UNIT-2

Interface in Java

Interface in Java

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods
- The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

- Java Interface also **represents the IS-A relationship**.
- It cannot be instantiated just like the abstract class.
- Since Java 8, we can have **default and static methods** in an interface.
- Since Java 9, we can have **private methods** in an interface

- **A real-world example:**
  Let's consider the example of vehicles like bicycle, car, bike……, they have common functionalities. So we make an interface and put all these common functionalities. And lets Bicycle, Bike, car ….etc implement all these functionalities in their own class in their own way

- There are mainly three reasons to use interface.
- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling

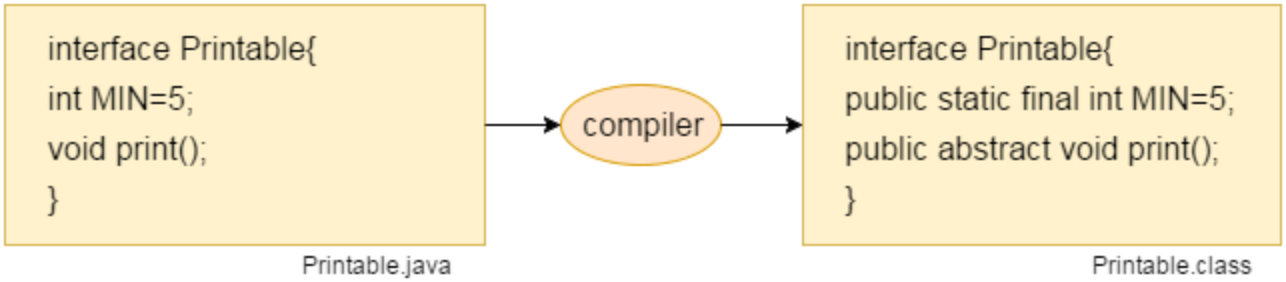
How to declare an interface

- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```



Java Interface Example

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class

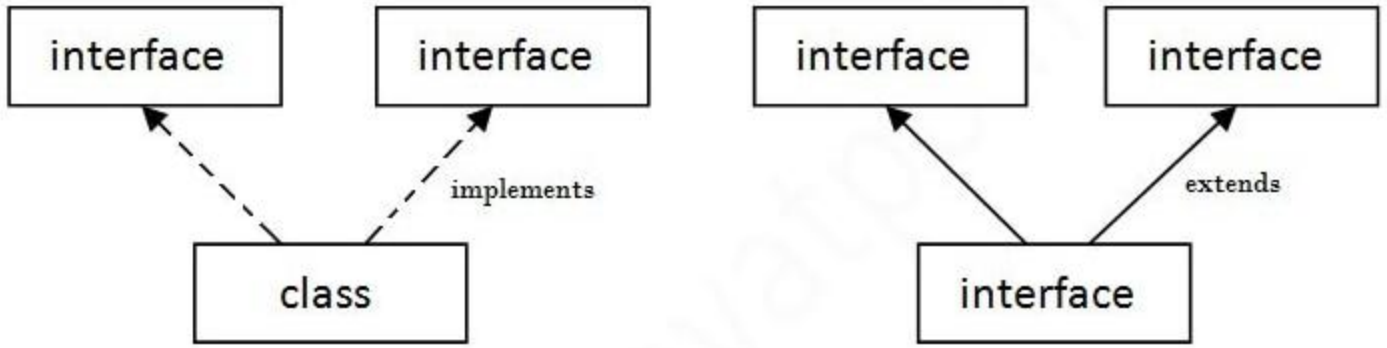
Java Interface Example: Drawable

```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

Java Interface Example: Drawable

- In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

Multiple inheritance in Java by interface



Multiple Inheritance in Java

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

Multiple inheritance in Java by interface

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Test it Now

```
Output:Hello
       Welcome
```

Key points to remember about interfaces

- We can't instantiate an interface in java. That means we cannot create the object of an interface
- Interface provides full abstraction as none of its methods have body. On the other hand abstract class provides partial abstraction as it can have abstract and concrete(methods with body) methods both.
- "implements" keyword is used by classes to implement an interface.
- While providing implementation in class of any method of an interface, it needs to be mentioned as public.
- Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.
- Interface cannot be declared as private, protected or transient.
- All the interface methods are by default abstract and public.
- Variables declared in interface are public, static and final by default.

Advantages of interface in java

- Without bothering about the implementation part, we can achieve the security of implementation
- In java, multiple inheritance is not allowed, however you can use interface to make use of it as you can implement more than one interface.

DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

ABSTRACT CLASS	INTERFACE
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Sinc Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword extends.	An **interface class** can be implemented using keyword implements
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

FINAL KEYWORD

- Final keyword can be used along with **variables, methods and classes.**
- **1) final variable**
- **2) final method**
- **3) final class**

1. Java final variable

- A final variable is a variable whose value cannot be changed at anytime once assigned, it remains as a constant forever.

2. Java final method

- When you declare a method as final, then it is called as final method. A final method cannot be overridden.

3. Java final class

- A final class cannot be extended(cannot be subclassed), lets take a look into the below example package com.javainterviewpoint;

OBJECT CLONING

- The **object cloning is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.**
- The **java.lang.Cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.**
- The **clone() method is defined in the Object class.**

Syntax of the clone() method:

protected Object clone() throws CloneNotSupportedException

Advantage of Object cloning

- You don't need to write lengthy and repetitive codes.
- Just use an abstract class with a 4- or 5-line long clone() method.
- It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project.
- Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done.
- Clone() is the fastest way to copy array.

Disadvantage of Object cloning

- To use the Object.clone() method, we have to change a lot of syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone() etc.
- Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.
- Object.clone() doesn?t invoke any constructor so we don?t have any control over object construction.

Example of clone() method (Object cloning)

```
class Student implements Cloneable{
int rollno;
String name;
Student(int rollno,String
name){ this.rollno=rollno;
this.name=name;
}
public Object clone()throws CloneNotSupportedException{
return super.clone();
}
public static void main(String args[]){
try{
Student s1=new Student(101,"amit");
Student s2=(Student)s1.clone();
System.out.println(s1.rollno+" "+s1.name);
System.out.println(s2.rollno+" "+s2.name);
}
catch(CloneNotSupportedException c){}
}
}
```

Output:

```
101 amit
101 amit
```

INNER CLASSES

- Inner class means one class which is a member of another class. There are basically four types of inner classes in java.
- 1) Nested Inner class
- 2) Method Local inner classes
- 3) Anonymous inner classes
- 4) Static nested classes

Nested Inner class

- Nested Inner class can access any private instance variable of outer class. Like any other instance variable, we can have access modifier private, protected, public and default modifier. Like class, interface can also be nested and can have access specifiers.

Example: Nested Inner class

```
class Outer {
// Simple nested inner class
public void show() {
System.out.println("In a nested class method");
}
}
}
class Main {
public static void main(String[] args) {
Outer.Inner in = new Outer().new Inner();
in.show();
}
}
```

Output:

In a nested class method ass class Inner

Method Local inner classes

- Inner class can be declared within a method of an outer class. In the following example, Inner is an inner class in outerMethod().

Example:

```
class Outer {
void outerMethod() {
 System.out.println("inside outerMethod"); // Inner class is local to
    outerMethod() class Inner {
void innerMethod() {
System.out.println("inside innerMethod");
}
}
Inner y = new Inner();
y.innerMethod();
}
}
class MethodDemo {
public static void main(String[] args) {
Outer x = new Outer(); x.outerMethod();
}
}
```

Output:

Inside outerMethod Inside innerMethod

Static nested classes

- Static nested classes are not technically an inner class. They are like a static member of outer

Example:

```
class Outer {
private static void outerMethod() {
System.out.println("inside outerMethod");
}
// A static inner class static class Inner {
public static void main(String[] args) {
System.out.println("inside inner class Method");
 outerMethod();
}
}
}
```

Output:

inside inner class Method inside outerMethod

Anonymous inner classes

- Anonymous inner classes are declared without any name at all. They are created in two ways.

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Method Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

STRINGS IN JAVA

- In java, string is basically an object that represents sequence of char values. Java String provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc.
- In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable. String s="javatpoint";
- There are two ways to create String object:
- 1. By string literal
- 2. By new keyword

- **1) String Literal**
- Java String literal is created by using double quotes. For Example: String s="welcome";
- **2) By new keyword**
- String s=new String("Welcome");
- Page

- 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','t','p','o','i','n','t'};
- String s=**new** String(ch);

String s="welcome";

String methods:

String methods:		
1.	char charAt(int index)	returns char value for the particular index
2.	int length()	returns string length
3.	static String format(String format, Object... args)	returns formatted string
4.	static String format(Locale l, String format, Object... args)	returns formatted string with given locale
5.	String substring(int beginIndex)	returns substring for given begin index
6.	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
7.	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
8.	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string
9.	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string
10.	boolean equals(Object another)	checks the equality of string with object
11.	boolean isEmpty()	checks if string is empty
12.	String concat(String str)	concatinates specified string

14.	String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
15.	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16.	String[] split(String regex)	returns splitted string matching regex
17.	String[] split(String regex, int limit)	returns splitted string matching regex and limit
18.	String intern()	returns interned string
19.	int indexOf(int ch)	returns specified char value index
20.	int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index
21.	int indexOf(String substring)	returns specified substring index
22.	int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
23.	String toLowerCase()	returns string in lowercase.
24.	String toLowerCase(Locale l)	returns string in lowercase using specified locale.
25.	String toUpperCase()	returns string in uppercase.
26.	String toUpperCase(Locale l)	returns string in uppercase using specified locale.
27.	String trim()	removes beginning and ending spaces of this string

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);
}}
```

Test it Now

```
java
strings
example
```

ArrayList class declaration Syntax

public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Clone able, Serializable

Java ArrayList class

- Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.
- The important points about Java ArrayList class are:
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred
- if any element is removed from the array list.

Constructors of Java ArrayList

CONSTRUCTOR	DESCRIPTION
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of Java ArrayList

METHOD	DESCRIPTION
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void clear()	It is used to remove all of the elements from this list.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

Java ArrayList Example: Book Example:

```
import java.util.*; class Book {
int id;
String name,author,publisher; int quantity;
public Book(int id, String name, String author, String
   publisher, int quantity) {
this.id = id;
this.name = name;
this.author = author;
this.publisher = publisher;
this.quantity = quantity;
}
}
```

```
public class ArrayListExample { public static void main(String[]
    args) {
//Creating list of Books
List<Book> list=new ArrayList<Book>();
//Creating Books
Book b1=new Book(101,"Let us C","Yashwant
    Kanetkar","BPB",8);
Book b2=new Book(102,"Data Communications &
    Networking","Forouzan","Mc Graw Hill",4);
Book b3=new Book(103,"Operating
    System","Galvin","Wiley",6);
//Adding Books to list list.add(b1); list.add(b2); list.add(b3);
    //Traversing list for(Book b:list){
System.out.println(b.id+" "+b.name+" "+b.author+"
    "+b.publisher+" "+b.quantity);
}
}
}
```

Java String

Java String

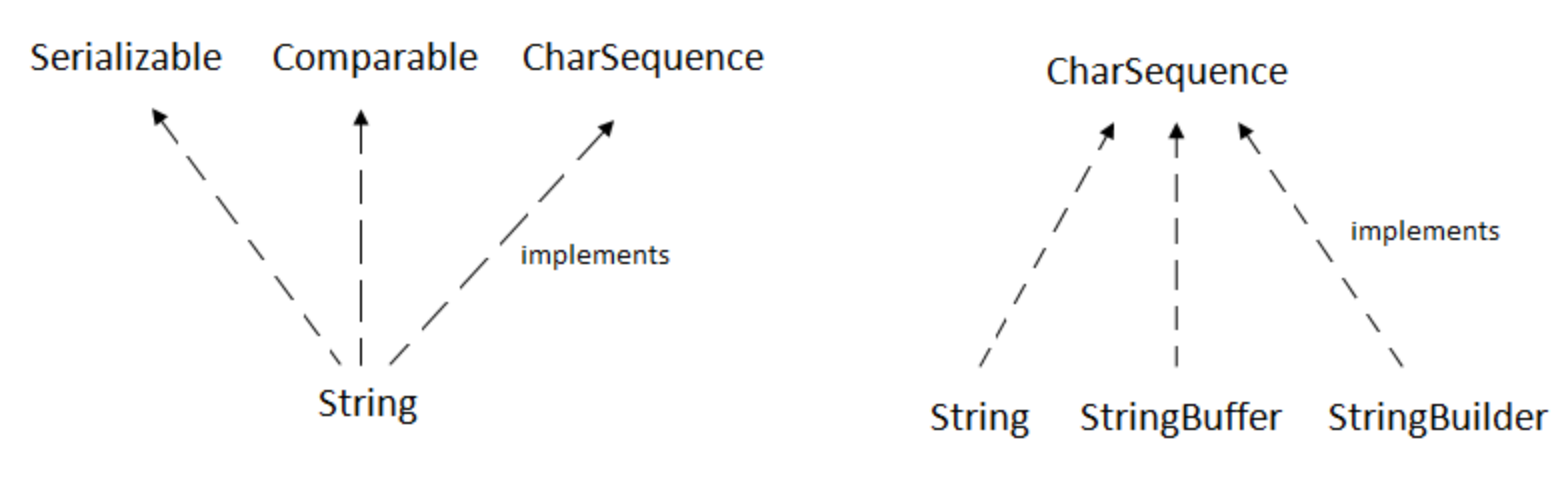
Java String

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','t','p','o','i','n','t'};
String s=new String(ch);
String s="javatpoint";
```

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



Java String compare

- 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:

By equals() method

By = = operator

By compareTo() method

```
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
 }
}
```

Test it Now

```
Output:Sachin
```

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

Test it Now

```
Output:Sachin Tendulkar
```

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
 }
}
```

Test it Now

```
Output:true
       true
       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
 }
}
```

Test it Now

Output:

```
false
true
```

2) 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)
 }
}
```

Test it Now

```
Output:true
       false
```

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 )
 }
}
```

Test it Now

```
Output:0
       1
       -1
```

1) 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
 }
}
```

Test it Now

```
Output:Sachin Tendulkar
```

2) 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
  }
}
```

Test it Now

```
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.
- **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
- **startIndex:** inclusive
- **endIndex:** 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
 }
}
```

Test it Now

```
Tendulkar
Sachin
```

Java String toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and stri
letter.

```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
```

Test it Now

```
SACHIN
sachin
Sachin
```

Java String length() method

The string length() method returns length of the string.

```
String s="Sachin";
System.out.println(s.length());//6
```

Test it Now

```
6
```

Java String replace() method

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

```
String s1="Java is a programming language. Java is a platform. Java is an Island.";
String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
System.out.println(replaceString);
```

Output:

```
Kava is a programming language. Kava is a platform. Kava is an Island.
```

- **Output:**
- 101 Let us C Yashwant Kanetkar BPB 8
- 102 Data Communications & Networking Forouzan Mc Graw Hill 4 103 Operating System Galvin Wiley 6