**Multithreading – Part-02 – Create Thread Using Thread Class**

**Note: Thread is a separate flow of execution and each Thread will have a job.**

* Defining a Thread:
  + We can define a thread in the following two ways.
    - By extending Thread class.
    - By implementing Runnable interface.
* Extending Thread class.

class MyThread extends Thread{

public void run(){

for(int i=0; i<10; i++){

System.out.println(“Child Thread”);

}

}

}

Note:

Whole class definition is the definition of the Thread.

run() method defines the “Job” of the Thread.

class ThreadDemo{

public static void main(String[] args){

// Till here there will be only one thread which is main Thread.

MyThread thread = new MyThread(); // Thread instantiation.

thread.start(); // After this line, there will be two threads, main and child thread.

for(int i = 0; i<10; i++){ // Job of main thread

System.out.println(“Main Thread”);

}

}

}

Case\_01:

Thread Scheduler:

It is the part of JVM, it is responsible to schedule threads that is, if multiple threads are waiting to get the chance of execution then in which order threads will be executed is decided by Thread Scheduler.

We can’t expect exact algorithm followed by thread scheduler, it is varied from JVM to JVM. Hence we can’t expect threads execution order and exact output.

Hence whenever situation comes to multithreading there is no guarantee for exact output, but we can provide several possible outputs.

The following are various possible outputs for the above program.

P1: P2: P3: P4:

Main thread Child Thread Main Thread Child Thread

Maint thread Child Thread Child Thread Main Thread

. . . Child Thread

. . . Main Thread

. . . .

Child Thread Main Thread .

Child Thread Main Thread

. .

. .

Case\_02:

Difference between t.start() and t.run()

In the case of t.start() a new Thread will be created which is responsible for the execution for run() method.

But in the case of t.run() a net Thread won’t be created and run method will be executed just like a normal method call by main Thread.

Hence in the above program if we replace t.start() with t.run() then output is.

Child Thread 10 times, followed by Main Thread 10 times. This total output produced by only main Thread.

Case\_03:

Importance of Thread class start() method:

Thread class start method is responsible to register the Thread with Thread Scheduler and all other mandatory activities. Hence without executing Thread class start() there is no chance of starting a new Thread in Java. Due to this Thread class start() method is considered as heart of multithreading.

start(){

1. Register this thread with thread scheduler
2. Perform all other mandatory activities.
3. Invoke run();

}

Case\_04:

Overloading of run()

Overloading of run() is always possible but Thread class start method can invoke no-argument run(), the other overloaded method we have to call explicitly like a normal method call.

class MyThread extends Thread{

public void run(){

System.out.println(“no-arg run”);

}

public void run(int i){

System.out.println(“int-arg run”);

}

}

class Test{

public static void main(String[] args){

MyThread thread = new MyThread();

thread.start();

}

}

Output: no-arg run.

* Case\_05:

If we are not overriding run method then Thread class run method will be executed. Which has empty implementation. Hence we won’t get any output.

class MyThread extends Thread{

}

class Test{

public static void main(String[] args){

MyThread t = new MyThread();

t.start();

}

}

Output: No output.

Note:

It is highly recommended to override run method otherwise don’t go for multithreading concept.

* Case\_06:

Overriding of start method:

If we override start method, then our start method will be executed just like a normal method call and new thread won’t be created.

class MyThread extends Thread{

pubic void start(){

System.out.println(“start method”);

}

public void run(){

System.out.println(“run method”);

}

}

class Test{

public static void main(String[] args){

MyThread t = new MyThread();

t.start();

System.out.println(“main method”);

}

}

Output: start method

main method

produced by only main Thread.

Note:

It is not recommended to override start method otherwise don’t go for multithreading concept.

* Case\_07:

class MyThread extends Thread{

public void start(){

super.start();

System.out.println(“start method”);

}

public void run(){

System.out.println(“run method”);

}

}

class Test{

public static void main(String[] args){

MyThread t = new MyThread();

t.start();

System.out.println(“main method”);

}

}

Output:

P1 P2 P3

run method start method start method

start method main method run method

main method run method main method

* Case\_08:

Thread life cycle:

Simple life cycle of a Thread:

MyThread t = new MyThread(); (New/ Born state)

|

|

t.start() (Ready/ Runnable)

|

| (If TS (Thread Schedular) allocated processor

Running

|

| if run() method completes

Dead

* Case\_09:

After starting a thread if we are trying to restart the same thread, then we will get runtime exception saying IllegalThreadStateException.

class Test{

public static void main(String[] args){

Thread t = new Thread();

t.start();

System.out.println(“main method”);

t.start();

}

}

Output:

Main method.

RE: IllegalThreadStateException