**Multi Threading**

**Agenda:**

1. Introduction
2. The ways to define instantiate and start a new Thread
3. Getting and setting name of a Thread
4. Thread prioritiess
5. The methods to prevent(stop) Thread execution

* yield()
* join()
* sleep()

1. Synchronization
2. Inter Thread communication
3. Deadlock
4. Daemon Threads

**Multitasking –** Executing several tasks simultaneously is the concept of multitasking. There are two types of multitasking’s.

1. Process based multitasking.
2. Thread based multitasking

**[1.] Process based multitasking –** Executing several tasks simultaneously where each task is a separate independent process such type of multitasking is called process based multitasking.

Example – While typing a java program in the editor we can able to listen mp3 audio songs at the same time we can download a file from the net all these tasks are independent of each other and executing simultaneously hence it is Process based multitasking.

* This type of multitasking is best suitable at “os level”.

**[2.] Thread based multitasking –** Executing several tasks simultaneously where each task is a separate independent part of the same program, is called Thread based multitasking. And each independent part is called a “Thread”.

* This type of multitasking is best suitable for “programatic level”.
* When compared with “C++”, developing multithreading examples is very easy in java because java provides in built support for multithreading through a rich API (Thread, Runnable, ThreadGroup, ThreadLocal….etc).
* In multithreading on 10% of the work the programmer is required to do and 90% of the work will be down by java API.
* The main important application areas of multithreading are:

1. To implement multimedia graphics.
2. To develop animations.
3. To develop video games etc.

* Whether it is process based or Thread based the main objective of multitasking is to improve performance of the system by reducing response time.

**The ways to define instantiate and start a new Thread –** We can define a Thread in the following 2 ways.

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

**[1.] Defining a Thread by extending “Thread class”:**

Example –class MyThread extends Thread {

public void run() {

for(int i=0; i<5; i++) {

Defining a Thread

System.out.println("Child-thread "); Job of a Thread

}

}

}

class ThreadDemo{

public static void main(String ar[]) {

MyThread t = new MyThread(); //Instantiation of a Thread

t.start(); // Starting a Thread

for(int i=0; i<5; i++) {

System.out.println("Main-thread ");

}

}

}

**Case–1: Thread Scheduler**

* If multiple Threads are waiting to execute then which Thread will execute 1st is decided by “**Thread Scheduler**” which is part of JVM.
* Which algorithm followed by Thread Scheduler we can’t expect exactly it is the JVM vendor dependent, hence in multithreading examples we can’t expect exact execution order and exact output.
* The following are various **possible outputs** for the above program :–

**-----------------** More Output

|  |  |  |
| --- | --- | --- |
| P1 | P2 | P3 |
| Main-thread  Main-thread  Main-thread  Main-thread  Main-thread  Child-thread  Child-thread  Child-thread  Child-thread  Child-thread | Child-thread  Child-thread  Child-thread  Child-thread  Child-thread  Main-thread  Main-thread  Main-thread  Main-thread  Main-thread | Main-thread  Main-thread  Child-thread  Child-thread  Child-thread  Main-thread  Main-thread  Main-thread  Child-thread  Child-thread |

**Case–2: Difference between t.start() and t.run() methods**

* 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.
* In the above program if we are replacing t.start() with t.run() the following is the output.
* Entire output produced by only main Thread.

**Output:**– Child-thread

Child-thread

Child-thread

Child-thread

Child-thread

Main-thread

Main-thread

Main-thread

Main-thread

Main-thread

**Case–3: Importance of Thread class start() method**

* For every Thread the required mandatory activities like registering the Thread with Thread Scheduler will takes care by Thread class start() method and programmer is responsible just to define the job of the Thread inside run() method. That is start() method acts as best assistant to the programmer.
* We can conclude that without executing Thread class start() method there is no chance of starting a new Thread in java.

Example – start() {

1. Register Thread with Thread Scheduler

2. All other mandatory low level activities

3. Invoke or calling run() method

}

**Case–4: If we are not overriding run() method**

* 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.
* It is highly recommended to override run() method. Otherwise don’t go for multithreading concept.

Example – class MyThread extends Thread {

// not overriding run() method

}

class ThreadDemo {

public static void main(String[] args) {

MyThread t = new MyThread();

t.start();

}

}

**Case–5: Overriding of run() method**

* 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.

Example – class MyThread extends Thread {

public void run() {

System.out.println("no arg method");

}

public void run(int i) {

System.out.println("int arg method");

}

}

class ThreadDemo {

public static void main(String[] args) {

MyThread t = new MyThread();

t.start();

}

}

**Output:–** no arg method

**Case–6: Overriding of start() method**

* 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.’
* Entire output produced by only main Thread.

Example – class MyThread extends Thread {

public void start() {

System.out.println("start method");

}

public void run() {

System.out.println("run method");

}

}

class ThreadDemo {

public static void main(String[] args) {

MyThread t = new MyThread();

t.start();

System.out.println("main method");

}

}

**Output:–** start method

main method

**Case–7: Overriding of start() method and call super.start()**

Example – class MyThread extends Thread {

public void start() {

super.start();

System.out.println("start method");

}

public void run() {

System.out.println("run method");

}

}

class ThreadDemo {

public static void main(String[] args) {

MyThread t=new MyThread();

t.start();

System.out.println("main method");

}

}

**Output:–**

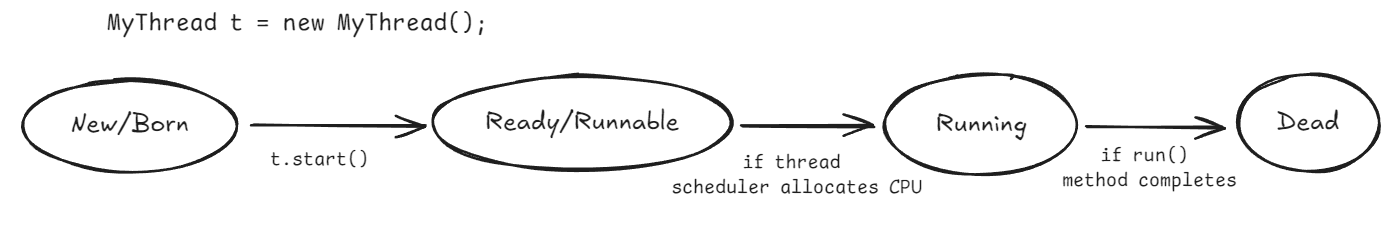
*child thread*  *main thread*

run method start method

main method

|  |  |
| --- | --- |
| start method  main method  run method | start method  run method  main method |

**Case–8: Life cycle of the Thread**

* Once we created a Thread object then the Thread is said to be in new state or born state.
* Once we call start() method then the Thread will be entered into Ready or Runnable state.
* If Thread Scheduler allocates CPU then the Thread will be entered into running state.
* Once run() method completes then the Thread will entered into dead state.

**Case–9:** After starting a Thread we are not allowed to restart the same Thread once again otherwise we will get runtime exception saying “IllegalThreadStateException”.

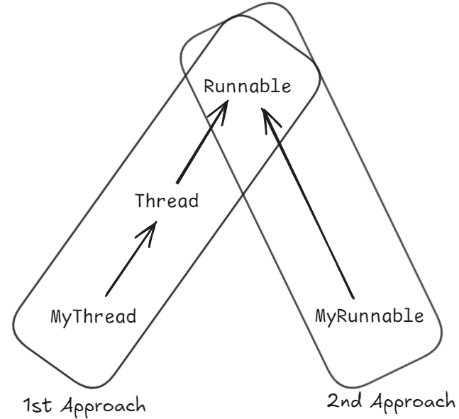
Example – MyThread t = new MyThread();

t.start(); //valid

- - - - - - - - - - - -

t.start(); //invalid ! we will get R.E saying:

**Output:-** Runtime exception – Exception in thread "main" java.lang.IllegalThreadStateException



**[2.] Defining a Thread by implementing “Runnable interface”:**

* We can define a Thread even by implementing Runnable interface also. Runnable interface present in java.lang.pkg and contains only one method run().
* We can’t expect exact output but there are several possible outputs.

Example –class MyRunnable implements Runnable {

public void run() {

for(int i=0; i<5; i++) {

Defining a Thread

System.out.println("Child-thread "); Job of a Thread

}

}

}

class ThreadDemo{

public static void main(String ar[]) {

MyRunnable r = new MyRunnable ();

Thread t = new Thread(r); // here r is a target Runnable

t.start(); // Starting a Thread

for(int i=0; i<5; i++) {

System.out.println("Main-thread ");

}

}

}

* The following are various **possible outputs** for the above program :–

**-----------------** More Output

|  |  |  |
| --- | --- | --- |
| P1 | P2 | P3 |
| Main-thread  Main-thread  Main-thread  Main-thread  Main-thread  Child-thread  Child-thread  Child-thread  Child-thread  Child-thread | Child-thread  Child-thread  Child-thread  Child-thread  Child-thread  Main-thread  Main-thread  Main-thread  Main-thread  Main-thread | Main-thread  Main-thread  Child-thread  Child-thread  Child-thread  Main-thread  Main-thread  Main-thread  Child-thread  Child-thread |

**Case study:**

MyRunnable r = new MyRunnable();

Thread t1 = new Thread();

Thread t2 = new Thread(r);

**Case–1:** **t1.start():**

* A new Thread will be created which is responsible for the execution of Thread class run()method.

**Case–2:** **t1.run ():**

**Case–3:** **t2.start():**

**Case–4:** **t2.run ():**

**Case–5:** r**.start():**

**Case–6:** r**.run ():**