**=Lab Sheet 5.1**: Create three threads by setting different priorities to each thread.

**Aim:** The aim of this program is to demonstrate how to work with thread priorities in Java. It creates three threads and sets their priorities to different values. The program also showcases the default priority of threads and how to set the priority of the main thread.

**Algorithm:**

1. Create a class **ThreadPrior** that extends the **Thread** class. This class will be used to create three threads.
2. Override the **run** method within the **ThreadPrior** class. In the **run** method, print a message, "Inside run method."
3. Create a class **TestThreadPrior** with the **main** method.
4. Inside the **main** method: a. Create three instances of the **ThreadPrior** class: **t1**, **t2**, and **t3**.

b. Print the default priority of each thread using the **getPriority** method.

c. Set the priorities of **t1**, **t2**, and **t3** using the **setPriority** method. The priorities are set to 2, 5, and 8, respectively.

d. Attempt to set the priority of **t3** to 21, which is not allowed, and would generate an error.

e. Print the updated priorities of **t1**, **t2**, and **t3**.

f. Print the currently executing thread's name using **Thread.currentThread().getName()**.

g. Print the default priority of the main thread using **Thread.currentThread().getPriority()**.

h. Set the priority of the main thread to 10 using **Thread.currentThread().setPriority(10)**.

i. Print the updated priority of the main thread.

**Program Explanation:**

1. Three threads, **t1**, **t2**, and **t3**, are created and their priorities are set to 2, 5, and 8, respectively.
2. The program attempts to set the priority of **t3** to 21, which is not a valid priority value. In Java, thread priorities must be in the range of 1 to 10. An error would occur for this line.
3. The program then prints the updated priorities of all three threads, which should reflect the changes made earlier.
4. It displays the name of the currently executing thread, which is the main thread.
5. The program prints the default priority of the main thread, which is usually 5.
6. The main thread's priority is set to 10, the maximum priority value allowed in Java.
7. Finally, the program prints the updated priority of the main thread, which should now be 10.

This program demonstrates the use of thread priorities and how to work with them in Java, while also highlighting the constraints on thread priority values.

Top of Form

**Program**:

class ThreadPrior extends Thread {

public void run()

{

// Print statement

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

}

}

public class TestThreadPrior {

public static void main(String[] args)

{

ThreadPrior t1 = new ThreadPrior();

ThreadPrior t2 = new ThreadPrior();

ThreadPrior t3 = new ThreadPrior();

System.out.println("t1 thread priority : "+ t1.getPriority());

System.out.println("t2 thread priority : "+ t2.getPriority());

System.out.println("t3 thread priority : "+ t3.getPriority());

t1.setPriority(2);

t2.setPriority(5);

t3.setPriority(8);

t3.setPriority(21); //error

System.out.println("t1 thread priority : " + t1.getPriority());

System.out.println("t2 thread priority : "+ t2.getPriority());

System.out.println("t3 thread priority : " + t3.getPriority());

// Main thread

System.out.println("Currently Executing Thread : "+Thread.currentThread().getName());

System.out.println(

"Main thread priority : "+ Thread.currentThread().getPriority());

// Main thread priority is set to 10

Thread.currentThread().setPriority(10);

System.out.println(

"Main thread priority : "+ Thread.currentThread().getPriority());

}

}

**OUTPUT:**

t1 thread priority : 5

t2 thread priority : 5

t3 thread priority : 5

t1 thread priority : 2

t2 thread priority : 5

t3 thread priority : 8

Currently Executing Thread : main

Main thread priority : 5

Main thread priority : 10

**Lab Sheet 5.2**: Demonstrate Thread Synchronization for a given resource to avoid race condition.

* Create a Resource class to keep two resources [ and ]. No thread can take ] without [
* Create three threads to access the above resource without synchronization
* Access the above resource using synchronization

**Aim:** The aim of this program is to demonstrate how to use multithreading in Java. It creates three threads that share a common resource object, and each thread tries to use the resource while being synchronized to avoid concurrent access.

**Algorithm:**

1. Create a class **Resource** with a method **use(String name)** that takes a name as a parameter. Inside this method:
   * Print the opening bracket "[" followed by the name.
   * Put the current thread to sleep for 1 second (simulating resource usage) using **Thread.sleep(1000)**.
   * Catch any **InterruptedException** that may occur while sleeping and print "Interrupted."
   * Print a closing bracket "]" after the sleep.
2. Create a class **MyThread** that extends the **Thread** class. This class will represent the threads that use the **Resource**.
   * Define instance variables **name** to store the thread's name and **r** to store the resource being used.
   * Create a constructor that takes **name** and **Resource r** as parameters, calls the superclass constructor with **name**, and initializes the instance variables.
   * Override the **run** method:
     + Synchronize on the **Resource** object (**r**) to ensure only one thread accesses the resource at a time.
     + Call the **use** method of the **Resource** object, passing the thread's name.
3. Create a class **TestMultiThread** with the **main** method.
   * Create an instance of the **Resource** class, **res**.
   * Create three instances of the **MyThread** class, **t1**, **t2**, and **t3**, passing their names and the **res** object as parameters.
   * Start al l three threads using **t1.start()**, **t2.start()**, and **t3.start()**.
   * Use **t1.join()**, **t2.join()**, and **t3.join()** to ensure that the main thread waits for the three threads (**t1**, **t2**, and **t3**) to complete their execution.

**Program Explanation:**

1. The program creates a **Resource** object called **res** and three **MyThread** objects named **t1**, **t2**, and **t3**. Each **MyThread** object is associated with the same **Resource** object.
2. When the threads (**t1**, **t2**, and **t3**) are started, they execute the **run** method. Inside the **run** method, they synchronize on the **Resource** object (**r**) to ensure that only one thread can access the **use** method at a time.
3. The **use** method of the **Resource** object prints the name of the thread enclosed in brackets and simulates resource usage by sleeping for 1 second. After sleeping, it prints the closing bracket.
4. The **main** thread in the **TestMultiThread** class starts the three threads and waits for them to finish by calling **join()** on each of them. This ensures that the main thread waits for all three threads to complete their execution before continuing.
5. As a result, the program demonstrates multithreading with shared resources and proper synchronization to avoid concurrent access to the resource, allowing each thread to use the resource in a controlled manner.

Top of Form

**Program:**

class Resource {

void use(String name) {

System.out.print("[" + name);

try {

Thread.sleep(1000);

} catch(InterruptedException e) {

System.out.println("Interrupted");

}

System.out.println("]");

}

}

class MyThread extends Thread {

String name;

Resource r;

MyThread (String name,Resource r){

super(name);

this.name = name;

this.r=r;

}

public void run() {

synchronized(r) {

r.use(name);

}

}

}

public class TestMultiThread {

public static void main(String args[]) {

Resource res=new Resource();

MyThread t1=new MyThread("1st",res);

MyThread t2=new MyThread("2nd",res);

MyThread t3=new MyThread("3rd",res);

t1.start();

t2.start();

t3.start();

try {

t1.join();

t2.join();

t3.join();

} catch (InterruptedException excetion) {

System.out.println("Interruption occurs in Main Thread");

}

}

}

**OUTPUT:**

[1st]

[3rd]

[2nd]