Que. Give difference between final, finally and finalize.
Ans.

No.	**Final**	**finally**	**Finalize**
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

Que. How to compare two Objects ?
Ans.
➔ In java **= = operator** used to compare object's **references**, not values.
➔ **equals( )** method of Object class can use to compare objects but **only for String class objects and wrapper class objects.** (Prof. Viral S. Patel)
Example :

```
Integer x = new Integer(5);
Integer y = new Integer(5);
Integer z = new Integer(10);

System.out.println(x.equals(y));   // display true
System.out.println(x.equals(z));   // display false

String s1 = "VIRAL";
String s2 = "VIRAL";
String s3 = "PATEL";

System.out.println(s1.equals(s2));   // display true
System.out.println(s1.equals(s3));   // display false

String s4 = "VIRAL";
String s5 = "viral";

System.out.println(s4.equals(s5));   // display false
System.out.println(s4.equalsIgnoreCase(s5));   // display true
```

➔**For other classes equals method compare only references**. **If values of two objects are same still return false.**

Example :

```
class Myclass1
{
   int a;

   Myclass1(int a)
   {
       this.a = a;
   }
}
```

```
class Cmp1
{
   public static void main(String args[])
   {
      Myclass1 obj1 = new Myclass1(5);
      Myclass1 obj2 = new Myclass1(5);
      Myclass1 obj3 = obj1; // create new reference of obj1
      System.out.println(obj1.equals(obj2)); //display false
              // because references are of different objects
      System.out.println(obj1.equals(obj3)); // display true
                  // because references are of same object
   }
}
```

➔ So to compare Objects of other classes in java, we have to **override equals( ) method** of Object class **or** have to **create separate method** and compare values of one object with other object. (Prof. Viral S. Patel)

Example :

```
class Myclass2
{
   int a;
   int b;

   Myclass2(int a,int b)
   {
       this.a = a;
       this.b = b;
   }

// Override equals method
   boolean equals(Object o)
   {
      MyClass2 ob = (MyClass2) o;

       if(a==ob.a && b==ob.b)
          return true;
       else
          Return false;
   }

// create new own method of class
   boolean cmpObjects(Myclass2 o)
   {
       if(a==o.a && b==o.b)
          return true;
       else
          Return false;
   }
}
```

```
class Cmp2
{
   public static void main(String args[])
   {
      Myclass2 obj1 = new Myclass2(5,6);
      Myclass2 obj2 = new Myclass2(5,6);
      Myclass2 obj3 = new Myclass2(5,7);

      System.out.println(obj1.equals(obj2));
                                    //display true

      System.out.println(obj1.equals(obj3));
                                    //display false

  System.out.println(obj1.cmpObjects(obj2));
                                     //display true

  System.out.println(obj1.cmpObjects(obj3));
                                     //display false

   }
}
```

Note :

MyClass2 ob = (MyClass2) o;

This statement is used to casting Object class reference to convert in MyClass2 reference.

Que. Explain the use of Super keyword in java.

Ans. Whenever a subclass needs to **refer to its immediate superclass**, it can do so by use of the keyword super.

➔ super has two general forms :

The first calls the superclass' constructor.

The second is used to access a member of the superclass that has been hidden (override) by a member of a subclass.

Using super to Call Superclass Constructors :

➔ super( ) must always be the first statement executed inside a subclass' constructor.

Example :

```
class SuperClass
{
   int a;
   SuperClass(int z)
   {
       a = z;
   }
}
```

```
class SubClass extends SuperClass
{
   int b;
   SubClass(int x,int y)
   {
       super(x);  // call to SuperClass Constructor
       b = y;
   }
}
class DemoSuper
{
   public static void main(String args[])
   {
      SubClass obj1 = new Subclass(2,3);  // a=2 & b=3
   }
}
```

A Second Use for super :
➔The second form of super acts somewhat like this, except that it always refers to the superclass. But it is used in subclass.
➔general form: **super.member**
Here, member can be either a method or an instance variable.
➔This second form of super is most applicable to situations in which members of a subclass hide members of superclass by using the **same name**.

```
class A
{
   int a;
   void disp()
   {
     System.out.println(a);
   }
}

class B extends A
{
   int a;
   B(int x,int y)
   {
     super.a = x; // a in class A
     a = y;  // a in class B
   }
   void disp() // override method
   {
      super.disp(); // call disp in A
      System.out.println(a);
   }
}
```

```
class DemoSuper
{
   public static void main(String args[])
   {
      B obj = new Obj(2,3);
      obj.disp();
   }
}

Output :
2
3
```

Que. Explain the calling sequence of default constructor with example.
Ans. If super( ) is not used, then the default (parameterless) constructor of each superclass will be executed.

The constructors are called in order of derivation. Because a superclass has no knowledge of any subclass, any initialization it needs to perform is separate from and possibly before to any initialization performed by the subclass. Therefore, it must be executed first.

For example, given a subclass called B and a superclass called A, is A’s constructor called before B’s.

```
class A
{
   A()
   {
      System.out.println(“A”);
   }
}
class B extends A
{
   B()
   {
      System.out.println(“B”);
   }
}
```

```
class C extends B
{
   C()
   {
      System.out.println(“C”);
   }
}
class DemoSuper
{
   public static void main(String args[])
   {
      C obj = new C();
   }
}
Output :
A
B
C
```

Que. Give difference between Abstract Class and Interface
Ans.

ABSTRACT CLASS	INTERFACE
Abstract class can have **abstract and non-abstract (concrete) methods**	Interface can have only **abstract methods.**

abstract keyword must be use to make method abstract.
abstract class A
{
   abstract void disp(); // abstract must be write
}
	No need to use abstract keyword as by default method become abstract.
	Interface A
	{
	      void disp(); // **abstract by default**
	}

Abstract class doesn’t support multiple inheritance.
class A { … }
class B { … }
class C extends A,B // give compile time error
{ …
}
	Interface supports multiple inheritance.
	interface A { … }
	interface B { … }
	class C implements A,B //correct
	{ …
	}
	We can also write
	interface C extends A,B //correct
	{ …
	}

Abstract class can have **final, non-final, static and non-static variables.**	Interface has **only static and final variables**. (no need to use ‘final’and ‘static’ words because implicitly by default they are static and final. Only have to specify value with that variables.)
	interface A
	{
	     int i = 0; // i is implicitly static and final
	}

Abstract class can **have static methods, main**	Interface **can’t have static methods, main method**

method and constructor.	**or constructor.** (Prof. Viral S. Patel)

Abstract class can provide the implementation of interface.
Example:
interface A
{
}
abstract Class B implements A //correct
{
 	
}
	Interface can't provide the implementation of abstract class.
	Example :
	abstract class A
	{
	}
	interface B extends A //error
	{
	
	}

Abstract class can extend another java class and implement multiple java interfaces.	An interface can **extend** another **interface only**.
Abstract class have **private, protected or public** members.	Members (variables and methods) of a Java interface are **public by default.**

The method which we define in child class (inherited from the abstract class) is not necessary to be public.
abstract class A
{
    abstract void disp();
}
class B extends A //correct
{
    void disp() { … }
}
	The method which we implement in child class (implements interface) must have to be public so we must have to write public keyword before it.
	interface A
	{
	    void disp();
	}
	class B implements A //correct
	{
	    public void disp() { … }
	}

Que. What is Interface ? (2)
Ans. An interface in java is a blueprint of a class. It is a fully abstract class. It has static constants and abstract methods (without method body) only. It is used to achieve fully abstraction and multiple inheritance in Java.
Ex.

```
interface Printable
{
        public static final int min=5;
        public abstract void print();
}
```

Que. Explain abstract keyword. (2)
Ans. A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).
➔Abstract class needs to be extended and its abstract method have to implemented by child class. It cannot be instantiated (i.e. we cannot create object of abstract class).
➔ A method that is declared as abstract and does not have implementation is known as abstract method.
➔ Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Que. Explain Package in Java. How to create and use package in java.

Ans. Packages are containers for classes that are used to keep the class name space compartmentalized. JAVA API provides a large number of classes grouped into different packages according to functionality. (Prof. Viral S. Patel)

➔To create package simply include a package command as the first statement in a java source file.

➔General form of the package statement : **package *pkg*;**

➔General form of multilevel packages statement : **package *pkg1[.pkg2][.pkg3];***

➔**Example : package java.applet.Applet;**

➔To execute the program in package, package have to be in current directory, or a subdirectory of the current directory and also have to specify a directory path or paths by setting the **CLASSPATH** environmental variable.

➔The easiest way to try the examples shown in this book is to simply create the package directories below your current development directory, put the .class files into the appropriate directories and then execute the programs from the development directory.

➔we can also create package in current directory from command line by using command.

Example: Suppose MyJavaFile.java have fist statement 'package p1' and then 'Demo' class. By using command (Prof. Viral S. Patel)

javac –d . MyJavaFile.java

we can create package p1 in current directory and Demo.class file in it.
This Demo class can be executed by command

java p1.Demo

➔Java package can be accessed either using a fully qualified class name or using import statement that can be used to search a list of packages for a particular class.

import *pkg1[.pkg2][.pkg3].classname;*

To import all classes from package : **import *pkg1[.pkg2][.pkg3].*;***

➔**Example :**

```
package p1;

public class A
{
  public void display()
  {
       System.out.println("Class A");
  }
}
```

```
import p1.A;

class PackageTest1
{
   public static void main(String args[])
   {
             A obj = new A();
             Obj.display();
   }
}
```

Que. How Java restricts the access of classes and methods at package level using access modifiers ?

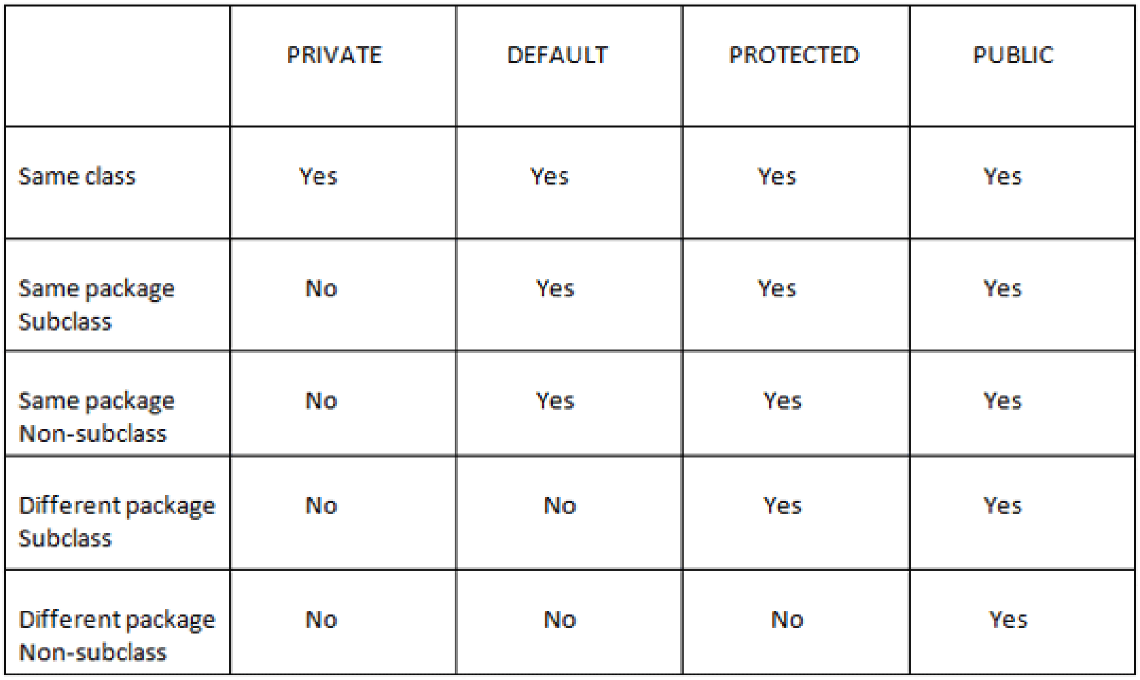
Ans. Java addresses four categories of visibility for class members:

■ Subclasses in the same package
■ Non-subclasses in the same package
■ Subclasses in different packages
■ Non-subclass in different package

The Four types of access specifiers – default, private, public, and protected, provide a variety of ways to produce the many levels of access required by these categories.

➔Anything declared **public** can be accessed from anywhere.
➔Anything declared **private** cannot be seen outside of its class.
➔When a member does not have an explicit access specification , it is visible to subclasses as well as to other classes in the same package. This is the **default** access.
➔If we want to allow an element to be seen outside your current package, but only to classes that subclass your class directly, then we can declare that element **protected**.

A class has only two possible access levels: default and public.
➔When a class is declared as **public**, it is accessible by any other code.
➔If a class has **default** access, then it can only be accessed by other code within its same package.



	PRIVATE	DEFAULT	PROTECTED	PUBLIC
Same class	Yes	Yes	Yes	Yes
Same package Subclass	No	Yes	Yes	Yes
Same package Non-subclass	No	Yes	Yes	Yes
Different package Subclass	No	No	Yes	Yes
Different package Non-subclass	No	No	No	Yes

Que. How default access modifier is different than public modifier ? (2)

Ans. In same package there in no difference but in different package subclass/non subclass concept public data members or public functions can be access but default not access. Default modifier allow to access only for same package classes.

Que. Difference between length and length( ).
Ans.

Length	**length( )**
length - Used to find the length of array.	**length( )- Used to find the length of String**
Example : int a[]={1,2,3,4}; ➔ a.length ➔ 4	**Example :** String x="abc"; ➔ x.length() ➔ 3

Que. Difference between equals and compareTo methods.
Ans.

equals	**compareTo**
Syntax : boolean equals (Object anObject) (parameter is any object of Wrapper Class and String Class)	Syntax : int compareTo (Object anObject) (parameter is any object of Wrapper Class and String Class)
equals method return true or false value.	s1.compareTo(s2) returns positive value if s1 > s2. It returns 0 value if s1 = s2. It returns negative value if s1 < s2.
equals only tells you whether they're equal or not.	compareTo gives information on how the Strings compare lexicographically. So we can know one string is large, small or equal to other string according to their ASCII values.

Ex. String Class objects
String s1 = “ABC”;
String s2 = “XYZ”;
System.out.println(s1.equals(s2));
Output : false
	Ex. String Class objects
	String s1 = “ABC”; // ASCII value of A 65
	String s2 = “XYZ”; // ASCII value of X 88
	System.out.println(s1.compareTo(s2)); // 65-88
	Output : -23

Ex. Wrapper Class objects
Integer a = new Integer(6);
Integer b = new Integer(5);
System.out.println(a.equals(b));
Output : false
	Ex. Wrapper Class objects
	Integer a = new Integer(6);
	Integer b = new Integer(5);
	System.out.println(a.compareTo(b));
	Output : 1

Que. What is Anonymous Inner Classes ?
Ans. An inner class without a name. It allows the declaration of the class, creation of the object and execution of the methods in it at one shot. It should be used if you have to override method of class or interface.

```
class A
{
  public void read()
  {
   System.out.println("Programmer Interview!");
  }
}
class B
{
  public static void main(String args[])
  {
  /*  This creates an anonymous inner class: */
     A obj = new A() {
      public void read()
      {
       System.out.println("anonymous A");
      }
     };
  obj.read();
  }
}
```

***note** : Anonymous class is created but its name is decided by the compiler which extends the class 'A' and provides the implementation of the read() method as given here.

```
class AnonymousInner1 extends A
{
   AnonymousInner1(){ }

   public void read()
   {
   System.out.println("anonymous A");
   }
}
```

Que. Difference between = = and equals method.
Ans.

= =	**equals**
'= =' operator compares references. It does not compare values.	'equals' method compares the original content of the string type objects.
Ex. String s1 = "ABC"; String s2 = new String("ABC") String s3 = s1; System.out.println(s1 = = s2); System.out.println(s1 = = s3); **Output :** false true	Ex. String s1 = "ABC"; String s2 = new String("ABC") String s3 = "VSP"; System.out.println(s1.equals(s2)); System.out.println(s1.equals(s3)); **Output :** true false

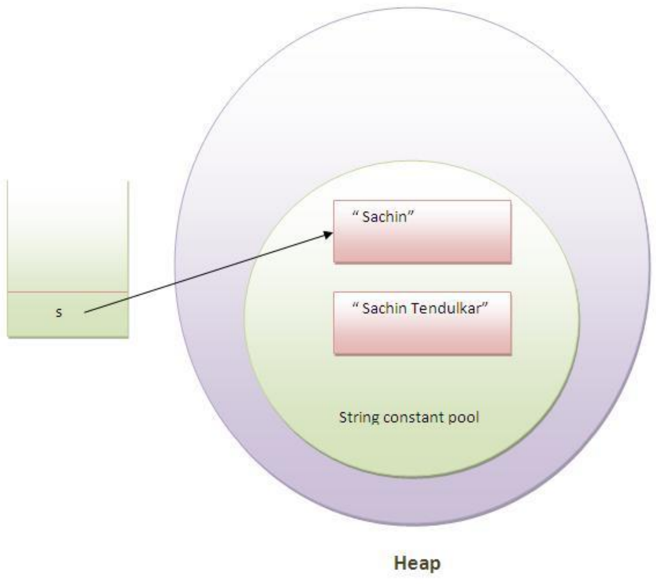
Que. Explain String is immutable in java.

Ans. 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. Example:

```
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

Output: Sachin
```

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



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 explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object. For example:

```
String s="Sachin";
s=s.concat(" Tendulkar");
System.out.println(s);
```

```
Output: Sachin Tendulkar
```

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

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.

Que. Difference between String and StringBuffer class.
Ans.

String	**StringBuffer**
String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when we concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concat 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.

Que. List functions of String Class. Explain any seven.
Ans.

length()	**startsWith()**	**concat()**
charAt()	**endsWith()**	**replace()**
getChars()	**compareTo()** //method of java.lang.Comparable interface	**trim()**
toCharArray()	**compareToIgnoreCase( )**	**toLowerCase()**
equals() //method of Object class	**indexOf()**	**toUpperCase()**
equalsIgnoreCase()	**lastIndexOf()**	
regionMatches()	**substring()**	

length( ) : calculate length of a string means the number of characters that it contains.
Syntax : int length( )
Example :

```
class A
{
  public static void main(String args[])
  {
     String s1 = "VSP";
     System.out.println(s1.length( ));
  }
}
```

Output : 3

charAt( ) : extract a single character from a String.
Syntax : char charAt(int index)
Example :

```
class A
{
  public static void main(String args[])
  {
     char ch = "VSP".charAt(1);
     System.out.println(ch);
  }
}
```

Output : S

getChars( ) : Extract more than one character at a time (substring) and store in char array.

Syntax : void getChars(int startIndex, int endIndex, char arr[ ], int arrStartIndex)

The substring contains the characters from ‘startIndex’ to ‘endIndex’–1. The array that will receive the characters is specified by ‘arr’. The index within ‘arr’ at which the substring will be copied is passed in ‘arrStartIndex’.

Example :

```
class A
{
  public static void main(String args[])
  {
      String s = "Prof Viral Patel";
      int start = 5;
      int end = 10;
      char arr[] = new char[end - start];
      s.getChars(start, end, arr, 0);
     System.out.println(arr);
  }
}
```

Output : Viral

toCharArray( ) : convert all the characters in a String object into a character array

Syntax : char[ ] toCharArray( )

Example :

```
class A
{
  public static void main(String args[])
  {
      String s = "Prof Viral Patel";
      char arr[] = s.toCharArray( );
     System.out.println(arr);
  }
}
```

Output : Prof Viral Patel

equals( ) and equalsIgnoreCase( ) :To compare two strings for equality.

Syntax : boolean equals(Object str)
boolean equalsIgnoreCase(String str)

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

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

Example :

```
class A
{
  public static void main(String args[])
  {
        String s1 = "Prof";
        String s2 = "viral";
        String s3 = "VIRAL";
        String s4 = "viral";
        System.out.println(s1.equals(s2));
        System.out.println(s2.equals(s3));
        System.out.println(s2.equals(s4));
        System.out.println(s2.equalsIgnoreCase(s3));
   }
}
```

Output : false
false
true
true

startsWith( ) and endsWith( ) :
Syntax : boolean startsWith(String str)
boolean endsWith(String str)
boolean startsWith(String str, int startIndex)
startsWith( ) method determines whether a given String begins with a specified string. Conversely, endsWith( ) determines whether the String in question ends with a specified string.
Example :

```
class A
{
   public static void main(String args[])
   {
        System.out.println("Foobar".endsWith("bar"));
        System.out.println("Foobar".startsWith("Foo"));
        System.out.println("Foobar".startsWith("bar",3));
   }
}
```

Output : true
true
true

compareTo( ) and compareToIgnoreCase( ): To compare two strings as one is less than, equal to, or greater than the another.
Syntax : int compareTo(String str)
int compareToIgnoreCase(String str)

s1.compareTo(s2)
if s1 > s2 It returns positive value.
if s1 = s2 It returns 0 value.
if s1 < s2 It returns negative value.

Example :

```
class A
{
  public static void main(String args[])
  {
     String s1 = "ABC";  // ASCII value of A 65
     String s2 = "XYZ";  // ASCII value of X 88
     System.out.println(s1.compareTo(s2)); // 65-88
  }
}
```

Output : -23

compareToIgnoreCase( ) method returns the same results as compareTo( ), except that case differences are ignored.

IndexOf( ) and lastIndexOf( ) : indexOf( ) Searches for the first occurrence of a character or substring. lastIndexOf( ) Searches for the last occurrence of a character or substring.

Syntax :

```
int indexOf(int ch)
int lastIndexOf(int ch)
```

To search for the first or last occurrence of a substring, use

```
int indexOf(String str)
int lastIndexOf(String str)
```

We can specify a starting point for the search using these forms:

```
int indexOf(int ch, int startIndex)
int lastIndexOf(int ch, int startIndex)
int indexOf(String str, int startIndex)
int lastIndexOf(String str, int startIndex)
```

Example :

```
class A
{
   public static void main(String args[])
   {
     String s = "Now is the time for all good men " +
                "to come to the aid of their country.";
     System.out.println( s.indexOf('t'));              //  7
     System.out.println(s.lastIndexOf('t'));           //  65
     System.out.println(s.indexOf("the"));             //  7
     System.out.println(s.lastIndexOf("the"));         //  55
     System.out.println(s.indexOf('t', 10));           //  11
     System.out.println(s.lastIndexOf('t', 60));       //  55
     System.out.println(s.indexOf("the", 10));         //  44
     System.out.println(s.lastIndexOf("the", 60));     //  55
   }
}
```

substring() : It returns a portion of a String.
Syntax : String substring(int startIndex)
String substring(int startIndex, int endIndex)
Example :

```
class A
{
        public static void main(String args[])
       {
           String s= "Asst Prof Viral Patel";
           String s1 = s.substring(5);
           System.out.println(s1);
           String s2 = s.substring(10,15);
           System.out.println(s2);
        }
}
```

Output : Prof Viral Patel
Viral

concat( ) : We can concatenate two strings using concat( ).
Syntax : String concat(Strng str)
Example :

```
class A
{
  public static void main(String args[])
  {
      String s1 = “Viral”;
      String s2 = s1.concat(“Patel”)
      System.out.println(s2);
  }
}
```

Output : ViralPatel

replace( ) : The replace( ) method replaces all occurrences of one character in the invoking string with another character.
Syntax : String replace(char original, char replacement)

Example :

```
class A
{
  public static void main(String args[])
  {
      String s1 = “Viraj”.replace(‘j’,’l’ );
      System.out.println(s1);
  }
}
```

Output : Viral

trim( ) : The trim( ) method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.
Syntax : Stirng trim( )

Example :
```
class A
{
  public static void main(String args[])
  {
      String s1 = "       Prof Viral S Patel         ".trim( );
      System.out.println(s1);
  }
}
```

Output : Prof Viral S Patel

toLowerCase( ) and toUpperCase( ): The method toLowerCase( ) converts all the characters in a string from uppercase to lowercase. The toUpperCase( ) method converts all the characters in a string from lowercase to uppercase.
Syntax : Stirng toLowerCase( )
String toUpperCase( )

Example :
```
class A
{
  public static void main(String args[])
  {
      String s1 = "Prof Viral S Patel";
      String upper = s1.toUpperCase( );
      System.out.println(upper);
      String lower = s1.toLowerCase( );
      System.out.println(lower);
  }
}
```

Output :
```
PROF VIRAL S PATEL
prof viral s patel
```

Que. List functions of StringBuffer Class. Explain any seven.
Ans.

length()	**capacity()**	**setLength()**
ensureCapacity()	**charAt()**	**setCharAt()**
getChars()	**append()**	**insert()**
reverse()	**delete()**	**deleteCharAt()**
replace()	**substring()**	

length() : calculate length of a string means the number of characters that it contains.
capacity() : the total allocated capacity (memory space) can be found through this method.
Syntax : int length()
int capacity()
Example :

```
class A
{
  public static void main(String args[])
  {
     StringBuffer s = new StringBuffer("VSP");
     System.out.println(s.length());
     System.out.println(s.capacity());
  }
}
```

Output : 3
19

setLength() : To set the length of the buffer within a StringBuffer object. When we increase the size of the buffer, null characters are added to the end of the existing buffer. If we call setLength( ) with a value less than the current value returned by length( ), then the characters stored beyond the new length will be lost.
Syntax : void setLength(int len)
Example :

```
class A
{
  public static void main(String args[])
  {
     StringBuffer s = new StringBuffer("Viral");
     s.setLength(3);
     System.out.println(s);
}
```

Output : Vir

ensureCapacity() : It is used to set the size of the buffer. This is useful if we know in advance that we will be appending a large number of small strings to a StringBuffer.
Syntax : void ensureCapacity(int capacity)
Example :

```
class A
{
  public static void main(String args[])
  {
     StringBuffer s = new StringBuffer("Viral");
     System.out.println(s.capacity()); // (5 + 16 = 21)
     s.ensureCapacity(22); // (ensure capacity = old capacity * 2 + 2 = 21 * 2 + 2 = 44)
     System.out.println(s.capacity());
  }
}
```

Output : 21
44

charAt() : extract a single character from a String.
setCharAt() : set character at particular index of String.
Syntax : char charAt(int index)
void setCharAt(int index, char ch)
Example :

```
class A
{
  public static void main(String args[])
  {
     StringBuffer s = new StringBuffer("VSP");
     char ch = s.charAt(1);
     System.out.println(ch);
     s.setCharAt(1,'I');
     System.out.println(s);
  }
}
```

Output : S
VIP

getChars() : Extract more than one character at a time (substring) and store in char array.
Syntax : void getChars(int startIndex, int endIndex, char arr[ ], int arrStartIndex)
The substring contains the characters from 'startIndex' to 'endIndex'–1. The array that will receive the characters is specified by 'arr'. The index within 'arr' at which the substring will be copied is passed in 'arrStartIndex'.
Example :

```
class A
{
  public static void main(String args[])
  {
      StringBuffer s = new StringBuffer("Prof Viral Patel");
      int start = 5;
      int end = 10;
      char arr[] = new char[end - start];
      s.getChars(start, end, arr, 0);
     System.out.println(arr);
}
```

Output : Viral

append() : This method concatenates the given argument with current string. After the concatenation has been performed, the compiler inserts a **call to toString( )** to turn the modifiable StringBuffer back into a constant String.
Syntax : StringBuffer append(String str)
StringBuffer append(int num)
StringBuffer append(Object obj)
Example :

```
class A
```

```
{
        public static void main(String args[])
       {
           StringBuffer s=new StringBuffer("Hello ");
           s.append("Prof  ").append("VSP");   // s.append("Prof  ").append("VSP").toString();
           System.out.println(s);
        }
}
```

Output : Hello Prof VSP

---

insert() : Inserts one string into another.
Syntax : StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)
Example :

```
class A
{
        public static void main(String args[])
       {
           StringBuffer s=new StringBuffer("Hello  VSP");
           s.insert(6, "Prof  ");
           System.out.println(s);
        }
}
```

Output : Hello Prof VSP

---

reverse() : reverse the characters within a StringBuffer object
Syntax : StringBuffer reverse( )
Example :

```
class A
{
        public static void main(String args[])
       {
           StringBuffer s=new StringBuffer("VSP");
           s.reverse();
           System.out.println(s);
        }
}
```

Output : PSV

---

delete () : deletes a sequence of characters from the invoking object
deleteCharAt () : deletes the character from the specified index
Syntax : StringBuffer delete(int startIndex, int endIndex)
StringBuffer deleteCharAt(int loc)
Example :

```
class A
{
        public static void main(String args[])
       {
            StringBuffer s=new StringBuffer("Prof Viral Patel");
            s.delete(4,10);
            System.out.println(s);
            s.deleteCharAt(2);
            System.out.println(s);
        }
}
```

Output : Prof Patel
Prf Patel

---

replace() : It replaces one set of characters with another set inside a StringBuffer object.
Syntax : StringBuffer replace(int startIndex, int endIndex, String str)
Example :

```
class A
{
        public static void main(String args[])
        {
            StringBuffer s=new StringBuffer("This is a test.");
            s.replace(5,7, "was");
            System.out.println(s);
        }
}
```

Output : This was a test.

---

substring() : It returns a portion of a StringBuffer.
Syntax : String substring(int startIndex)
String substring(int startIndex, int endIndex)
Example :

```
class A
{
        public static void main(String args[])
        {
            StringBuffer s=new StringBuffer("Asst Prof Viral Patel");
            String s1 = s.substring(5);
            System.out.println(s1);
            String s2 = s.substring(10,15);
            System.out.println(s2);
        }
}
```

Output : Prof Viral Patel
Viral