**Multithreading in Java-**

Multiple threading is the process of executing multiple threads simultaneously.

Multitasking is a process of executing multiple tasks simultaneously. It is achieved by using two ways.

a) Process Based b) Thread Based

Example for Multitasking-

In online session, what are the different activities done by students as?

- Listen the class
- Taking running notes
- Checking mobile

Process based-

1. Executing several tasks simultaneously where each task is separate independent process such as multitasking is called as process based.
2. Example 1- Typing java program into eclipse, also listening the audio songs, download a file from internet.
3. In this every activity is independent process here.
4. Example-2 Task manager, see the multiple process list. Operating system is a process based multitasking.
5. Process is heavy weight components.
6. Each process has address into memory.

Thread based-

1. Executing several tasks simultaneously where each task is separate part of same program called as thread based.
2. Example- suppose I have 1000 lines of code into java program and it will takes 8 hours to execute it where first 500 line is executed after that remaining 500 lines is executed but there is no any dependency between them so I can run that tasks simultaneously to minimize the execution time.
3. Thread is light weight components.
4. Thread shares the same address space.

What is thread?

The smallest unit of program called as Thread.

How to create the Thread?

There are two ways to create the thread as

1. By extending Thread class
2. By implementing Runnable interface.

1. By extending 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.

Constructors

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Methods-

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.

**Program for Thread-**

```
package com.threads;

public class ThreadDemo extends Thread {

	public void run() {

		for (int i = 1; i <= 10; i++) {
			System.out.println(i);
		}
	}

	public static void main(String[] args) {

		ThreadDemo thread = new ThreadDemo();
		thread.start();
	}
}
```

2. By implementing 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().

Program-1

```
package com.threads;

public class ThreadDemo implements Runnable {

	public static void main(String[] args) {

		ThreadDemo thread = new ThreadDemo();
		Thread t = new Thread(thread);
		t.start();
	}
```

```
	@Override
	public void run() {

		for (int i = 1; i <= 10; i++) {
			System.out.println(i);
		}

	}
}
```

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

When to use which way?

**Extending thread class**- if the class is not extending another class then we should go for thread class.

**Implementing runnable interface**- If our class is already extending another class then we could not use extend keyword due to multiple inheritance. So best way to go for runnable interface.

Difference between thread class and runnable interface(Self assignments)

Example- Program for multithreading

```
package com.multi;

public class MultithreadingExample extends Thread {

	public void run() {

		for (int i = 1; i <= 5; i++) {
			try {
				Thread.sleep(500);// it will pause the
thread execution for perticular milliseconds
			} catch (Exception e) {
				System.out.println(e.getMessage());
```

```
                }
                System.out.println(i);
            }
        }

        public static void main(String[] args) {

            MultithreadingExample thread1 = new
MultithreadingExample();
            MultithreadingExample thread2 = new
MultithreadingExample();
            thread1.start();
            thread2.start();

        }

}
```

Output-

```
1
1
2
2
3
3
4
4
5
5
```

Advantages for multithreading in java

The users are not blocked because threads are independent, and we can perform multiple operations at times

Program-3

```
package com.threads;

public class ThreadDemo extends Thread {

	public static void main(String[] args) {

		ThreadDemo thread = new ThreadDemo();
		System.out.println(thread.getId());
		System.out.println(thread.getName());
		System.out.println(thread.getPriority());
		System.out.println(thread.getState());

	}
}
```

Output-

```
12
Thread-0
5
NEW
```

**Note:**

1. In the case of t.start() a new Thread will be created which is responsible for the execution of run() method. But in the case of t.run() no new Thread will be created and run() method will be executed just like a normal method by the main Thread.
2. Start() has responsibility of :
    a) Register Thread with Thread Scheduler.
    b) All other mandatory low level activities.
    c) Invoke or calling run() method.
3. If we are not overriding run() method then Thread class run() method will be executed which has empty implementation and hence we won't get any output.
4. We can overload run() method but Thread class start() method always invokes no argument run() method the other overload run() methods we have to call explicitly then only it will be executed just like normal method.

5. If we override start() method then our start() method will be executed just like a normal method call and no new Thread will be started.

**RACE condition:**

Executing multiple Threads simultaneously and causing data inconsistency problems is nothing but Race condition we can resolve race condition by using synchronized keyword.

**Method which are not recommanded to use :**

1) We can call stop() method to stop a Thread in the middle then it will be entered into dead state immediately.
   public final void stop();
2) A Thread can suspend another Thread by using suspend() method then that Thread will be paused temporarily.
   public final void suspend();
3) A Thread can resume a suspended Thread by using resume() method then suspended Thread will continue its execution.
   public final void resume();

**How to prevent thread execution:**

We can prevent(stop) a Thread execution by using the following methods.

1. yield();
   yield() method causes "to pause current executing Thread for giving the chance of remaining waiting Threads of same priority".
   public static native void yield();
2. join();
   If a Thread wants to wait until completing some other Thread then we should go for join() method.
   Example: If a Thread t1 executes t2.join() then t1 should go for waiting state until completing t2.
   a. public final void join()throws InterruptedException
   b. public final void join(long ms) throws InterruptedException
   c. public final void join(long ms,int ns) throws InterruptedException
3. sleep();
   If a Thread don't want to perform any operation for a particular amount of time then we should go for sleep() method.
   a. public static native void sleep(long ms) throws InterruptedException
   b. public static void sleep(long ms,int ns)throws InterruptedException