprintf()* function in c language and *printf()* method of java language.

```
public class FormatExample{
public static void main(String args[]){
String name="sonoo";
String sf1=String.format("name is %s",name);
String sf2=String.format("value is %f",32.33434);
String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional
part filling with 0


System.out.println(sf1);
System.out.println(sf2);
System.out.println(sf3);
}}
```

JAVA STRING GETBYTES()

The **java string getBytes()** method returns the byte array of the string. In other words, it returns sequence of bytes.

```
public class StringGetBytesExample{
public static void main(String args[]){
String s1="ABCDEFG";
byte[] barr=s1.getBytes();
for(int i=0;i<barr.length;i++){
System.out.println(barr[i]);
}
}}
```

JAVA STRING GETCHARS()

The **java string getChars()** method copies the content of this string into specified char array. There are 4 arguments passed in getChars() method. The signature of getChars() method is given below:

```
public class StringGetCharsExample{
public static void main(String args[]){
 String str = new String("hello javatpoint how r u");
      char[] ch = new char[10];
      try{
         str.getChars(6, 16, ch, 0);
         System.out.println(ch);
      }catch(Exception ex){System.out.println(ex);}
}}
```

JAVA STRING INDEXOF()

The **java string indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

```
public class IndexOfExample{
public static void main(String args[]){
String s1="this is index of example";
//passing substring
int index1=s1.indexOf("is");//returns the index of is substring
int index2=s1.indexOf("index");//returns the index of index substring
System.out.println(index1+"  "+index2);//2 8
//passing substring with from index
int index3=s1.indexOf("is",4);//returns the index of is substring after 4th index
System.out.println(index3);//5 i.e. the index of another is
//passing char value
int index4=s1.indexOf('s');//returns the index of s char value
System.out.println(index4);//3
}}
```

JAVA STRING INTERN()

The **java string intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from memory, if it is created by new keyword. It creates exact copy of heap string object in string constant pool.

```
public class InternExample{
public static void main(String args[]){
String s1=new String("hello");
String s2="hello";
String s3=s1.intern();//returns string from pool, now it will be same as s2
System.out.println(s1==s2);//false because reference variables are pointing to differ
ent instance
System.out.println(s2==s3);//true because reference variables are pointing to same i
nstance
}}
```

JAVA STRING ISEMPTY()

The **java string isEmpty()** method checks if this string is empty or not. It returns *true*, if length of string is 0 otherwise *false*. In other words, true is returned if string is empty otherwise it returns false.

The isEmpty() method of String class is included in java string since JDK 1.6.

```
public class IsEmptyExample{
public static void main(String args[]){
String s1="";
String s2="java";

System.out.println(s1.isEmpty());
System.out.println(s2.isEmpty());
}}
```

JAVA STRING JOIN()

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.

There are two types of join() methods in java string.

```
public class StringJoinExample{
public static void main(String args[]){
String joinString1=String.join("-","welcome","to",“google");
System.out.println(joinString1);
}}
```

JAVA STRING LASTINDEXOF()

The **java string lastIndexOf()** method returns last index of the given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

```
public class LastIndexOfExample{
public static void main(String args[]){
String s1="this is index of example";//there are 2 's' characters in this sentence
int index1=s1.lastIndexOf('s');//returns last index of 's' char value
System.out.println(index1);//6
}}
```

JAVA STRING LENGTH()

The **java string length()** method length of the string. It returns count of total number of characters. The length of java string is same as the unicode code units of the string.

```
public class LengthExample{
public static void main(String args[]){
String s1="python";
System.out.println("string length is: "+s1.length());
}}
```

JAVA STRING REPLACE()

The **java string replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Since JDK 1.5, a new replace() method is introduced, allowing you to replace a sequence of char values.

```
public class ReplaceExample1{
public static void main(String args[]){
String s1="javatpoint is a very good website";
String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
System.out.println(replaceString);
}}
```

JAVA STRING REPLACEALL()

The **java string replaceAll()** method returns a string replacing all the sequence of characters matching regex and replacement string.

```
public class ReplaceAllExample1{
public static void main(String args[]){
String s1=“google is a very good website";
String replaceString=s1.replaceAll("a","e");//replaces all occurrences of "a" to "e"
System.out.println(replaceString);
}}
```

JAVA STRING SPLIT()

The **java string split()** method splits this string against given regular expression and returns a char array.

```
public class SplitExample{
public static void main(String args[]){
String s1="java string split method demo";
String[] words=s1.split("\\s");//splits the string based on whitespace
//using java foreach loop to print elements of string array
for(String w:words){
System.out.println(w);
}
}}
```

JAVA STRING STARTSWITH()

The **java string startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

```
public class StartsWithExample{
public static void main(String args[]){
String s1="java string start with";
System.out.println(s1.startsWith("ja"));
System.out.println(s1.startsWith("java string"));
}}
```

JAVA STRING SUBSTRING()

The **java string substring()** method returns a part of the string.

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.

There are two types of substring methods in java string.

```
public class SubstringExample{
public static void main(String args[]){
String s1=“core java";
System.out.println(s1.substring(2,4));
System.out.println(s1.substring(2));
}}
```

JAVA STRING TOCHARARRAY()

The **java string toCharArray()** method converts this string into character array. It returns a newly created character array, its length is similar to this string and its contents are initialized with the characters of this string.

```
public class StringToCharArrayExample{
public static void main(String args[]){
String s1="hello";
char[] ch=s1.toCharArray();
for(int i=0;i<ch.length;i++){
System.out.print(ch[i]);
}
}}
```

JAVA STRING TOLOWERCASE()

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

```
public class StringLowerExample{
public static void main(String args[]){
String s1=" HELLO stRIng";
String s1lower=s1.toLowerCase();
System.out.println(s1lower);
}}
```

JAVA STRING TOUPPERCASE()

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

```
public class StringUpperExample{
public static void main(String args[]){
String s1="hello string";
String s1upper=s1.toUpperCase();
System.out.println(s1upper);
}}
```

JAVA STRING TRIM()

The **java string trim()** method eliminates leading and trailing spaces. The unicode value of space character is '\u0020'. The trim() method in java string checks this unicode value before and after the string, if it exists then removes the spaces and returns the omitted string.

```
public class StringTrimExample{
public static void main(String args[]){
String s1="  hello string   ";
System.out.println(s1+"java");//without trim()
System.out.println(s1.trim()+"java");//with trim()
}}
```

JAVA STRING VALUEOF()

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

```
public class StringValueOfExample{
public static void main(String args[]){
int value=30;
String s1=String.valueOf(value);
System.out.println(s1+10);//concatenating string with 10
}}
```

WRAPPER CLASSES IN JAVA

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive.*

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

USE OF WRAPPER CLASSES IN JAVA

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

Change the value in Method: Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.

Serialization: We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.

Synchronization: Java synchronization works with objects in Multithreading.

java.util package: The java.util package provides the utility classes to deal with objects.

Collection Framework: Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

The eight classes of the *java.lang* package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

AUTOBOXING

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

UNBOXING

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

```
//Java program to convert object into primitives
//Unboxing example of Integer to int
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}}
```