Exception Handling

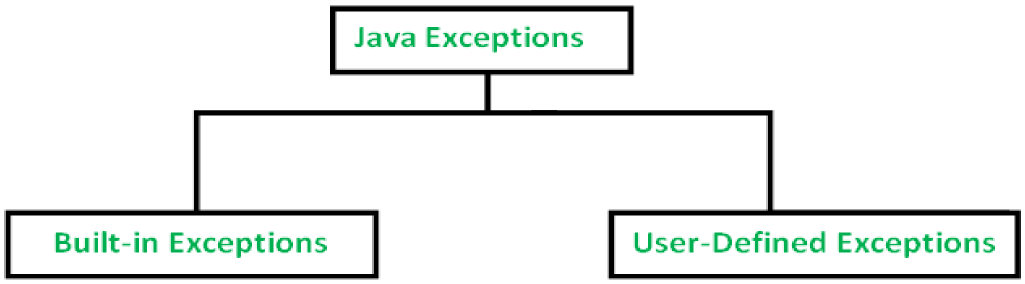
The **exception handling** isone of the powerful *mechanism in java.*

Before take a look on exception handling, let's understand what is exception.

Exception: An exception is abnormal condition(error) which occurs while execution of java program.

When an exception occurs, then an object is created and thrown by JVM.

If the object will not be handled then the application or program terminate itself suddenly.



1) Built-in Exceptions

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. **Arithmetic Exception**
   It is thrown when an exceptional condition has occurred in an arithmetic operation.
2. **ArrayIndexOutOfBoundException**
   It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3. **ClassNotFoundException**
   This Exception is raised when we try to access a class whose definition is not found
4. **FileNotFoundException**
   This Exception is raised when a file is not accessible or does not open.
5. **IOException**
   It is thrown when an input-output operation failed or interrupted
6. **InterruptedException**
   It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.
7. **NoSuchFieldException**
   It is thrown when a class does not contain the field (or variable) specified

8. **NoSuchMethodException**
   It is thrown when accessing a method which is not found.
9. **NullPointerException**
   This exception is raised when referring to the members of a null object. Null represents nothing
10. **NumberFormatException**
    This exception is raised when a method could not convert a string into a numeric format.
11. **RuntimeException**
    This represents any exception which occurs during runtime.
12. **StringIndexOutOfBoundsException**
    It is thrown by String class methods to indicate that an index is either negative than the size of the string

2)User defined/ custom exception in java

In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom** exceptions.

We will see user defined exception later, now let's see what is exception handling?

Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

The core advantage of exception handling is to maintain the normal flow of the application.

There are 5 keywords used in java to handle exception.

1. try 2.catch 3.finally 4.throw 5.throws

In Java, the code that may generate (or throw) an exception is enclosed in a try block. after try block one or more catchblocks should be Witten and after all finallyblock as the last block. The try block can also end without a catch block, and the catch blocks need not always have finally as the last block.

Each catch block specifies the type of exception it can catch and contains an exception handler code. After the last catch block, an optional finally block provides code that is always executed regardless of whether or not an exception occurs. If there are no catch blocks following a try block, the finally block is required.

Let's see about above 5 keywords used while exception handling

1) try block (try keyword)

The statements in a program that may raise exception(s) are placed within a try block.

A try block starts with the keyword try.To handle a run-time error and monitor the results, we simply enclose the code inside a try block. If an exception occurs within the try block, it is handled by the appropriate exception handler (catch block) associated with the try block.

Syntax of try block

```
try

{

  //logic of try block

}
```

Single java program may contains only one try block

2) catch block(catch keyword)

A catch block is a group of Java statements, enclosed in braces { } which are used to handle a specific exception that has been thrown. catch blocks (or handlers) should be placed after the try block. That is, a catch clause follows immediately after the try block.

When an exception occurs in a try block, program control is transferred to the appropriate catch block.

A catch block is specified by the keyword catch followed by a single argument within parentheses ( ).

```
catch (Exception_typeobj)

{

  //java statements to handle the exception Exception_1

}
```

Single java program may contains multiple catch blocks

3) finally block(finally keyword)

The code within the finally block is always executed regardless of what happens within the try block.

That is, if no exceptions are thrown, then no part of the code in the catch blocks is executed but the code in the finally block is executed.

```
finally
{
//cleanup code or exit code (optional)
}
```

4) throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or unchekedexception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
throw  new ExceptionType();
```

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
classTestThrow{

public static void main(String args[]){

try

    {

throw new ArithmeticException();

    }

catch(ArithmeticException e)

     {

System.out.println("Use of throw keyword...");

 }  } }
```

5) throws keyword:

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception

Syntax:

```
type method_name(parameters) throws exception_list
```

To prevent compile time error we can handle the exception in two ways:

1. By using try catch
2. By using **throws** keyword

Let’s see exception handling by try catch method

Examples of Built-in Exception:

- **Arithmetic exception**

```
// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
     public static void main(String args[])
    {
        try{
            int a = 30, b = 0;
            int c = a/b;  // cannot divide by zero
            System.out.println ("Result = "+ c);
        }
        catch(ArithmeticException e) {
            System.out.println ("Can't divide a number by 0");
        }
    }
}
```

•**Output:** `Can't divide a number by 0`

•ArrayIndexOutOfBounds Exception

```
// Java program to demonstrate ArrayIndexOutOfBoundException
class ArrayIndexOutOfBound_Demo
{
    public static void main(String args[])
    {
        try{
            int a[] = new int[5];
            a[6] = 9; // accessing 7th element in an array of
                      // size 5
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println ("Array Index is Out Of Bounds");
        }
    }
}
```

Output: `Array Index is Out Of Bounds`

•NullPointer Exception

```
//Java program to demonstrate NullPointerException
class NullPointer_Demo
{
    public static void main(String args[])
    {
        try{
            String a = null; //null value
            System.out.println(a.charAt(0));
        } catch(NullPointerException e) {
            System.out.println("NullPointerException..");
        }
    }
}
```

•**Output:** `NullPointerException..`

•StringIndexOutOfBound Exception

```
// Java program to demonstrate StringIndexOutOfBoundsException
Class StringIndexOutOfBound_Demo
{
    public static void main(String args[])
    {
        try{
            String a = "This is like chipping "; // length is 22
            charc = a.charAt(24); // accessing 25th element
            System.out.println(c);
        }
        catch(StringIndexOutOfBoundsException e) {
            System.out.println("StringIndexOutOfBoundsException");
        }
    }
}
```

•Output: `StringIndexOutOfBoundsException`

•FileNotFound Exception

```
//Java program to demonstrate FileNotFoundException
importjava.io.File;
importjava.io.FileNotFoundException;
importjava.io.FileReader;
 class File_notFound_Demo {

    public static void main(String args[])  {
        try{

            // Following file does not exist
            File file = newFile("E://file.txt");

            FileReader fr = new FileReader(file);
        } catch(FileNotFoundException e) {
           System.out.println("File does not exist");
        }
    }
}
```

•Output: `File does not exist`

•NumberFormat Exception

```
// Java program to demonstrate NumberFormatException
class NumberFormat_Demo
{
    publicstaticvoidmain(String args[])
    {
        try{
            // "akk" is not a number
            Int num = Integer.parseInt("akk") ;

            System.out.println(num);
        } catch(NumberFormatException e) {
            System.out.println("Number format exception");
        }
    } }
```

•**Output:** `Number format exception`

And the another way to handle the exception is, using throws keyword.

We can use throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

```
classtst
{
   public static void main(String[] args)throws InterruptedException
    {
        Thread.sleep(10000);
        System.out.println("Hello Geeks");
    }
}
```

Some Programs on Exception handling.

* Nested try block: a try block within a try block is called as nested try block

```
 class NestedTry
        {
       Public static void main(String args[])
         {
        try
          {
              try
               {
               throw new ArithmeticException();
               }
               catch(ArithmeticException a)
                {
                System.out.println("Inner try block exception caught");
                }
            throw Exception();
          }
          catch(Exception e)
                {
                System.out.println("outer try block exception caught");
                }

       }

      }
```

* program for single try block and multiple catch blocks

```
       class MCatch
        {
       Public static void main(String args[])
         {
            int a[]=new int[3];
        try
          {
            a[5]=99/0;
             }
                catch(ArithmeticException a)   //catch block1
                 {
                 System.out.println("can't divide by zero");
                 }
                catch(ArrayIndexOutOfBoundsException e) //catch block2
                 {
                 System.out.println("you are trying to access an invalid index of array");
                 }

          }

        }
```

We already seen that what is user defined exception, here a recap on that and example of user

Defined exception and handling of user defined exception.

2)User defined/ custom exception in java

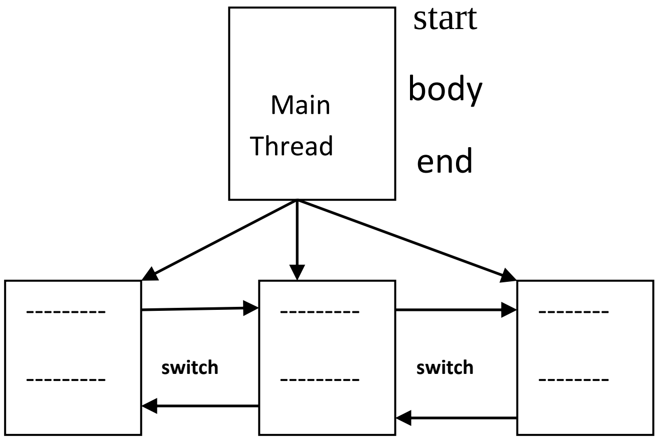
In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom** exceptions.

Example of User defined exception in Java

```
class MyException extends Exception{

public String toString(){
        return ("MyException Occurred: ") ;
   }
}

class Example1{
public static void main(String args[]){
        try{
                System.out.println("Starting of try block");
                // I'm throwing the custom exception using throw
                throw new MyException();
        }
        catch(MyException exp){
                System.out.println("Catch Block") ;
                System.out.println(exp) ;
        }
   }
}
```

Multithreading

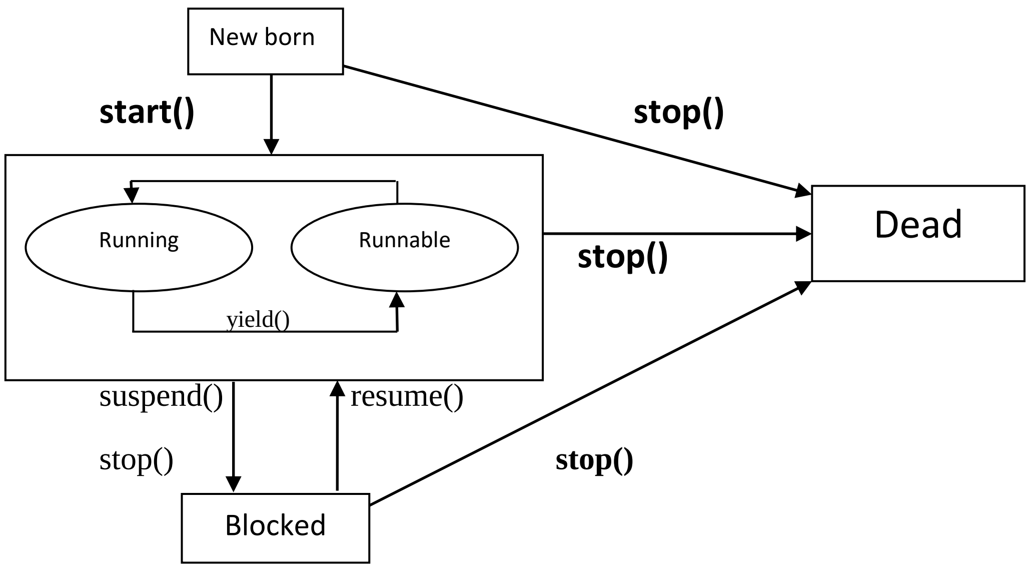
Before we talk about **multithreading**, let's discuss threads.
Thread: A thread is a light-weight smallest part of a process that can run concurrently with the other parts(other threads) of the same process. Threads are independent because they all have separate path of execution that's the reason if an exception occurs in one thread, it doesn't affect the execution of other threads. All threads of a process share the common memory.
Multithreading: The process of executing multiple threads simultaneously is known as multithreading.
In Multithreading process is divided into number of Processes, each process is called as thread, so these multiple threads executes simultaneously.
Each thread have it's beginning, body and end.



Advantages of Java Multithreading

1) It doesn't block the user because threads are independent and you can perform multiple operations at same time.
2) You can perform many operations together so it saves time.
3) Threads are independent so it doesn't affect other threads if exception occur in a single thread.

Thread Life Cycle:



1) **Newborn state**: When a thread is currently create it's object, at that time thread comes in Newbornstate.It remains in this state until the program starts the thread. It is also referred to as a born thread.

A thread starts it's execution from Newborn state by start() method.

Also forcefully terminated by stop() method after that it goes to **dead** state

2) **Runnable** :After a newly born thread is started, the thread comes in runnable state.

A threadin this state is considered to be executing its task.

We can forcefully terminate a thread by stop() method after that it goes to **dead** state.

3) **Runnig** : When a thread comes under running state it starts it's execution.

We can use yield() method to pause that thread for specific time after that it goes in**runnable** state.

After use of sleep() and suspend() methods thread goes in **blocked** state.

4) **Blocked state** : When a tread is suspended or in sleep mode at that time it comes under Blocked state.

By resume() method a slept thread can be resumed.

By stop() method a thread can be sent to **dead** state

5) **Dead state** : Every thread have a dead state where a thread destroyed forever.

Threads can be created by using two mechanisms :
1. Extending the Thread class
2. Implementing the Runnable Interface

1) Thread creation by extending the Thread class
We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

```
classMultithreadingDemo extends Thread
{
public void run()
   {
try
        {
            // Displaying the thread that is running
System.out.println ("Thread " +Thread.currentThread().getId() + " is running");
```

```
        }
catch (Exception e)
        {
            // Throwing an exception
System.out.println ("Exception is caught");
        }
    }
}
public class Multithread
{
public static void main(String[] args)
    {
MultithreadingDemo object1 = new MultithreadingDemo();
MultithreadingDemo object2 = new MultithreadingDemo();
MultithreadingDemo object3= new MultithreadingDemo();
MultithreadingDemo object4 = new MultithreadingDemo();
object1.start();
object2.start();
object3.start();
object4.start();
    }
}
```

1. **void run():** is used to perform action for a thread.
2. **void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **void join():** waits for a thread to die.
5. **void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **intgetPriority():** returns the priority of the thread.
7. **intsetPriority(int priority):** changes the priority of the thread.
8. **String getName():** returns the name of the thread.
9. **voidsetName(String name):** changes the name of the thread.
10. **Thread currentThread():** returns the reference of currently executing thread.
11. **intgetId():** returns the id of the thread.
12. **Thread.StategetState():** returns the state of the thread.
13. **booleanisAlive():** tests if the thread is alive.
14. **void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **void suspend():** is used to suspend the thread(depricated).

16. **void resume():** is used to resume the suspended thread(depricated).
17. **void stop():** is used to stop the thread(depricated).
18. **booleanisDaemon():** tests if the thread is a daemon thread.
19. **voidsetDaemon(boolean b):** marks the thread as daemon or user thread.
20. **void interrupt():** interrupts the thread.
21. **booleanisInterrupted():** tests if the thread has been interrupted.

2)Thread creation by implementing runnable interface

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

```
importjava.lang.*;
class MultithreadingDemo implements Runnable
{
public void run()
   {
try
       {
System.out.println ("Thread " + Thread.currentThread().getId() +" is running");
        }
catch (Exception e)
       {
System.out.println ("Exception is caught");
       }
   }
}

class MultithreadImpl
{
public static void main(String[] args)
   {
MultithreadingDemo object1=new MultithreadingDemo();
MultithreadingDemo object2=new MultithreadingDemo();
MultithreadingDemo object3=new MultithreadingDemo();

Thread r1=new Thread(object1);
Thread r2=new Thread(object2);
Thread r3=new Thread(object3);

r1.start();
r2.start();
r3.start();
}
}
```

If we are not extending the Thread class ,our class object would not be treated as a thread object.So we need to explicitely create Thread class object.We are passing the object of our class that implements Runnable so that our class run() method may execute.

Synchronization in Java

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.

So the solution is Synchronization.

Synchronization is a mechanism by which only one thread can access the resource at a given point of time.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

- Mutual Exclusive

1. Synchronized method.
2. Synchronized block.
3. static synchronization.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

```
class Table{
void printTable(int n){//method not synchronized
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
```

```
}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class Test{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
```

}**<u>Solution by synchronized method</u>**

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the method returns.

```
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Output:

```
10
15
20
25
100
200
300
400
500
5
10
15
20
25
100
200
300
400
500
```

2) Synchronized block

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.

- Scope of synchronized block is smaller than the method.

Syntax to use synchronized block

```
synchronized (object reference expression)
{
      //code block
      }
```

Example of synchronized block

Let's see the simple example of synchronized block.

```
class Table{
 void printTable(int n){
synchronized(this)
{
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }
 }
 }
}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

3) Static synchronization

Static synchronized methods synchronize on the class object. If one thread is executing a static synchronized method, all other threads trying to execute any static synchronized methods will be blocked

Example of static synchronization

In this example we are applying synchronized keyword on the static method to perform static synchronization.

```
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
} }
```

Inter-thread communication in Java

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

---

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Descriptions
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

---

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

public final void notify()

---

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

Difference between wait and sleep?

Let's see the important differences between wait and sleep methods.

Wait() method	Sleep () method
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	Is the method of Thread class
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

Example of inter thread communication in java

Let's see the simple example of inter thread communication.

```
class Customer{
int amount=10000;

synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
   }
this.amount-=amount;
System.out.println("withdraw completed...");
   }
synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
   }
   }
class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
   }.start();
new Thread(){
public void run(){c.deposit(10000);}
   }.start();
   }}
```

The join() method

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

```
Syntax:
public void join()throws InterruptedException
public void join(long milliseconds)throws InterruptedException
```

Example of join() method

```
class TestJoinMethod1 extends Thread{
    public void run(){
     for(inti=1;i<=5;i++){
      try{
Thread.sleep(500);
      }catch(Exception e){System.out.println(e);}
System.out.println(i);
     }
    }
   public static void main(String args[]){
    TestJoinMethod1 t1=new TestJoinMethod1();
    TestJoinMethod1 t2=new TestJoinMethod1();
    TestJoinMethod1 t3=new TestJoinMethod1();
    t1.start();
    try{
     t1.join();
    }catch(Exception e){System.out.println(e);}

    t2.start();
    t3.start();
    }
   }
```

As you can see in the above example,when t1 completes its task then t2 and t3 starts executing.

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defiend in Thread class:

```
public static int MIN_PRIORITY
public static int NORM_PRIORITY
public static int MAX_PRIORITY
```

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread
{
public void run()
 {
  System.out.println("running thread name     is:"+Thread.currentThread().getName());
  System.out.println("running thread priority
  is:"+Thread.currentThread().getPriority());
  }
public static void main(String args[])
{
 TestMultiPriority1 m1=new TestMultiPriority1();
 TestMultiPriority1 m2=new TestMultiPriority1();
 m1.setPriority(Thread.MIN_PRIORITY);
 m2.setPriority(Thread.MAX_PRIORITY);
 m1.start();
 m2.start();

}      }
```

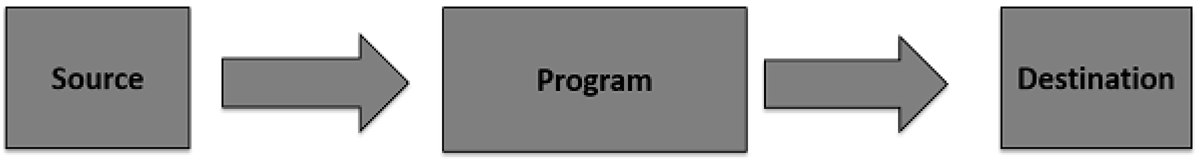
Input / Output Handling

The java.io package contains nearly every class we might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.
Java Program performs input / output through stream. So let's understand stream.

Stream

A stream can be defined as a sequence of byts(data) which flows from input device to output device and vice-versa.
There are two kinds of Streams −

- **InPutStream** − The InputStream is used to read data from a source.
- **OutPutStream** − The OutputStream is used for writing data to a destination.



Java provides strong but flexible support for I/O related to files and networks
Both Input stream and Output stream classified into two parts
1) Byte Stream (Binary Straem)
2) Character Stream (Text stream)

Byte Streams(Binary streams)

Java byte streams are used to perform input and output of bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

Input stream classes

Table Input Byte Stream Classes

Class	Description
BufferedInputStream	contains methods to read bytes from the buffer (memory area)
ByteArrayInputStream	contains methods to read bytes from a byte array
DataInputStream	contains methods to read Java primitive data types
FileInputStream	contains methods to read bytes from a file

FilterInputStream	contains methods to read bytes from other input streams which it uses as its basic source of data
ObjectInputStream	contains methods to read objects
PipedInputStream	contains methods to read from a piped output stream. A piped input stream must be connected to a piped output stream
SequenceInputStream	contains methods to concatenate multiple input streams and then read from the combined stream

The Input Stream class defines various methods to perform reading operations on data of an input stream. Some of these methods along with their description are listed in Table

Table InputStream Class Methods

Method	**Description**
int read()	returns the integral representation of the next available byte of input. It returns -1 when end of file is encountered
int read (byte buffer [])	attempts to read buffer. length bytes into the buffer and returns the total number of bytes successfully read. It returns -1 when end of file is encountered
int read (byte buffer [], intloc, intnBytes)	attempts to read 'nBytes' bytes into the buffer starting at buffer [loc] and returns the total number of bytes successfully read. It returns -1 when end of file is encountered
int available ()	returns the number of bytes of the input available for reading
Void mark(intnBytes)	marks the current position in the input stream until 'nBytes' bytes are read
void reset ()	Resets the input pointer to the previously set mark
long skip (long nBytes)	skips 'nBytes' bytes of the input stream and returns the number of actually skippedbyte
void close ()	closes the input source. If an attempt is made to read even after closing the stream then it generates IOException

Output ByteStream classes

Java's output **Byte**stream classes are used to write 8-bit bytes to a stream. The OutputStream class is the superclass for all byte-oriented output stream classes. All the methods of this class throw an IOException. Being an abstract class, the OutputStream class cannot be instantiated hence, its subclasses are used. Some of these are listed in Table

Table Output Stream Classes

Class	**Description**
BufferedOutputStream	Contains methods to write bytes into the buffer
ByteArrayOutputStream	Contains methods to write bytes into a byte array
DataOutputStream	Contains methods to write Java primitive data types
FileOutputStream	Contains methods to write bytes to a file
FilterOutputStream	Contains methods to write to other output streams
ObjectOutputStream	Contains methods to write objects
PipedOutputStream	Contains methods to write to a piped output stream
PrintStream	Contains methods to print Java primitive data types

The OutputStream class defines methods to perform writing operations. These methods are discussed in Table

TableOutputStream Class Methods

.Method	**Description**
void write (int i)	writes a single byte to the output stream
void write (byte buffer [])	writes an array of bytes to the output stream
Void write(bytes buffer[],intloc, intnBytes)	writes 'nBytes' bytes to the output stream from the buffer b starting at buffer [loc]
void flush ()	Flushes the output stream and writes the waiting buffered output bytes
void close ()	closes the output stream. If an attempt is made to write even after closing the stream then it generates IOException

Character Streams(Text Streams)

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses

FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

Input Character classes(Character Reader classes)

Character Reader classes are used to read 16-bit unicode characters from the input stream. The Reader class is the superclass for all character-oriented input stream classes. All the methods of this class throw an IOException. Being an abstract class, the Reader class cannot be instantiated hence its subclasses are used. Some of these are listed in Table

TableCharacter Reader Classes

Class	**Description**
BufferedReader	contains methods to read characters from the buffer
CharArrayReader	contains methods to read characters from a character array
FileReader	contains methods to read from a file
FilterReader	contains methods to read from underlying character-input stream
InputStreamReader	contains methods to convert bytes to characters
PipedReader	contains methods to read from the connected piped output stream
StringReader	contains methods to read from a string

The Reader class defines various methods to perform reading operations on data of an input stream. Some of these methods along with their description are listed in Table.

Table Reader Class Methods

Method	**Description**
int read()	returns the integral representation of the next available character of input. It returns -1 when end of file is encountered
int read (char buffer [])	attempts to read buffer. length characters into the buffer and returns the total number of characters successfully read. It returns -I when end of file is encountered
int read (char buffer [], intloc, intnChars)	attempts to read 'nChars' characters into the buffer starting at buffer [loc] and returns the total number of characters successfully read. It returns -1 when end of file is encountered

void mark(intnChars)	marks the current position in the input stream until 'nChars' characters are read
void reset ()	resets the input pointer to the previously set mark
long skip (long nChars)	skips 'nChars' characters of the input stream and returns the number of actually skipped characters
boolean ready ()	returns true if the next request of the input will not have to wait, else it returns false
void close ()	closes the input source. If an attempt is made to read even after closing the stream then it generates IOException

Character Output classes (Character Writer Classes)

Character Writer classes are used to write 16-bit Unicode characters onto an outputstream. The Writer class is the superclass for all character-oriented output stream classes .All the methods of this class throw

anIOException. Being an abstract class, the Writer class cannot be instantiated hence, its subclasses are used. Some of these are listed in Table.

Table Writer Classes

Class	Description
BufferedWriter	Contains methods to write characters to a buffer
FileWriter	Contains methods to write to a file
FilterWriter	Contains methods to write characters to underlying output stream
CharArrayWriter	Contains methods to write characters to a character array
OutputStreamWriter	Contains methods to convert from bytes to character
PipedWriter	Contains methods to write to the connected piped input stream
StringWriter	Contains methods to write to a string

The Writer class defines various methods to perform writing operations on output stream. These methods along with their description are listed in Table

Table Writer Class Methods

Method	Description
void write ()	writes data to the output stream

void write (int i)	Writes a single character to the output stream
void write (char buffer [])	writes an array of characters to the output stream
void write(char buffer [],intloc, intnChars)	writes 'n' characters from the buffer starting at buffer [loc] to the output stream
void close ()	closes the output stream. If an attempt is made to perform writing operation even after closing the stream then it generates IOException
void flush ()	flushes the output stream and writes the waiting buffered output characters

File : File is a place on secondary storage where data can be stored permanently.
Java provides some methods to read/ write the data from/ on file.
In java all file data is byte oriented, java allows us to wrap byte oriented file stream within Character based objects, using character stream object such as InputStreamReader.
Files are of two types as,

1) Binary files
2) Text Files

1) Binary Files : The files which operates i.e read or write binary stream(8 bit data) those files are called Binary Files. Binary files used to operate binary data such as images, videos etc.
2) Text files: The files which operates text streams (16 bit Unicode data) those files are called Text files.

a)To Perform operation on file first we have to open the file by using syntax
Fileclassname obj=new fileclassname("filename with path");
b)After opening of file we are able to perform operations on file.
c)And after all we have to close the file by syntax
Obj.close();

Writing Binary Files in Java

You can create and write to a binary file for that we need **FileOutputStream.**

```
import java.io.FileOutputStream;

class FileOutputStreamExample {

public static void main(String args[]){
```

```
try{

FileOutputStream fout=new FileOutputStream("F:\\testout.txt");

               String s="sangola";

byte b[]=s.getBytes();//converting string into byte array

fout.write(b);

fout.close();

System.out.println("success...");

}catch(Exception e){System.out.println(e);}

         }

    }
```

Reading Binary Files in Java

If you want to read a binary file, or a text file containing 'weird' characters (ones that your system doesn't deal with by default), you need to use **FileInputStream**.FileInputStream defines a method called **read()** to read the data.

```
 import java.io.FileInputStream;

class DataStreamExample {

public static void main(String args[]){

try{

FileInputStream fin=new FileInputStream("F:\\testout.txt");

int i=0;

while((i=fin.read())!=-1){

System.out.print((char)i);

           }

fin.close();

}catch(Exception e){System.out.println(e);}

        }

       }
```

Writing Text Files in Java

To write a text file in Java, we can use **FileWriter** with **BufferedOutputWriter** class

In which **FileWriter** object wrapped in **BufferedOutputWriter.**

```
import java.io.*;

class FileWriterExample {

public static void main(String args[]){

    String str="hello";

try{

FileWriter fw=new FileWriter("F:\\buf.txt");

BufferedWriter b=new BufferedWriter(fw);

b.write(str);

b.close();

}catch(Exception e){System.out.println(e);}

System.out.println("Success...");

}   }
```

Reading Ordinary Text Files in Java

If you want to read an ordinary text file in your system's default encoding (usually the case most of the time for most people), use **FileReader** and wrap it in a **BufferedReader**

```
import java.io.*;
public class TextFileReaderExample {
public static void main(String args[])throws Exception{
FileReader fr=new FileReader("F:\\testout.txt");
BufferedReader b=new BufferedReader(fr);
int i;
while((i=b.read())!=-1)
System.out.print((char)i);
b.close();
   }   }
```

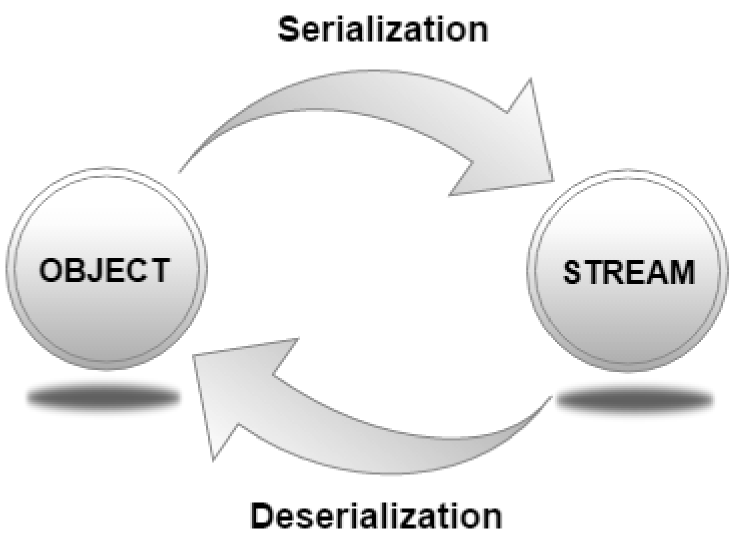
Object Streams and serializable Interface

Object Streams are the streams where bytestreams can be stored which are obtained after object state conversion.

Serializable interface (Serialization in Java)

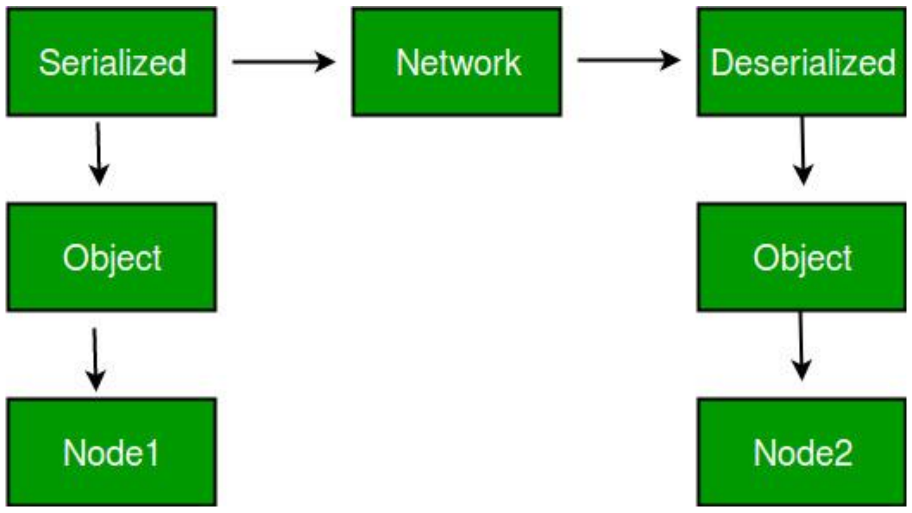
Serialization is a mechanism of converting the state of an object into a byte stream.
Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.
To do so **Serializable interface** is used.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.



Serializable interface: It is an interface which is predefined in **java.io** package.
To make a Java object serializable we implement the **java.io.Serializable** interface.
The ObjectOutputStream class contains **writeObject()** method for serializing an Object.
The ObjectInputStream class contains **readObject()** method for deserializing an object.

Advantages of Serialization
1. To save/persist state of an object.
2. To travel an object across a network.



Only the objects of those classes can be serialized which are implementing **java.io.Serializable** interface.
Serializable is a **marker interface** (has no data member and method). It is used to "mark" java classes so that objects of these classes may get certain capability. Other examples of marker interfaces are:- Cloneable and Remote.

Points to remember
1. If a parent class has implemented Serializable interface then child class doesn't need to implement it but vice-versa is not true.
2. Only non-static data members are saved via Serialization process.
3. Static data members and transient data members are not saved via Serialization process.So, if you don't want to save value of a non-static data member then make it transient.
4. Constructor of object is never called when an object is deserialized.
5. Associated objects must be implementing Serializable interface.

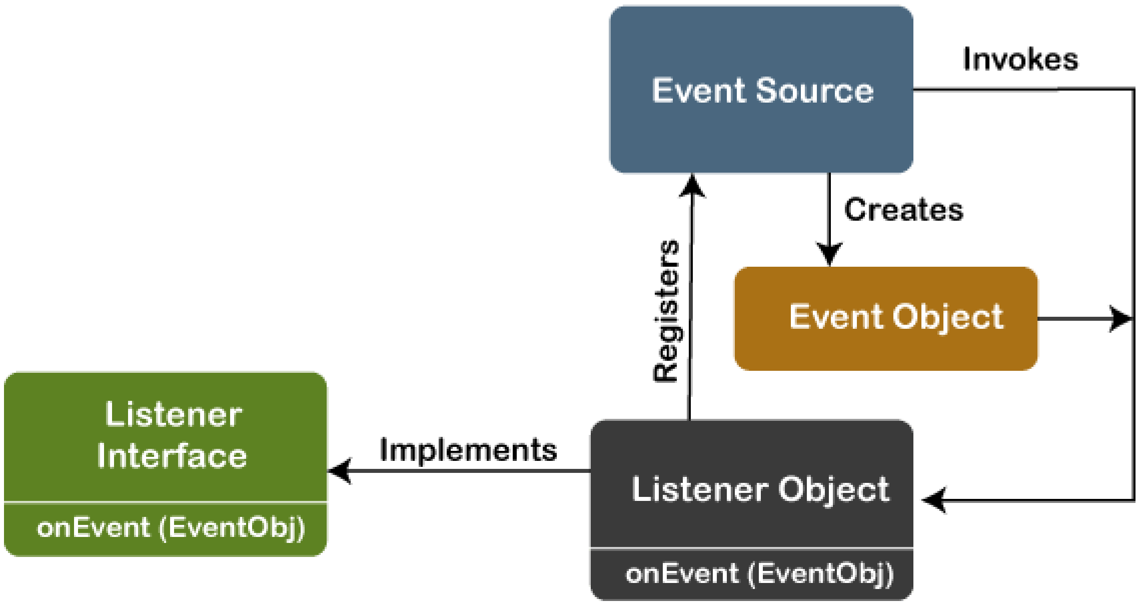
```
import java.io.*;

class Demo implements Serializable
{
    public int a;
     public String b;

    // Default constructor
    public Demo(int a, String b)
    {
        this.a = a;
        this.b = b;
    }

}

 class Test
```

```
{
    public static void main(String[] args)
    {
        Demo object = new Demo(1, "xyz");
        String filename = "file.ser";

        // Serialization
        try
        {
            //Saving of object in a file
            FileOutputStream file = new FileOutputStream(filename);
            ObjectOutputStream out = new ObjectOutputStream(file);

            // Method for serialization of object
            out.writeObject(object);

            out.close();
            file.close();

            System.out.println("Object has been serialized");

        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }


        Demo object1 = null;

        // Deserialization
        try
        {
            // Reading the object from a file
            FileInputStream file = newFileInputStream(filename);
            ObjectInputStream in = newObjectInputStream(file);

            // Method for deserialization of object
            object1 = (Demo)in.readObject();

            in.close();
            file.close();

            System.out.println("Object has been deserialized ");
            System.out.println("a = "+ object1.a);
            System.out.println("b = "+ object1.b);
        }

        catch(IOException ex)
        {
            System.out.println("IOException is caught");
        }

        catch(ClassNotFoundException ex)
        {
            System.out.println("ClassNotFoundException is caught");
        }

    }
}
```

Event and Listener (Java Event Handling)

Event model-It is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.



The modern approach for event processing is based on the Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Event Model is that the application logic is completely separated from the interface logic.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

Basically, an Event Model is based on the following three components:

1) Events 2) Events Sources 3) Events Listeners

1) Events:

The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements. Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on.

2) Events Sources

A source is an object that causes and generates an event. It generates an event when the internal state of the object is changed. The sources are allowed to generate several different types of events.

A source must register a listener to receive notifications for a specific event. Each event contains its registration method. Below is an example:

```
public void addTypeListener (TypeListener e1)
```

3) Events Listeners
An event listener is an object that is invoked when an event triggers. The listeners require two things; first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events. Second, it must implement the methods to receive and process the received notifications.

The events are categories into the following two categories:

The Foreground Events:

The foreground events are those events that require direct interaction of the user. These types of events are generated as a result of user interaction with the GUI component. For example, clicking on a button, mouse movement, pressing a keyboard key, selecting an option from the list, etc.

The Background Events :

The Background events are those events that result from the interaction of the end-user. For example, an Operating system interrupts system failure (Hardware or Software).

The java.awt.event package provides many event classes and Listener interfaces for event handling.
Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling
Following steps are required to perform event handling:

1) Register the component with the Listener
Registration Methods
For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}

- **Checkbox**
    - public void addItemListener(ItemListener a){}
- **Choice**
    - public void addItemListener(ItemListener a){}
- **List**
    - public void addActionListener(ActionListener a){}
    - public void addItemListener(ItemListener a){}

Java Event Handling Code
We can put the event handling code into one of the following places:

1) Within class
2) Other class
3) Anonymous class

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class AEvent extends Frame implements ActionListener{
JTextField tf;
AEvent(){

//create components
tf=new JTextField();
tf.setBounds(60,50,170,20);
JButton b=new JButton("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}  }
```

2) Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class AEvent2 extends Frame{
JTextField tf;
AEvent2(){
//create components
tf=new JTextField();
tf.setBounds(60,50,170,20);
JButton b=new JButton("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}
```

3) Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class AEvent3 extends Frame{
JTextField tf;
AEvent3(){
tf=new JTextField();
tf.setBounds(60,50,170,20);
JButton b=new JButton("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent3();
}
}
```

Java Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components.
The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

What is JFC

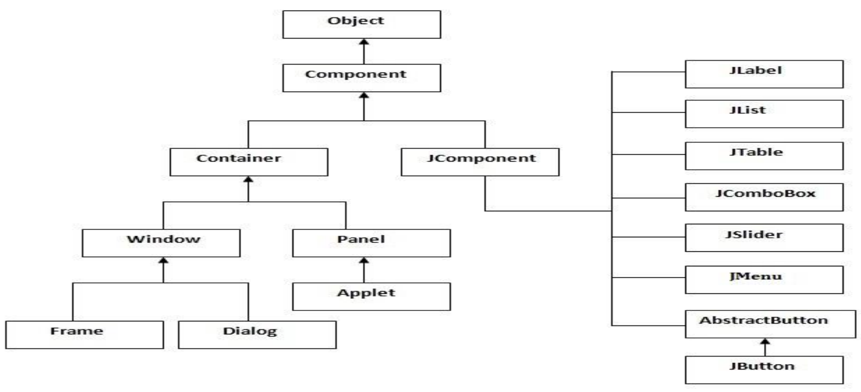
The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.
The hierarchy of java swing API is given below.

Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Hierarchy of Java Swing classes



Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
import javax.swing.*;
    public class FirstSwingExample {
    public static void main(String[] args) {
    JFrame f=new JFrame();//creating instance of JFrame
    JButton b=new JButton("click");//creating instance of JButton
    b.setBounds(130,100,100, 40);//x axis, y axis, width, height
    f.add(b);//adding button in JFrame
    f.setSize(400,500);//400 width and 500 height
    f.setLayout(null);//using no layout managers
    f.setVisible(true);//making the frame visible
    }  }
```

Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

```
File: Simple.java
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
public static void main(String[] args) {
new Simple();
}  }
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

```
public class JButton extends AbstractButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
   JFrame f=new JFrame("Button Example");
   final JTextField tf=new JTextField();
   tf.setBounds(50,50, 150,20);
   JButton b=new JButton("Click Here");
   b.setBounds(50,100,95,30);
   b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
         tf.setText("Welcome to Sangola.");
      }
   });
   f.add(b);f.add(tf);
   f.setSize(400,400);
```

```
        f.setLayout(null);
        f.setVisible(true);
    }
    }
```

Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

```
public class JTextField extends JTextComponent implements SwingConstants
```

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

Methods	Description
void addActionListener(ActionListener	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

```
      import javax.swing.*;
      import java.awt.event.*;
    public class TextFieldExample implements ActionListener{
        JTextField tf1,tf2,tf3;
        JButton b1,b2;
        TextFieldExample(){
            JFrame f= new JFrame();
            tf1=new JTextField();
            tf1.setBounds(50,50,150,20);
            tf2=new JTextField();
            tf2.setBounds(50,100,150,20);
            tf3=new JTextField();
            tf3.setBounds(50,150,150,20);
            tf3.setEditable(false);
            b1=new JButton("+");
            b1.setBounds(50,200,50,50);
            b2=new JButton("-");
            b2.setBounds(120,200,50,50);
            b1.addActionListener(this);
            b2.addActionListener(this);
            f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
            f.setSize(300,300);
            f.setLayout(null);
            f.setVisible(true);
        }
        public void actionPerformed(ActionEvent e) {
            String s1=tf1.getText();
            String s2=tf2.getText();
            int a=Integer.parseInt(s1);
            int b=Integer.parseInt(s2);
            int c=0;
            if(e.getSource()==b1){
                c=a+b;
            }else if(e.getSource()==b2){
                c=a-b;
            }
            String result=String.valueOf(c);
            tf3.setText(result);
        }
    public static void main(String[] args) {
        new TextFieldExample();
    } }
```

Java JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

```
public class JTextArea extends JTextComponent
```

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener{
JLabel l1,l2;
JTextArea area;
JButton b;
TextAreaExample() {
    JFrame f= new JFrame();
    l1=new JLabel();
    l1.setBounds(50,25,100,30);
    l2=new JLabel();
    l2.setBounds(160,25,100,30);
    area=new JTextArea();
    area.setBounds(20,75,250,200);
    b=new JButton("Count Words");
    b.setBounds(100,300,120,30);
    b.addActionListener(this);
    f.add(l1);f.add(l2);f.add(area);f.add(b);
    f.setSize(450,450);
    f.setLayout(null);
    f.setVisible(true);
}
public void actionPerformed(ActionEvent e){
    String text=area.getText();
    String words[]=text.split("\\s");
    l1.setText("Words: "+words.length);
    l2.setText("Characters: "+text.length());
}
public static void main(String[] args) {
    new TextAreaExample();
}
}
```

Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

```
public class JCheckBox extends JToggleButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

```
import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample
{
    CheckBoxExample(){
      JFrame f= new JFrame("CheckBox Example");
      final JLabel label = new JLabel();
      label.setHorizontalAlignment(JLabel.CENTER);
      label.setSize(400,100);
      JCheckBox checkbox1 = new JCheckBox("C++");
      checkbox1.setBounds(150,100, 50,50);
      JCheckBox checkbox2 = new JCheckBox("Java");
      checkbox2.setBounds(150,150, 50,50);
      f.add(checkbox1); f.add(checkbox2); f.add(label);
      checkbox1.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
            label.setText("C++ Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
          }
       });
      checkbox2.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
            label.setText("Java Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
          }
       });
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckBoxExample();
}
}
```

Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

```
public class JRadioButton extends JToggleButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Example.

```
import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener{
JRadioButton rb1,rb2;
JButton b;
RadioButtonExample(){
rb1=new JRadioButton("Male");
rb1.setBounds(100,50,100,30);
rb2=new JRadioButton("Female");
rb2.setBounds(100,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(rb1);bg.add(rb2);
b=new JButton("click");
b.setBounds(100,150,80,30);
b.addActionListener(this);
add(rb1);add(rb2);add(b);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are Male.");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are Female.");
}
}
public static void main(String args[]){
```

```
new RadioButtonExample();
}}
```

Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

```
public class JComboBox extends JComponent implements ItemSelectable, ListDataListener, ActionListener, Accessible
```

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Commonly used Methods:

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Java JComboBox Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
    f=new JFrame("ComboBox Example");
    final JLabel label = new JLabel();
    label.setHorizontalAlignment(JLabel.CENTER);
    label.setSize(400,100);
    JButton b=new JButton("Show");
    b.setBounds(200,100,75,20);
    String languages[]={"C","C++","C#","Java","PHP"};
    final JComboBox cb=new JComboBox(languages);
    cb.setBounds(50, 100,90,20);
    f.add(cb); f.add(label); f.add(b);
    f.setLayout(null);
    f.setSize(350,350);
    f.setVisible(true);
    b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
String data = "Programming language Selected: "
  + cb.getItemAt(cb.getSelectedIndex());
label.setText(data);
}
});
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}
```

Java JOptionPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

Common Constructors of JOptionPane class

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Common Methods of JOptionPane class

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.

void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Sangola.");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOptionPane.WA
RNING_MESSAGE);
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

Java JOptionPane Example: showInputDialog()

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    String name=JOptionPane.showInputDialog(f,"Enter Name");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

Java JOptionPane Example: showConfirmDialog()

```
import javax.swing.*;
import java.awt.event.*;
```

```
public class OptionPaneExample extends WindowAdapter{
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   f.addWindowListener(this);
   f.setSize(300, 300);
   f.setLayout(null);
   f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
   f.setVisible(true);
}
public void windowClosing(WindowEvent e) {
   int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
if(a==JOptionPane.YES_OPTION){
   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
public static void main(String[] args) {
   new  OptionPaneExample();
}
}
```

Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

public class JMenuBar extends JComponent implements MenuElement, Accessible

JMenu class declaration

public class JMenu extends JMenuItem implements MenuElement, Accessible

JMenuItem class declaration

public class JMenuItem extends AbstractButton implements Accessible, MenuElement

Java JMenuItem and JMenu Example

```
import javax.swing.*;
class MenuExample
{
        JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1); menu.add(i2); menu.add(i3);
        submenu.add(i4); submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
    new MenuExample();  }}
```

Example of creating Edit menu for Notepad:

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample implements ActionListener{
JFrame f;
JMenuBar mb;
JMenu file,edit,help;
JMenuItem cut,copy,paste,selectAll;
JTextArea ta;
MenuExample(){
f=new JFrame();
cut=new JMenuItem("cut");
copy=new JMenuItem("copy");
paste=new JMenuItem("paste");
selectAll=new JMenuItem("selectAll");
cut.addActionListener(this);
copy.addActionListener(this);
paste.addActionListener(this);
selectAll.addActionListener(this);
mb=new JMenuBar();
file=new JMenu("File");
edit=new JMenu("Edit");
help=new JMenu("Help");
edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
mb.add(file);mb.add(edit);mb.add(help);
ta=new JTextArea();
ta.setBounds(5,5,360,320);
f.add(mb);f.add(ta);
f.setJMenuBar(mb);
f.setLayout(null);
f.setSize(400,400);
f.setVisible(true);
}
public void actionPerformed(ActionEvent e) {
if(e.getSource()==cut)
ta.cut();
if(e.getSource()==paste)
ta.paste();
if(e.getSource()==copy)
ta.copy();
if(e.getSource()==selectAll)
ta.selectAll();
}
public static void main(String[] args) {
    new MenuExample();
}}
```

Java JTree

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

Java JTree Example

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample {
JFrame f;
TreeExample(){
    f=new JFrame();
    DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
    DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
    DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
    style.add(color);
    style.add(font);
    DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
    DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
    DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");
    DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");
    color.add(red); color.add(blue); color.add(black); color.add(green);
    JTree jt=new JTree(style);
    f.add(jt);
    f.setSize(200,200);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TreeExample();  }}
```

Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

Commonly used Constructors:

Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
   f=new JFrame();
   JTextArea ta=new JTextArea(200,200);
   JPanel p1=new JPanel();
   p1.add(ta);
   JPanel p2=new JPanel();
   JPanel p3=new JPanel();
   JTabbedPane tp=new JTabbedPane();
   tp.setBounds(50,50,200,200);
   tp.add("main",p1);
   tp.add("visit",p2);
   tp.add("help",p3);
   f.add(tp);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
public static void main(String[] args) {
   new TabbedPaneExample();
}}
```

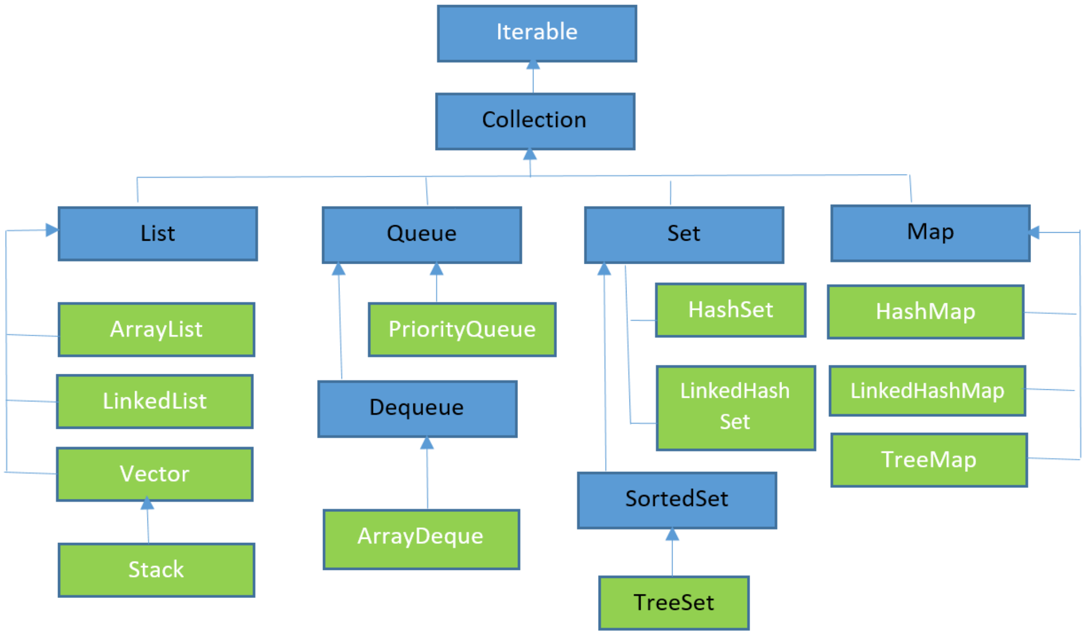
Collections in Java

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque,) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Collection Interface

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.Some of the methods of Collection interface are Boolean add (Object obj), Boolean addAll (Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.



List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.

1. boolean add(Object o):

- It adds an element of Specific Object type at the end of Arraylist as no index is mentioned in the method.
- It returns True if element is successfully added, and returns false if it is not.

2. void add(int index, Object element):

- It adds an element of Specific Object type at the specified index of the Arraylist as given in the argument of the method.
- It does not return anything as its return type is **void**.
- If in case the index specified is out of range it throws an **IndexOutOfBoundsException.**

3. boolean addAll(Collection c):

- This method adds each element of the Specific collection type at the end of the arraylist.
- It returns True if collection is successfully added, and returns false if it is not.
- If the collection passed in as an argument is null then it throws **Null Pointer Exception**

4. boolean addAll(int index, Collection c):

- This methods add each element of the Specific collection type at the specified index as mentioned in the argument.
- It returns true if collection is successfully added, and returns false if it is not.
- If the collection passed in as an argument is null then it throws **Null Pointer Exception.**The program of this method is:

5. void clear():

- This method remove all the elements of the arraylist.

6. Object clone():

- This method returns the exact same copy of the arraylist object.

7. boolean contains(Object element):

- This method returns true if the calling arraylist object contains the specific element as given in the argument list, otherwise it returns false.

12. Object remove(int index):

- It deletes the element from the given index from the arraylist.
- It returns an Exception IndexOutOfBoundsException, If index specified is out of range.

13.protected void removeRange(int first, int last):

- It deletes the group of elements from the first to last as mentioned in the argument.
- It includes the first index and excludes the last index

14.int size():

- This method returns the size of the arraylist.
- size() methods start count with 1 not 0.

```
import java.util.*;
public class Demo
{
    public static void main(String[] args)
    {
      ArrayList<String> l1= new ArrayList<String>();
```

```
                l1.add("aaa");
                l1.add("bbb");
                l1.add("ccc");
                l1.add("ddd");
                l1.add(3,"pppp");
                ArrayList<String> l2= new ArrayList<String>();
                 l2.add("Sangola");
                 l2.add("college");
                l2.addAll(2,l1);
                System.out.println("Hello");
        System.out.println(l1);
System.out.println("Given element is in the list" l1.contains("college"));
        System.out.println("Array list value from l2 beore remove");
         for(String s:l2)
         {
                 System.out.println(s);
         }
        l2.remove(1);
        System.out.println("Array list value from l2 after remove");
        for(String s:l2)
        {
                 System.out.println(s);
        }
        System.out.println("Size of Arraylist is "+l2.size());
       }

    }
```

```
import java.util.*;
class demo
 {
    public static void main(String args[])
     {
      ArrayList l1= new ArrayList();
       l1.add("sangali");
       l1.add("kolhapur");
       l1.add("satara");
       l1.add("pune");
       l1.add(11);
       l1.add(12.89);
       l1.add('a');
       System.out.println(l1);

      System.out.println(" Display Arraylist element using iterator ");
      Iterator it= l1.iterator();
      while(it.hasNext())
           {
             System.out.println(it.next());
            }
     System.out.println(" Display Arraylist element using foreach loop");
     for(Object s:l1)
        {
          System.out.println(s);
        }
    System.out.println(" Display Arraylist element using forloop");
     for(int i=0;i<l1.size();i++)
        {
          System.out.println(l1.get(i));
        }
System.out.println(" Display Arraylist element using sort method");
//     Collections.sort(l1);
     for(Object s:l1)
        {
          System.out.println(s);
        }
     }
}
```
LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

1. void addFirst(Object o):

This method add the specified element at the first position of the list.

2.void addLast(Object o):

This method add the specified element at the last position of the list

3.Object getFirst():

This method returns the first element in this linked list

4. **Object getLast():**

This method returns the last element in this linked list

5.Object remove(int index):

It deletes the element from the given index from the Linked List. It returns an IndexOutOfBoundsException if index specified is out of range

6. **Object set(int index, Object element):**

This method replaces the content at index given with the element given in argument list

```
import java.io.*;
import java.util.*;
public class Demo
{

        public static void main(String[] args)
        {
                LinkedList<Integer> al=new LinkedList<Integer>();
                 al.add(11);
                 al.add(12);
                 al.add(13);
                 al.add(14);
                 System.out.println("Linked List Element using foreach loop");
                 for(Object s:al)
                  System.out.println(s);
                 System.out.println("Linked List Element using iterable");
                 Iterator<Integer> i= al.iterator();
                 while(i.hasNext() )
                 {
                         System.out.println(i.next());
                 }
                 System.out.println("Linked List Element using for loop");
                 for(int j=0;j<al.size();j++ )
                 {
                         System.out.println(al.get(j));
                 }

                 System.out.println(" First Element of linked list is " +al.getFirst() );
                 System.out.println(" Last Element of linked list is " +al.getLast());
                 al.addFirst(10);
                 al.addLast(15);
                 al.set(5, 20);
         System.out.println("Linked List Element after add first and add last");
                 for(Object s:al)
                  System.out.println(s);

                 al.remove();
                 al.remove(3);
                 al.removeLast();
                 System.out.println("Linked List Element after remove");
                 for(Object s:al)
                  System.out.println(s);
                 al.clear();
                 System.out.println("Linked List Element after clear");
                 for(Object s:al)
                  System.out.println(s);
        }

}
```

Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

1. boolean add(E e)

This method adds element at the end of the vector

2. void add(int index, E element)

This method adds an element at specified index and moves the whole elements one step in forward direction

3. int capacity()

This method gives the capacity of the vector

4. void clear()

This method clears all the elements in the vector

5. clone clone()

This method gives a duplicate copy of the whole vector

6. boolean contains(Object o)

This method tells whether the vector contains the specified element in the vector, it will return true if that element is present or false if it is not presen

7.Object firstElement()

This method returns the first element at index 0

8.Object lastElement()

This method returns the last element of the vector

9. Object get(int index)

This method retrieves an element at specified index

```
import java.util.Vector;

public class Demo_vector
{

        public static void main(String[] args)
```

```
	{
		Vector<String> v= new Vector<String>();
		v.add("aaa");
		v.add("bbb");
		v.add("ccc");
		System.out.println("Value in vector as follow");
		for(String s:v)
		{
			System.out.println(s);
		}
		v.add(0,"zzz");
		System.out.println("Value in vector after add using index 0");
		for(String s:v)
		{
			System.out.println(s);
		}
		System.out.println("aaa value contain in vector "+v.contains("aaa"));
		System.out.println("first value of vector" +v.firstElement());
		System.out.println("last value of vector" +v.lastElement());
		System.out.println("In vector at 2 index value is  "+v.get(2));
		System.out.println("In vector at change value at index  2
						"+v.set(2,"sangola"));
		System.out.println("Vector value aftee set method");
		for(String s:v)
		{
			System.out.println(s);
		}
		System.out.println("aaa value is vector " +v.contains("aaa"));
		Vector<String> v1= new Vector<String>();
		v1=(Vector<String>)v.clone();
		System.out.println("Vector value in Vector v1");
		for(String s:v1)
		{
			System.out.println(s);
		}


	}

}
```

Stack

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

Method	Modifier and Type	Method Description
empty()	boolean	The method checks the stack is empty or not.
push(E item)	E	The method pushes (insert) an element onto the top of the stack.
pop()	E	The method removes an element from the top of the stack and returns the same element as the value of that function.
peek()	E	The method looks at the top element of the stack without removing it.
search(Object o)	int	The method searches the specified object and returns the position of the object.

```
import java.util.Stack;
import java.io.*;
public class Test {

        public static void main(String[] args)
        {
                // TODO Auto-generated method stub
                Stack<Integer> s= new Stack<Integer>();
                System.out.println(" Initially Capacity of stack is " +s.capacity());
                s.add(11);
                s.add(12);
                s.add(13);
                System.out.println("Valu of Stack");
                for(int a:s)
                {
                        System.out.println(a);
                }
    s.add(1,20);
    System.out.println("Value of Stack after add using index");
                for(int a:s)
                {
                        System.out.println(a);
                }
                System.out.println("Capacity of stack is " +s.capacity());
                System.out.println("11 value is in stack " +s.contains(11));
                System.out.println("First element of stack is " +s.firstElement());
                System.out.println("Last element of stack is " +s.lastElement());
                System.out.println("using get method element of stack is " +s.get(2));
                s.push(15);
                s.push(16);
                System.out.println("Value of Stack after using push");
                for(int a:s)
                {
                        System.out.println(a);
                }
                System.out.println("element of stack is popped " +s.pop());
                System.out.println("peek value from stack " +s.peek());
                System.out.println("15 value in stack  " +s.contains(15));
        }

}
```

Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

1. Set<data-type> s1 = **new** HashSet<data-type>();
2. Set<data-type> s2 = **new** LinkedHashSet<data-type>();
3. Set<data-type> s3 = **new** TreeSet<data-type>();

HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

Methods of Java HashSet class

Various methods of Java HashSet class are as follows:

SN	Modifier & Type	Method	Description
1)	boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
2)	void	clear()	It is used to remove all of the elements from the set.

3)	object	clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	contains(Object o)	It is used to return true if this set contains the specified element.
5)	boolean	isEmpty()	It is used to return true if this set contains no elements.
6)	Iterator<E>	iterator()	It is used to return an iterator over the elements in this set.
7)	boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.
8)	int	size()	It is used to return the number of elements in the set.

```
import java.util.*;

public class Test {

        public static void main(String[] args)
        {
                HashSet<Integer> h= new HashSet<Integer>();
                h.add(11);
                h.add(13);
                h.add(12);
                h.add(11);
                 System.out.println("Hashset Element  ");
                 for(int a:h)
                 {
                         System.out.println(a);
                 }
                 System.out.println("11 value is in hashset " +h.contains(11));
                 HashSet<Integer> h1= new HashSet<Integer>();

                 System.out.println("Is HashSet is empty " +h.isEmpty());
                 h1=(HashSet<Integer>)h.clone();
                 System.out.println("Hashset Element from hash set h1  ");
                 for(int a:h1)
                 {
                         System.out.println(a);
                 }
                 Iterator<Integer > i= h.iterator();
```

```
		System.out.println("Value from hash set using iterator");
		while(i.hasNext())
		  System.out.println(i.next());
		h.remove(12);
		System.out.println("Hashset Element after remove");
		for(int a:h)
		{
			System.out.println(a);
		}

	}

}
```

LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements. Various methods of Java LinkedHashSet class are as follows:

SN	Modifier & Type	Method	Description
1)	boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
2)	void	clear()	It is used to remove all of the elements from the set.
3)	object	clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	contains(Object o)	It is used to return true if this set contains the specified element.
5)	boolean	isEmpty()	It is used to return true if this set contains no elements.
6)	Iterator<E>	iterator()	It is used to return an iterator over the elements in this set.
7)	boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.

8)	int	size()	It is used to return the number of elements in the set.

```
import java.util.*;
public class Test {

        public static void main(String[] args)
        {
                LinkedHashSet<String> l=new LinkedHashSet<String>();
                 l.add("Sangali");
                 l.add("Satara");
                 l.add("kolhapur");

                 System.out.println("Element of Linked hash set is ");
                  for(String a:l)
                  {
                          System.out.println(a);
                  }
                  System.out.println("Element is contain in Linked hash set "
+l.contains("Sangali"));
              Iterator i= l.iterator();
              System.out.println("linked hast set element using iterator");
              while(i.hasNext())
              {
                System.out.println(i.next());
              }
              System.out.println("Element is  removed using remove method
"+l.remove("Satara"));
              System.out.println("Size of Linked Hash Set is "+l.size());
              l.clear();
              System.out.println("Element of Linked hash set after clear ");
                          for(String a:l)
                          {
                                  System.out.println(a);
                          }
         LinkedHashSet<String> h1= new LinkedHashSet<String>();
                         h1=(LinkedHashSet<String>)l.clone();
                         System.out.println("Element of Linked hash set after clone  ");
                          for(String a:h1)
                          {
                                  System.out.println(a);
                          }

        }

}
```

TreeSet
Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

1. boolean add(Object o):
This methods adds element in the object, which will automatically stores all the elements in increasing order

2. boolean addAll(Collection c):
This method adds all the elements of one object to another

3. void clear():
This method removes all of the elements from this obje

4. clone clone()
This method gives a duplicate copy of the whole vector

5. boolean contains(Object o)
This method tells whether the vector contains the specified element in the vector, it will return true if that element is present or false if it is not present

```
import java.util.*;
public class Test {

        public static void main(String[] args)
        {
    TreeSet<Integer> t= new TreeSet<Integer>();

     t.add(11);
                t.add(12);
                t.add(13);
                t.add(14);
                System.out.println("TreeSet Element using foreach loop");
                for(Object s:t)
                 System.out.println(s);
                System.out.println("TreeSet  Element using iterable");
                Iterator<Integer> i= t.iterator();
                while(i.hasNext() )
                {
                        System.out.println(i.next());
                }
                t.remove(3);
                 System.out.println("TreeSet Element after remove");
                for(Object s:t)
                 System.out.println(s);
                System.out.println("11 value is present in treeset " +t.contains(11));
                 t.clear();
                System.out.println("TreeSet Element after clear");
                for(Object s:t)
                 System.out.println(s);
        }

}
```

Map Interface

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.
A Map is useful if you have to search, update or delete elements on the basis of a key.

Java HashMap

Java **HashMap** class implements the Map interface which allows us *to store key and value pair*, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.
HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as HashMap<K,V>, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

Points to remember

- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap is non synchronized.
- Java HashMap maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75

public class HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneabl
e, Serializable

HashMap Methods in JAVA:

1. Object put(Object key, Object value):
This method adds the key value pair to the HashMap object,
2. int size():
This method returns the size of the HashMap
3. void clear():
This method clears all the key value pairs in the HashMap
4. Object clone():
This method returns the exact copy of the HashMap
5. boolean containsKey(Object key):
This method checks the presence of specified key in the HashMap, it will return true if the key is present and false if the mentioned key is not present
6. boolean containsValue(Object value):
This method checks the presence of specified value in the HashMap, it will return true if the value is present and false if the mentioned value is not present
7. Set entrySet():
This method returns a Set view of the mappings contained in the HashMap

8.Object get(Object key):
This method returns the value corresponding to the specified key
9.boolean isEmpty():
This method as the name suggests checks whether the HashMap is empty or not, It will return true if it is empty, or false if it is not empty,
10. Set keySet():
This method returns a Set view of the keys contained in the HashMap
11. putAll(Map m):
This method copies all the key value pair form one hashmap to another
12. Object remove(Object key):
This method as the name specifies removes the key value pair, corresponding to the mentioned key in the argument list
13. Collection values():
This method returns the collection of all the values in the hashmap

```
import java.util.HashMap;
import java.util.Map;

public class Test
{
   public static void main(String[] args)
        {
           HashMap<Integer,String> h= new HashMap<Integer,String>();
                h.put(1,"pen");
                h.put(2, "Pencil");
                h.put(3, "Notebook");
                h.put(4, "Eraser");
                System.out.println("Element of HashMap is ");
                for(Map.Entry m:h.entrySet())
                            {
 System.out.println("Key is " +m.getKey()+" Value is " +m.getValue());
                          }
 System.out.println("Conatins key method apply for hashmap is " +h.containsKey(1));
  System.out.println("Conatins Values method apply for hashmap is "
                                                      +h.containsValue("Pen"));
 System.out.println("Entry key method apply for hashmap is " +h.entrySet());
 System.out.println("get method apply for hashmap is " +h.get(2));
 System.out.println("keyset key method apply for hashmap is " +h.keySet());

 HashMap<Integer,String> h1= new HashMap<Integer,String>();
 h1.putAll(h);
 System.out.println("using putall method apply for hashmap1 is " +h1.keySet());
 System.out.println("Element of HashMap1 is ");
 h1.remove(1);
for(Map.Entry m:h1.entrySet())
{
   System.out.println("Key is " +m.getKey()+" Value is " +m.getValue());
}
h1.remove(1);
System.out.println("values method apply for hashmap1 is " +h1.values());
}
}
```

Java TreeMap class

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

- Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- Java TreeMap contains only unique elements.
- Java TreeMap cannot have a null key but can have multiple null values.
- Java TreeMap is non synchronized.
- Java TreeMap maintains ascending order.

public class TreeMap<K,V> **extends** AbstractMap<K,V> **implements** NavigableMap<K,V>,

1. Object put(Object key, Object value):

This method adds the key value pair to the TreeMap object,

2. int size():

This method returns the size of the TreeMap

3. void clear():

This method clears all the key value pairs in the TreeMap

4. Object clone():

This method returns the exact copy of the TreeMap

5. boolean containsKey(Object key):

This method checks the presence of specified key in the TreeMap, it will return true if the key is present and false if the mentioned key is not present

6. boolean containsValue(Object value):

This method checks the presence of specified value in the TreeMap, it will return true if the value is present and false if the mentioned value is not present

7. Set entrySet():

This method returns a Set view of the mappings contained in the TreeMap

8.Object get(Object key):

This method returns the value corresponding to the specified key

9.boolean isEmpty():

This method as the name suggests checks whether the TreeMap is empty or not, It will return true if it is empty, or false if it is not empty,

10. Set keySet():

This method returns a Set view of the keys contained in the TreeMap

11. putAll(Map m):

This method copies all the key value pair form one TreeMap to another

12. Object remove(Object key):

This method as the name specifies removes the key value pair, corresponding to the mentioned key in the argument list

13. Collection values():
This method returns the collection of all the values in the TreeMap

```
import java.util.TreeMap;
import java.util.Map;

public class Test
{
public static void main(String[] args)
{
   TreeMap<Integer,String> t= new TreeMap<Integer,String>();
       t.put(1,"pen");
        t.put(2, "Pencil");
        t.put(3, "Notebook");
       t.put(4, "Eraser");
 System.out.println("Element of TreeMap is ");
for(Map.Entry m:t.entrySet())
   {
       System.out.println("Key is " +m.getKey()+" Value is " +m.getValue());
   }
 System.out.println("Conatins key method apply for TreeMap is " +t.containsKey(1));
 System.out.println("Conatins Values method apply for TreeMap is "+t.containsValue("Pen"));
 System.out.println("Entry key method apply for TreeMap is " +t.entrySet());
 System.out.println("get method apply for TreeMap is " +t.get(2));
System.out.println("keyset key method apply for TreeMap is " +t.keySet());

TreeMap<Integer,String> t1= new TreeMap<Integer,String>();
                    t1.putAll(t);
System.out.println("using putall mettod apply for TreeMap1 is " +t1.keySet());
System.out.println("Element of TreeMap1 is ");
 t1.remove(1);
   for(Map.Entry m:t1.entrySet())
          {
           System.out.println("Key is " +m.getKey()+" Value is " +m.getValue());
          }
          t1.remove(1);
          System.out.println("values method apply for TreeMap1 is " +t1.values());
          //t1.clear();
          TreeMap <Integer,String> t2= new TreeMap<Integer,String>();
          t2=(TreeMap<Integer, String>)t1.clone();
          System.out.println("Element of TreeMap2 is ");
          for(Map.Entry m:t2.entrySet())
             {
              System.out.println("Key is " +m.getKey()+" Value is " +m.getValue());
           }
          }
}
```

Java LinkedHashMap class

Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

Points to remember

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains unique elements.
- Java LinkedHashMap may have one null key and multiple null values.
- Java LinkedHashMap is non synchronized.
- Java LinkedHashMap maintains insertion order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

LinkedHashMap class declaration

Let's see the declaration for java.util.LinkedHashMap class.

```
public class LinkedHashMap<K,V> extends HashMap<K,V> implements Map<K,V>
```

1. Object put(Object key, Object value):

This method adds the key value pair to the LinkedHashMap object,

2. int size():

This method returns the size of the LinkedHashMap

3. void clear():

This method clears all the key value pairs in the LinkedHashMap

4. Object clone():

This method returns the exact copy of the LinkedHashMap

5. boolean containsKey(Object key)

This methods checks whether the key given in the argument list is present or not in the LinkedHashMap entries

6. boolean containsValue(Object value)

This methods checks whether the value given in the argument list is present or not in the LinkedHashMap entries

7. Object get(Object key)

This method returns the value corresponding to the given key in the argument list

8. Object remove(Object key):

This method as the name specifies removes the key value pair, corresponding to the mentioned key in the argument list

```
import java.util.LinkedHashMap;
import java.util.Map;

public class Test {

    public static void main(String[] args)
    {
        LinkedHashMap<Integer,String> l= new LinkedHashMap<Integer,String>();
        l.put(1,"Pen");
        l.put(2,"Pencile");
```

```
                l.put(3,"Eraser");
                l.put(4,"Book");
        System.out.println("Element from Linkedhash Map is");
        for(Map.Entry<Integer,String> s:l.entrySet())
        {
            System.out.println("key is "+s.getKey()+" Value is "+s.getValue());
        }
        System.out.println("Size of LinkedHash Map is " +l.size());
        LinkedHashMap<Integer,String> l1= new LinkedHashMap<Integer,String>();
         l1=(LinkedHashMap<Integer,String>)l.clone();
         System.out.println("Element from LinkedhashMap 1 is");
         for(Map.Entry<Integer,String> s:l.entrySet())
         {
            System.out.println("key is "+s.getKey()+" Value is "+s.getValue());
         }
         System.out.println("Value is remove from linkedHashMap1 is "+l1.remove(3));
         System.out.println("Value is in linkedHashMap is "+l.containsKey(11));
         System.out.println("Value is in linkedHashMap is "+l.containsValue("Pen"));
         System.out.println("Value is in linkedHashMap is "+l.entrySet());
         l.replace(1,"Marker");
         System.out.println("Element from Linkedhash Map is after replace method");
         for(Map.Entry<Integer,String> s:l.entrySet())
         {
            System.out.println("key is "+s.getKey()+" Value is "+s.getValue());
         }
         l.clear();
            }

}
```