**MULTITHREADING**

**Multithreading :-**

Multithreading in java is a process of executing multiple threads simultaneously.

**Thread :-**

Thread is a single sequence stream with in a process. it is also called as lightweight process.

**Advantage of 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.

**Multitasking**:-

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

* Process-based Multitasking(Multiprocessing)
* Thread-based Multitasking(Multithreading)

1) **Process-based Multitasking (Multiprocessing)**

* Each process have its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

2) **Thread-based Multitasking (Multithreading)**

* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between the thread is low.

**Life cycle in 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

**Creating Thread**

we can create a thread by two ways like

1) By extending thread class

2) By implementing runnable interface

**By extending thread class -** we extending a thread class by using **extend** keyword.

Ex:-

class Multi extends Thread{

public void run(){

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

}

//main method is default thread creating by JVM

public static void main(String args[]){

Multi t1=new Multi();

t1.start(); //child thread class and it call the rum method

}

}

**By implementing runnable interface -** we implementing a runnable interface by using implement keyward. 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().

|  |
| --- |
|  |

|  |
| --- |
| 1. **public void run():**is used to perform action for a thread. |

Ex:-

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 creats main method to run run method

//m1 is the target runnable

Thread t1 =new Thread(m1);

t1.start();

}

}

**Note :**

**start() method** of Thread class is used to start a newly created thread. It performs following tasks:

* A new thread starts(with new callstack).
* The thread moves from New state to the Runnable state.
* When the thread gets a chance to execute, its target run() method will run.

Commonly used methods of Thread class:

1. **public void run():**is used to perform action for a thread.
2. **public void start():**starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):**Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():**waits for a thread to die.
5. **public void join(long miliseconds):**waits for a thread to die for the specified miliseconds.
6. **public int getPriority():**returns the priority of the thread.
7. **public int setPriority(int priority):**changes the priority of the thread.
8. **public String getName():**returns the name of the thread.
9. **public void setName(String name):**changes the name of the thread.
10. **public Thread currentThread():**returns the reference of currently executing thread.
11. **public int getId():**returns the id of the thread.
12. **public Thread.State getState():**returns the state of the thread.
13. **public boolean isAlive():**tests if the thread is alive.
14. **public void yield():**causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():**is used to suspend the thread(depricated).
16. **public void resume():**is used to resume the suspended thread(depricated).
17. **public void stop():**is used to stop the thread(depricated).
18. **public boolean isDaemon():**tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):**marks the thread as daemon or user thread.
20. **public void interrupt():**interrupts the thread.
21. **public boolean isInterrupted():**tests if the thread has been interrupted.
22. **public static boolean interrupted():**tests if the current thread has been interrupted.

**Sleep method in java**

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

**Syntax of sleep() method in java**

The Thread class provides two methods for sleeping a thread:

* public static void sleep(long miliseconds)throws InterruptedException
* public static void sleep(long miliseconds, int nanos)throws InterruptedException

**Example :-**

class TestSleepMethod1 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();

}

}

**Can we start a thread twice?**

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

**Example**

public class TestThreadTwice1 extends Thread{

public void run(){

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

}

public static void main(String args[]){

TestThreadTwice1 t1=new TestThreadTwice1();

t1.start();

t1.start();

}

}

**output**

running

Exception in thread "main" java.lang.IllegalThreadStateException

**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. By we can change the name of the thread by using setName() method. The syntax of setName() and getName() methods are given below:

* public String getName(): is used to return the name of a thread.
* public void setName(String name): is used to change the name of a thread.

**Example :-**

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());

}

}

**Current** Thread

The currentThread() method returns a reference of currently executing thread.

* Public static Thread currentThread()

**Example**

class TestMultiNaming2 extends Thread{

public void run(){

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

}

public static void main(String args[]){

TestMultiNaming2 t1=new TestMultiNaming2();

TestMultiNaming2 t2=new TestMultiNaming2();

t1.start();

t2.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.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

**Points**

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.