**Multithreading in Java**

Multithreading in java is a process of executing multiple threads simultaneously.Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.Java Multithreading is mostly used in games, animation etc.

**Thread in java**

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.

**Life cycle of a Thread (Thread States)**

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

New

Runnable

Running

Non-Runnable (Blocked)

Terminated

**1) New**

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

**2) Runnable**

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

**3) Running**

The thread is in running state if the thread scheduler has selected it.

**4) Non-Runnable (Blocked)**

This is the state when the thread is still alive, but is currently not eligible to run.

**5) Terminated**

A thread is in terminated or dead state when its run() method exits.

**How to create thread**

There are two ways to create a thread:

**By extending Thread class**

**By implementing Runnable interface.**

**Thread class:**

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

**1) Java Thread Example by extending Thread class**

class Multi extends Thread{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

Multi t1=new Multi();

t1.start();

}

}

**Runnable interface:**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

**public void run():**It is used to perform action for a thread.

**2) Java Thread Example by implementing Runnable interface**

class Multi3 implements Runnable{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

Multi3 m1=new Multi3();

Thread t1 =new Thread(m1);

t1.start();

}

}

=====================================================================================

**Thread Scheduler in Java**

Thread scheduler in java is the part of the JVM that decides which thread should run.

There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

=====================================================================================

**Sleep method in java**

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

class Sleep1 extends Thread{

public void run(){

for(int i=1;i<5;i++){

try{

Thread.sleep(500);

}

catch(InterruptedException e)

{

System.out.println(e);

}

System.out.println(i);

}

}

public static void main(String args[]){

TestSleepMethod1 t1=new TestSleepMethod1();

TestSleepMethod1 t2=new TestSleepMethod1();

t1.start();

t2.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.

class Join1 extends Thread{

public void run(){

for(int i=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();

}

}

=====================================================================

**getName(),setName(String) and getId() method**:

class TestJoinMethod3 extends Thread{

public void run(){

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

}

public static void main(String args[]){

TestJoinMethod3 t1=new TestJoinMethod3();

TestJoinMethod3 t2=new TestJoinMethod3();

System.out.println("Name of t1:"+t1.getName());

System.out.println("Name of t2:"+t2.getName());

System.out.println("id of t1:"+t1.getId());

t1.start();

t2.start();

t1.setName("Sonoo Jaiswal");

System.out.println("After changing name of t1:"+t1.getName());

}

}

=====================================================================================

**The currentThread() method:**

The currentThread() method returns a reference to the currently executing thread object.

class TestJoinMethod4 extends Thread{

public void run(){

System.out.println(Thread.currentThread().getName());

}

}

public static void main(String args[]){

TestJoinMethod4 t1=new TestJoinMethod4();

TestJoinMethod4 t2=new TestJoinMethod4();

t1.start();

t2.start();

}

}

=====================================================================================

**Naming Thread**

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on.

class TestMultiNaming1 extends Thread{

public void run(){

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

}

public static void main(String args[]){

TestMultiNaming1 t1=new TestMultiNaming1();

TestMultiNaming1 t2=new TestMultiNaming1();

System.out.println("Name of t1:"+t1.getName());

System.out.println("Name of t2:"+t2.getName());

t1.start();

t2.start();

t1.setName("Sonoo Jaiswal");

System.out.println("After changing name of t1:"+t1.getName());

}

}

=====================================================================================

**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 schedular 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

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();

}

}

===============================================================================

**Daemon Thread in Java**

Daemon thread in java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

**Note:** If we want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.

====================================================================

**Java Thread Pool**

Java Thread pool represents a group of worker threads that are waiting for the job and reuse many times.In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.