**Lab Sheet 4**.**1** Create a single thread by extending Thread class

**Aim:** The aim of this Java program is to demonstrate the creation and execution of a new thread by extending the **Thread** class. It illustrates how to create a custom thread class by extending **Thread** and overriding its **run()** method to execute a specific task in a separate thread, allowing for concurrent execution with the main thread.

**Algorithm:**

1. Create a custom class named **MyThread** that extends the **Thread** class.
2. Inside the **run()** method of the **MyThread** class, specify the actions the thread should perform. In this case, calculate the sum of two numbers (**a** and **b**), print a message indicating the thread has started, display the result, and indicate that the thread has completed.
3. Create a **TestThread** class with a **main** method.
4. Inside the **main** method:
   * Print a message indicating that the main thread has started.
   * Create an instance of the **MyThread** class and name it **t**.
   * Set the name of the **t** thread to "first" using the **setName** method.
   * Start the **t** thread using the **start()** method, which will invoke the **run()** method of the **MyThread** class.
   * Print a message indicating that the main thread has completed.

**Program Explanation:**

In this program, we create a custom thread class named **MyThread** by extending the **Thread** class. The **MyThread** class overrides the **run()** method to specify the tasks that the thread should perform. These tasks include:

1. Calculating the sum of two numbers, **a** and **b**.
2. Printing a message indicating that the thread has started running.
3. Displaying the result of the addition.
4. Printing a message indicating that the thread has completed its work.

In the **main** method of the **TestThread** class, we perform the following actions:

1. Print a message indicating that the main thread has started.
2. Create an instance of the **MyThread** class named **t**.
3. Set the name of the **t** thread to "first" using the **setName** method.
4. Start the **t** thread by calling the **start()** method. This initiates the execution of the **run()** method of the **MyThread** class in a separate thread.
5. Print a message indicating that the main thread has completed.

The key point to understand is that the tasks specified in the **run()** method of the **MyThread** class are executed concurrently with the main thread. This demonstrates how to create and run custom threads in Java by extending the **Thread** class.

Program:

Top of Form

class MyThread extends Thread {

// run() method to perform action for thread.

public void run()

{

int a= 10;

int b=12;

int result = a+b;

System.out.println(Thread.currentThread()+" started running..");

System.out.println("Sum of two numbers is: "+ result);

System.out.println(Thread.currentThread()+" completed..");

}

}

public class TestThread {

public static void main( String args[] )

{

System.out.println(Thread.currentThread()+" started");

// Creating instance of the class extend Thread class

MyThread t = new MyThread();

t.setName("first");

//calling start method to execute the run() method of the Thread class

t.start();

System.out.println(Thread.currentThread()+" completed");

}

}

**OUTPUT:**

Thread[main,5,main] started

Thread[main,5,main] completed

Thread[first,5,main] started running..

Sum of two numbers is: 22

Thread[first,5,main] completed..

**Lab Sheet 4.2:** Create 3 threads 1st, 2nd and 3rd to print numbers 5 to 1 concurrently by extending Thread Class.

Requirement:

* Override run() to print 5 to 1 using for loop
* Use sleep() for switching the context to other threads
* Use setName() to set the name of Thread or Use constructor Thread() to set the name of Thread

**Aim:** The aim of this Java program is to demonstrate the creation and execution of multiple threads, where each thread counts down from 5 to 1 at one-second intervals. The program shows how to create custom thread classes that extend **Thread**, set their names, and run them concurrently.

**Algorithm:**

1. Create a custom class named **MyThread** that extends the **Thread** class. Pass the name of the thread as a parameter to the constructor.
2. Inside the constructor, set the name of the thread using **setName(name)** or **super(name)**, and initialize the **name** field.
3. Print a message indicating that a new thread with the given name is created.
4. Implement the **run()** method to specify the tasks the thread should perform. In this case, count down from 5 to 1, printing the thread name and the current count, and sleep for one second in each iteration.
5. Handle exceptions if the thread is interrupted and print a message indicating that the thread is exiting.
6. In the **main** method of the **TestMultiThread** class:
   * Create three instances of the **MyThread** class with names "one," "two," and "three."
   * Start each thread using the **start()** method to execute the **run()** method concurrently.
   * Sleep the main thread for 8 seconds to allow the other threads to complete their tasks.
   * Handle exceptions if the main thread is interrupted and print a message indicating that the main thread is exiting.

**Program Explanation:**

In this program, we create a custom thread class named **MyThread** that extends the **Thread** class. Each instance of **MyThread** is initialized with a name passed to its constructor. The **run()** method of **MyThread** is implemented to count down from 5 to 1, printing the thread's name and the current count. Each iteration includes a one-second pause using **Thread.sleep(1000)**.

In the **main** method of the **TestMultiThread** class, we perform the following actions:

1. Create three instances of **MyThread** with names "one," "two," and "three."
2. Start each thread using the **start()** method, allowing them to run concurrently.
3. Sleep the main thread for 8 seconds to ensure that the other threads have enough time to complete their tasks.
4. Handle exceptions if the main thread is interrupted and print a message indicating that the main thread is exiting.

As a result, the program will display count-down messages from three threads ("one," "two," and "three") concurrently, and the main thread will wait for 8 seconds before exiting. This program demonstrates the concurrent execution of multiple threads with custom names and actions.

**Top of Form**

**Program:**

class MyThread extends Thread {

## String name;

## MyThread (String name){

## setName(name); or // super(name);

## this.name=name;

## System.out.println( "A New thread: " + name + "is created\n" );

## }

## public void run() {

## try {

## for(int j = 5; j > 0; j--) {

## System.out.println(name + ": " + j);

## Thread.sleep(1000);

## }

## }catch (InterruptedException e) {

## System.out.println(name + " thread Interrupted");

## }

## System.out.println(name + " thread exiting.");

## }

## }

## public class TestMultiThread {

## public static void main(String args[]) {

## MyThread t1=new MyThread(“one”);

## MyThread t2=new MyThread(“two”);

## MyThread t3=new MyThread(“three”);

## t1.start();

## t2.start();

## t3.start();

## try {

## Thread.sleep(8000);

## } catch (InterruptedException excetion) {

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

## }

## System.out.println("We are exiting from Main Thread");

## }

## }

## Output:

A New thread: oneis created

A New thread: twois created

A New thread: threeis created

one: 5

two: 5

three: 5

one: 4

two: 4

three: 4

one: 3

two: 3

three: 3

two: 2

one: 2

three: 2

two: 1

one: 1

three: 1

one thread exiting.

two thread exiting.

three thread exiting.

We are exiting from Main Thread

**Lab Sheet 4.3:**Create 3 threads 1st, 2nd and 3rd to print factorial of three different numbers concurrently by extending Thread Class.

Requirement:

* Override run() to print factorial using for loop
* Use sleep() for switching the context to other threads
* Use constructor Thread() to set the name of Thread
* Demonstrate join() and isAlive() method

**Aim:** The aim of this Java program is to demonstrate the creation and execution of multiple threads that calculate the factorial of a number. The program also showcases the use of the **join** method to wait for the threads to finish their tasks before the main thread exits.

**Algorithm:**

1. Create a custom class named **MyThread** that extends the **Thread** class. Pass an integer **number** and a **name** to the constructor.
2. In the constructor, call the **super(name)** constructor of the **Thread** class to set the thread's name, and initialize other instance variables like **number** and **name**.
3. Print a message indicating that a new thread with the given name is created.
4. Implement the **run()** method to calculate the factorial of the given **number**. During the calculation, print messages indicating the progress and sleep for one second in each iteration.
5. Handle exceptions if the thread is interrupted and print a message indicating that the thread is exiting after calculating the factorial.

In the **main** method of the **TestMultiThread** class:

1. Create three instances of **MyThread** with different numbers and names ("one," "two," and "three").
2. Start each thread using the **start()** method to execute the **run()** method concurrently.
3. Check and print the status of the threads using the **isAlive()** method before they have completed.
4. Use the **join()** method to wait for each thread to complete its task.
5. Check and print the status of the threads using the **isAlive()** method after they have completed.
6. Handle exceptions if the main thread is interrupted and print a message indicating that the main thread is exiting.

**Program Explanation:**

In this program, we create a custom thread class named **MyThread** that extends the **Thread** class. Each **MyThread** instance calculates the factorial of a given number, and the **run()** method prints messages indicating the progress of the calculation. The **join()** method is used to ensure that the main thread waits for the completion of the child threads.

In the **main** method of the **TestMultiThread** class:

1. We create three instances of **MyThread** with different numbers and names ("one," "two," and "three").
2. We start each thread, allowing them to run concurrently.
3. We check and print the status of the threads using the **isAlive()** method before they have completed. This shows that the threads are still running.
4. We use the **join()** method for each thread to wait for them to complete their tasks. The **join()** method ensures that the main thread doesn't exit until the child threads have finished.
5. After the **join()** method calls, we check and print the status of the threads using the **isAlive()** method again. This time, the threads are no longer alive as they have completed their tasks.
6. We handle exceptions if the main thread is interrupted during the execution of the child threads and print a message indicating that the main thread is exiting.

The program demonstrates how multiple threads can execute concurrently, calculate factorials, and how the **join()** method can be used to coordinate their execution with the main thread.

Top of Form

**Program:**

class MyThread extends Thread {

String name;

int number;

long fact=1;

MyThread (int number,String name){

super(name); //calling Thread()

this.number=number;

this.name=name;

System.out.println( "A New thread: " + name + " is created\n" );

}

public void run() {

try {

for(int i = 1; i <= number; i++) {

System.out.println(name + " calculating factorial");

fact=fact\*i;

Thread.sleep(1000);

}

}catch (InterruptedException e) {

System.out.println(name + " thread Interrupted");

}

System.out.println(name + " calculated factorial "+fact);

} }

public class TestMultiThread {

public static void main(String args[]) {

MyThread t1=new MyThread(5,"one");

MyThread t2=new MyThread(4,"two");

MyThread t3=new MyThread(3,"three");

t1.start();

t2.start();

t3.start();

System.out.println("1st Alive : "+t1.isAlive());

System.out.println("2nd Alive : "+t2.isAlive());

System.out.println("3rd Alive : "+t3.isAlive());

try {

t1.join();

t2.join();

t3.join();

System.out.println("1st Alive : "+t1.isAlive());

System.out.println("2nd Alive : "+t2.isAlive());

System.out.println("3rd Alive : "+t3.isAlive());

} catch (InterruptedException excetion) {

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

}

System.out.println("We are exiting from Main Thread");

}

}

**Output**

A New thread: one is created

A New thread: two is created

A New thread: three is created

1st Alive : true

2nd Alive : true

3rd Alive : true

two calculating factorial

three calculating factorial

one calculating factorial

one calculating factorial

three calculating factorial

two calculating factorial

one calculating factorial

two calculating factorial

three calculating factorial

two calculating factorial

one calculating factorial

three calculated factorial 6

one calculating factorial

two calculated factorial 24

one calculated factorial 120

1st Alive : false

2nd Alive : false

3rd Alive : false

We are exiting from Main Thread