Multithreading in java

Multithreading in java is process of executing multiple threads simultaneously.

What is the Thread?

A thread is lightweight sub process, the smallest unit of processing it is a separate path of execution or thread is sub part of process

What is the process?

Process current running program in ram or application execution in ram called as process.

Why use the multi threading in java or Advantages of Multithreading in java

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

How to implement the thread practically in java

If we want to use the thread in java we have the two ways
1) Using Thread class
2) Using Runnable interface

Note: Threading is concept of operating system and java is language where we can implement the thread concept

How to Implement the Thread by using Thread class

If we want to implement or use the thread in java using Thread class we have the some important steps.

1) Add the java.lang package in application

import java.lang.* : java.lang is default package in java means we not need to import it.

2) Create the class and inherit the Thread class in it

```
class A extends Thread
{

}
```

3) Override the run () method and write the thread logics

run () method is originally member of Runnable interface and Thread is implementer class of Runnable interface so we have to override the run method for writing the thread logic

Internal code of Run implementation

```
interface Runnable
{
   public void run();
}
class Thread implements Runnable
{
    public void run()
    {
    }

}
class A extends Thread
{  public void run()
   { //write here logics of thread
   }

 }
```

run () is abstract method from Runnable interface and which is used for writing the logics of thread.

Why java provide the run() as abstract method

It is best example of abstraction suppose consider we have the three java developer name as A ,B and C and every developer want to write the logic of thread but every developer having different logic of thread so java provide the only one method name as run() as abstract so every developer can override the run() method and can modify its logic as per his need.

```
class A extends Thread
{
   public void run()
   {
      try
      {
         for(int i=1; i<=5; i++)
         { System.out.printf("I =%d\n",i);
         }
      }
      catch(Exception ex)
      {  System.out.println("Error is  "+ex);
      }
   }
}
```

4) Create the object of Thread child class and use the Thread methods to work with Thread

Thread class provide the number method to us to manage the thread

void start() : this method is used for start the thread means when we call the start() method then run() method automatically get executed internally.

static void sleep(int milliseconds) : this method is used for sleep or stop thread execution for specified time period not permenantly.

void stop(): this method is used for stop the execution of thread.

void join(): this method can hold the another execution thread execution when ever running thread is not completed.

boolean isAlive(): this method can check wheather thread is running or not or live or not if thread is running return true otherwise return false.

void notify(),void notify(),wait(),wait(int),yield() etc : we will discuss later in this chapter.

Example of Thread using start(),run() and sleep() methods

```
package org.techhub;
import java.lang.*;
class A extends Thread
{  public void run()
        {   try
                {   for(int i=1;i<=5; i++)
                        {System.out.printf ("First Thread is %d\n",i);
                          Thread.sleep (10000);
                        }
                }
                catch (Exception ex)
                {    System.out.println ("Error is "+ex);
                }
        }
}
class B extends Thread
{      public void run()
                {   try
                        {   for(int i=1;i<=50; i++)
                                { System.out.printf("Second Thread is %d\n",i);
                                  Thread.sleep(1000);
                                }
                        }
                        catch(Exception ex){
                          System.out.println("Error is "+ex);
                        }
                }
}
public class ThreadingApplication {
        public static void main(String[] args) {
                A a1 = new A();
                 a1.start();
                B b1 = new B();
                b1.start();
        }
}
```

join () methods of Thread class

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 or join() method hold the other thread execution when ever running thread is not completed

Example

```
package org.techhub;
import java.lang.*;
class A extends Thread
{  public void run()
        {   try
                {   for(int i=1;i<=5; i++)
                        { System.out.printf("First Thread is %d\n",i);
                          Thread.sleep(10000);
                        }
                }
                catch(Exception ex)
                {
                        System.out.println("Error is "+ex);
                }
        }
}
class B extends Thread
{     public void run()
                {   try
                        {   for(int i=1;i<=50; i++)
                                { System.out.printf("Second Thread is %d\n",i);
                                  Thread.sleep(1000);
                                }
                        }
                        catch(Exception ex)
                        {  System.out.println("Error is "+ex);
                        }
                }
}
public class ThreadingApplication {
```

```
    public static void main(String[] args)throws Exception {
        // TODO Auto-generated method stub

         A a1 = new A();
          a1.start();
          a1.join();
         B b1 = new B();
         b1.start();
    }

}
```

boolean isAlive(): this method is used for check whether thread is live or not means check thread is running or not if thread is running return true otherwise return false.

Example

```
package org.techhub;
import java.lang.*;
class A extends Thread
{  public void run()
    {   try
        {   for(int i=1;i<=5; i++)
            { System.out.printf("First Thread is %d\t%b\n",i,isAlive());
              Thread.sleep(10000);
            }
        }
        catch(Exception ex)
        {
            System.out.println("Error is "+ex);
        }
    }
}
class B extends Thread
{     public void run()
        {   try
            {   for(int i=1;i<=50; i++)
```

```
                    { System.out.printf("Second Thread is %d\n",i);
                      Thread.sleep(1000);
                    }
                }
                catch(Exception ex)
                {  System.out.println("Error is "+ex);
                }
            }
}
public class ThreadingApplication {
    public static void main(String[] args)throws Exception {
            // TODO Auto-generated method stub
             A a1 = new A();
              a1.start();
              a1.join();
              //die first thread
              System.out.println("Now Status of First Thread is "+a1.isAlive());
             B b1 = new B();
             b1.start();
    }
}
```

void stop(): this method is used for stop the execution of thread or terminate the thread execution**.**

```
package org.techhub;
import java.lang.*;
class A extends Thread
{  public void run()
    {   try
        {   for(int i=1;i<=5; i++)
            { System.out.printf("First Thread is %d\t%b\n",i,isAlive());
              if(i==3)
              {
                  stop(); //terminate the execution of thread
              }
             Thread.sleep(10000);
            }
```

```
                }
                catch(Exception ex)
                {
                        System.out.println("Error is "+ex);
                }
        }
}
class B extends Thread
{       public void run()
                {  try
                        {  for(int i=1;i<=50; i++)
                                { System.out.printf("Second Thread is %d\n",i);
                                  Thread.sleep(1000);
                                }
                        }
                        catch(Exception ex)
                        {  System.out.println("Error is "+ex);
                        }
                }
}
public class ThreadingApplication {

        public static void main(String[] args)throws Exception {
                // TODO Auto-generated method stub
                 A a1 = new A();
                  a1.start();
                  a1.join();
                  //die first thread
                  System.out.println("Now Status of First Thread is "+a1.isAlive());
                 B b1 = new B();
                 b1.start();
  }

}
```

Synchronization and Asynchronization in Threading

Synchronization means if two or more than two threads use the single resource/object one by one sequentially called as synchronization in Thread.

Asynchronization means if two or more than two threads use the single resource/object simultaneously called as Asynchronization.

Q. What is the meaning resource?

Here resource indicate the object of the particular class

Example

```
package org.techhub;
class Table
{
 synchronized void showTable(int x)
        {
                try
                {
                        for(int i=1;i<=10;i++)
                        {
                                System.out.printf("%d X %d  = %d\n",i,x,i*x);
                                Thread.sleep(1000);
                        }
                }
                catch(Exception ex)
                {
                        System.out.println("Error is "+ex);
                }
        }
}

class Two extends Thread
{
 Table t;
 void setTable(Table t)
 {
        this.t=t;
 }
```

```
  public void run()
  {
          t.showTable(2);
  }
}
class Three extends Thread
{   Table t;
        void setTable(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.showTable(3);
        }
}
public class SyncAsyncApp {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
       Table t=new Table();
       Two tw=new Two();
       tw.setTable(t);
       tw.start();
       Three th=new Three();
       th.setTable(t);
       th.start();
        }

}
```

Note: refer Thread part-5 video for above concept.

Wait (), notify() and notifyAll() methods of Thread class

Wait () method is used for hold the thread execution for specified time period.

There are two types of wait

1) Conditional wait: conditional wait means if thread re-execute automatically after some specified time period called conditional wait.

Syntax: void wait (int milliseconds): this is used for hold the thread execution for specified time period when time period finish then thread re-executes itself.

2) Unconditional wait: unconditional wait means thread not re-execute self we have to send the request to thread for re-execution purpose for that we have the two methods name as notify() and notifyAll()

Syntax: void wait ():

notify(): notify() method is used for recall the waited thread for re-execution in first in first out format and call one thread at time.

notifyAll(): this method is used for call the all waited thread in single stroke and in last in first out format.

Example

```
package org.techhub;
import java.util.*;
class Table {
	synchronized void showTable(int x) {
		try {
			for (int i = 1; i <= 10; i++) {
				System.out.printf("%d X %d  = %d\n", i, x, i * x);
				if (i == 5) {
					wait(); // unconditional wait
				}
				Thread.sleep(1000);
			}
		} catch (Exception ex) {
			System.out.println("Error is " + ex);
		}
	}
```

```
	synchronized void restartTable() {
		try {
			notifyAll();
		} catch (Exception ex) {
			System.out.println("error is " + ex);
		}
	}
}

class Two extends Thread {
	Table t;

	void setTable(Table t) {
		this.t = t;
	}

	public void run() {
		t.showTable(2);
	}
}
class Three extends Thread {
	Table t;

	void setTable(Table t) {
		this.t = t;
	}

	public void run() {
		t.showTable(3);
	}
}

public class SyncAsyncApp {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Table t = new Table();
```

```
            Two tw = new Two();
            tw.setTable(t);
            tw.start();
            Three th = new Three();
            th.setTable(t);
            th.start();
            Scanner xyz  = new Scanner(System.in);
            do {
             String msg=xyz.nextLine();
              if(msg.equals("restart"))
              {
                    t.restartTable();
              }
              if(msg.equals("stop"))
              {
                    System.exit(0);
              }
            }while(true); //infinite string
      }
}
```

What is the diff between wait () and sleep()

1) wait () is method of java.lang.Object and sleep() is static method of Thread class
2) wait () method can work with conditional wait as well as uncondition wait but sleep() method can only work with conditional wait
3) wait() method only works in synchronized block and sleep() can work in synchronized as well as asynchronized block.

What is the diff between notify () and notifyAll () method

notify() method call one thread time from waiting stage of threads and work in first in first out manner means it work in queue format and notifyAll() method call all the waited thread in last in first out format means notifyAll() method work as stack format.
The similarity of notify () and notifyAll () is both method from java.lang.Object class as well as they need synchronized block if we want to use it.

How to create the thread by using Runnable interface

Q. why we need to implement the thread by using Runnable interface?

If we use the Thread class for thread implementation purpose then there is possibility of multiple inheritance and in java we cannot perform the multiple inheritance by using class for that we have to use the interface for resolve this problem we use the Runnable interface.

How to implement the thread by using Runnable interface

If we want to use the thread by using Runnable interface we have the some important steps.

1) add the java.lang package in application

java.lang is default package of java we not need to import it.

2) Create the class and implement the Runnable interface in it

```
class A implements Runnable
{
}
```

3) Override the run() method of Thread class and write its logic

```
class A implements Runnable
{
   public void run()
   {
        try
        {
            for(int i=1; i<=5; i++)
             { System.out.printf("I =%d\n",i);
                Thread.sleep(1000);
             }
        }
        catch(Exception ex)
        { System.out.println("Error is  "+ex);
        }
   }
}
```

3) Create the object of class in which Runnable implemented

A a1 =new A();

4) create the object of Thread class in given fashion

Thread ref = new Thread (Runnable): when we use the Runnable interface for thread implementation purpose then we have the pass Runnable interface reference in Thread class constructor. So as per the rule dynamic polymorphism if we have the reference of parent class then we can pass the reference of child as parent.

A a1 = new A();

Thread t = new Thread(a1);

Sample code

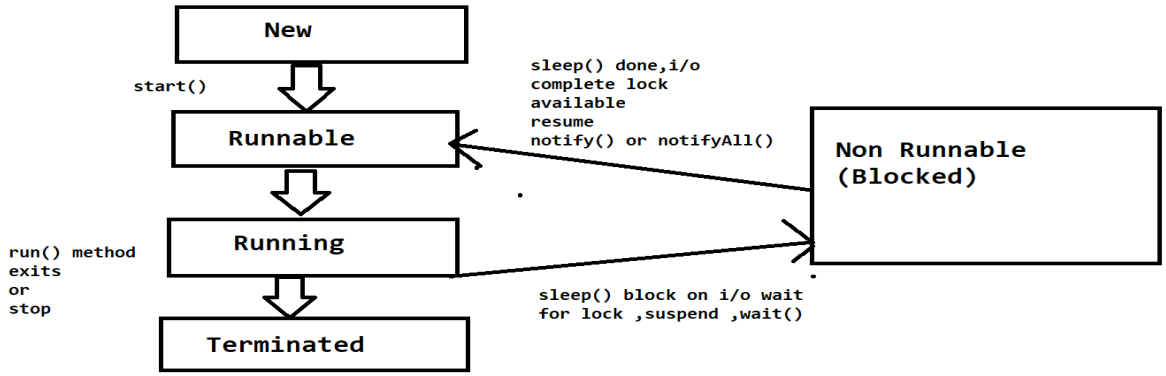
```
package org.techhub;
class A implements Runnable
{
        public void run() {
                try
```

```
            {
                for (int i=1; i<=5; i++)
                {       System.out.printf ("I=%d\n", i);
                        Thread.sleep (10000);
                }
            }
            catch (Exception ex)
            {  System.out.println ("Error is "+ex);
            }
        }
}
public class ThreadUsingRunnableApp {
        public static void main (String [] args) {
            A a1=new A ();
            Thread t=new Thread (a1);
             t.start ();
        }
}
```

Thread Life Cycle

A thread can be in one of the five states according to sun, there are 4 states in thread life cycle in java, new, runnable and non runnable and terminated. There is no running state but for better understanding the thread we have to know the five 5 states of thread

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



1) New: The thread is in new state if you create an instance of Thread class but before the invocation of state method

class A extends Thread

{

}

A a1 = new A(); //New state

2) Runnable: the thread is in runnable state after invocation of start () method but the thread scheduler has no selected it to be then running thread.

a1.start (); //runnable

3) Running: the thread is in running state if the thread scheduler has selected it.

e.g public void run() { logic }

4) Non-Runnable (Blocked): this is the state when the thread is still alive but is currently not eligible to run

e.g sleep(),wait() et**c**

5) Terminated: a thread is in terminated or dead state when its run() method is exists.

Executor Services and Thread pool

Executor a framework for creating and managing threads and thread pool.

If use the Executor Framework or Executor framework help you

1) Thread Creation: it provides various methods for creating thread more specially manage the thread pool**.** That your application can use to run task concurrently.
2) Thread Management : it manages the life cycle of the thread in the thread pool. You don't need to worry about whether thread in thread pool is active or busy or dead before submitted task for execution.
3) Task Submission and Execution: Executor Framework provides methods for submitting task for execution in the thread pool and also gives power to decide when task will be executed.
Example: schedule task to be executed later or make then execute periodically.
Java Currency API: Api defines the following three executor interfaces that cover everything that is needed for creating and managing thread using executor services.

1) Executor: a sample interface that contains a method called executed (Runnable) to launch task specified by Runnable object.
2) Executor Service: A sub interface of executor that add functionality to manage life cycle of the task. It also provide a submit method whose overloaded version can accept Runnable as well as Callable Object**.**
Callable is similar to Runnable to except that the task specified by callable object can also return value**.**
3) ScheduledExecutorService: A sub interface of ExecutorService . It add functionality to schedule the execution of the task**.**
Apart from the above three interfaces The API also provides an executors class. That contains factory method for creating different kind of Executor Service.

Thread Pool Concept

A Thread pool is used for reuse the previously created threads to execute the current task and offer a solution to the problem of thread life cycle overhead and resource thrashing.

Why we need to Thread Pooling concept

In Server programming such as database and web server repeatedly execute the requests from multiple clients and these are oriented around processing a large number of short task. An Approach for building server application would be to create new thread each time a request arrives and service this new request in the new created thread. While this approach seems simple to implement it has significant disadvantages. A server that creates a new thread for every request would spend more time and consume more system resources in creating and destroying threads than processing actual request. Since active threads consume system resources, JVM creating too many threads at the same time can cause the system to run out of memory.
So better way creating new thread every time and destroying if we reuse the same created thread for new request then we have to use the thread pool constant.