Unit-II

What is Abstract Class?

A class which has the abstract keyword in its declaration is called abstract class. Abstract classes should have zero or more abstract methods. i.e., methods without a body. It can have multiple concrete methods.

Abstract classes allow you to create blueprints for concrete classes. But the inheriting class should implement the abstract method.

Abstract classes cannot be instantiated.

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* ***total*** *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.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

Note: It cannot be instantiated just like the abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- 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.

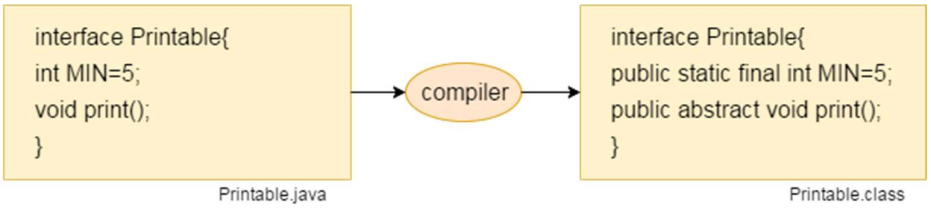
Syntax: **interface** <interface_name>{

```
// declare constant fields
// declare methods that abstract
```

```
                // by default.
                }
```

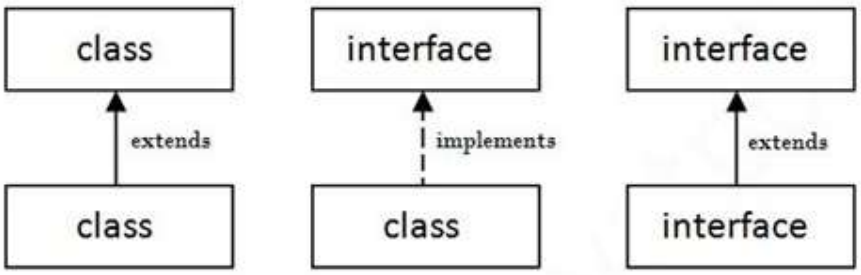
Note: The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, **Interface fields are public, static and final by default, and the methods are public and abstract.**



The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Java Interface Example

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

```
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();
 }
 }
```

Important Reasons For Using Interfaces

- Interfaces are used to achieve abstraction.
- Designed to support dynamic method resolution at run time
- It helps you to achieve loose coupling.
- Allows you to separate the definition of a method from the inheritance hierarchy

Important Reasons For Using Abstract Class

- Abstract classes offer default functionality for the subclasses.
- Provides a template for future specific classes
- Helps you to define a common interface for its subclasses
- Abstract class allows code reusability.

Packages:

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
  }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

→javac -d directory javafilename

Example: **javac -d . Simple.java**

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

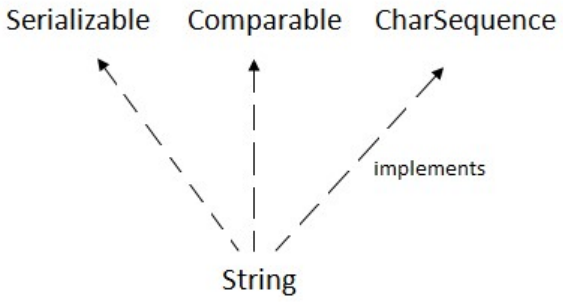
You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java **To Run:** java mypack.Simple

Strings:

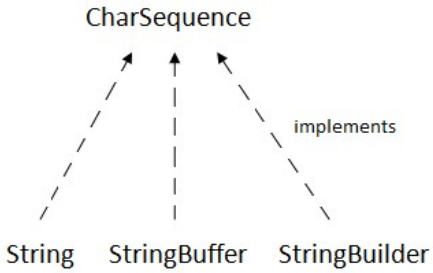
Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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



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.



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.

Subject & Code: Object Oriented Programming & 20IT303

String Methods:

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1	char charAt(int index)	It returns char value for the particular index
2	int length()	It returns string length
3	static String format(String format, Object... args)	It returns a formatted string.
4	static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
5	String substring(int beginIndex)	It returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
10	boolean equals(Object another)	It checks the equality of string with the given object.
11	boolean isEmpty()	It checks if string is empty.
12	String concat(String str)	It concatenates the specified string.
13	String replace(char old, char new)	It replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.

16	String[] split(String regex)	It returns a split string matching regex.
17	String[] split(String regex, int limit)	It returns a split string matching regex and limit.
18	String intern()	It returns an interned string.
19	int indexOf(int ch)	It returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
21	int indexOf(String substring)	It returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
23	String toLowerCase()	It returns a string in lowercase.
24	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
25	String toUpperCase()	It returns a string in uppercase.
26	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
27	String trim()	It removes beginning and ending spaces of this string.
28	static String valueOf(int value)	It converts given type into string. It is an overloaded method.

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.

Java StringBuffer Class

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

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

Important methods of StringBuffer class

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public synchronized StringBuffer	insert(int offset, String s)	It is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	It is used to return the current capacity.

public void	ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	It is used to return the character at the specified position.
public int	length()	It is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

What is a mutable String?

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

1) StringBuffer Class append() Method

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

StringBufferExample.java

```
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
}
}
```

Output: Hello Java

2) StringBuffer insert() Method

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

StringBufferExample2.java

```
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
}
}
```

Output: HJavaello

3) StringBuffer replace() Method

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

StringBufferExample3.java

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

4) StringBuffer delete() Method

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

StringBufferExample4.java

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

Output: Hlo

5) StringBuffer reverse() Method

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

StringBufferExample5.java

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

Output: olleH

6) StringBuffer capacity() Method

The capacity() method of the 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.

StringBufferExample6.java

```
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
}
}
```

Output:
```
16
16
34
```

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of the 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.

StringBufferExample7.java

```
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
        }
        }
```

Output: 16
16
34
34
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()	It creates an empty String Builder with the initial capacity of 16.
StringBuilder(String str)	It creates a String Builder with the specified string.
StringBuilder(int length)	It creates an empty String Builder with the specified capacity as length.

Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	It is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	It is used to reverse the string.
public int capacity()	It is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	It is used to return the character at the specified position.
public int length()	It is used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

Java StringBuilder Examples

Let's see the examples of different methods of StringBuilder class.

1) StringBuilder append() method

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

StringBuilderExample.java

```
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
}
}
```

Output: Hello Java

2) StringBuilder insert() method

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

StringBuilderExample2.java

```
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
}
}
```

Output: HJavaello

3) StringBuilder replace() method

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

StringBuilderExample3.java

```
class StringBuilderExample3{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.replace(1,3,"Java");
```

```
System.out.println(sb);//prints HJavalo
}
}
```

Output: HJavalo

4) StringBuilder delete() method

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

StringBuilderExample4.java

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

Output: Hlo

5) StringBuilder reverse() method

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

StringBuilderExample5.java

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

Output: olleH

6) 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.

StringBuilderExample6.java

```
class StringBuilderExample6{
```

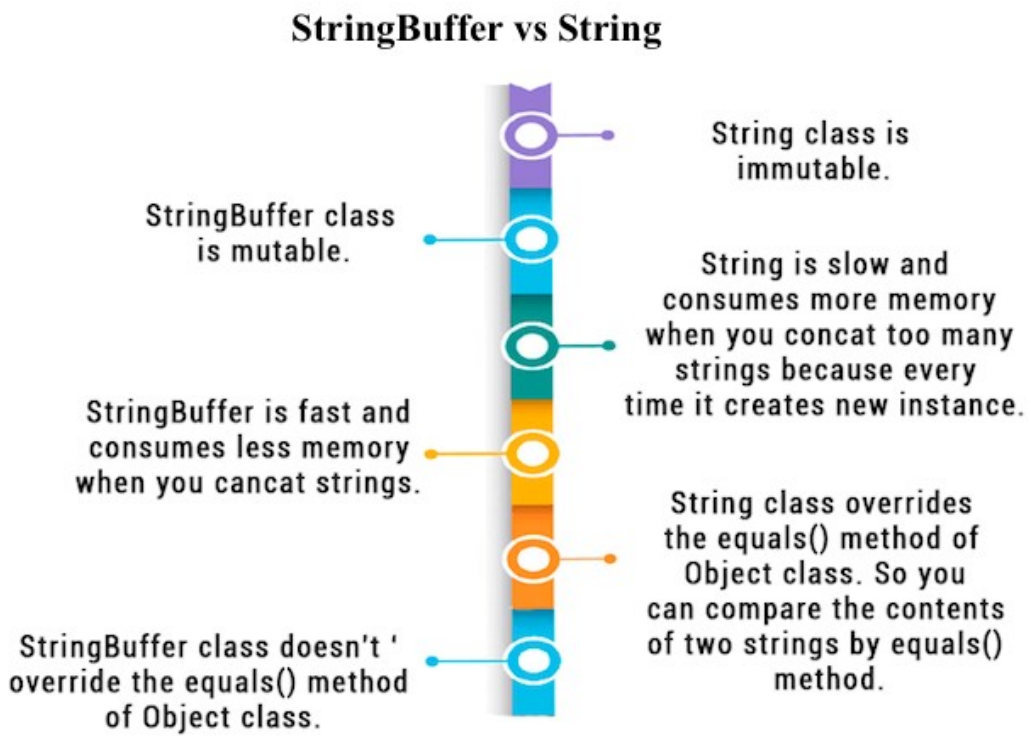
```
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("Java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2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

Output: 16
16
34

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)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t 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.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory



Performance Test of String and StringBuffer

ConcatTest.java

```
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.currentTimeMillis()-
startTime)+"ms");
        startTime = System.currentTimeMillis();
        concatWithStringBuffer();
        System.out.println("Time taken by Concating with  StringBuffer: "+(System.currentTimeMillis()-
startTime)+"ms");
    }
}
```

Output:

```
Time taken by Concating with String: 578ms
Time taken by Concating with  StringBuffer: 0ms
```

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 is 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.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Performance Test of StringBuffer and StringBuilder

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

ConcatTest.java

```
//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");
    }
}
```

Output:

```
Time taken by StringBuffer: 16ms
Time taken by StringBuilder: 0ms
```