**Practical No. 21 and 22: Develop a program for implementation of multithreading operation Part-1 and Part-2.**

1. **Practical Significance:**

Multithreading technique in java helps to run multiple programs or a processes concurrently by utilizing the maximum CPU time. Multithreading technique is implemented by creating, declaring, extending, implementing by thread. Student will be able to implement different types of thread methods by assigning the priority to illustrate simultaneous execution of thread operation.

1. **Relevant Course Outcome:**

Develop programs using multithreading.

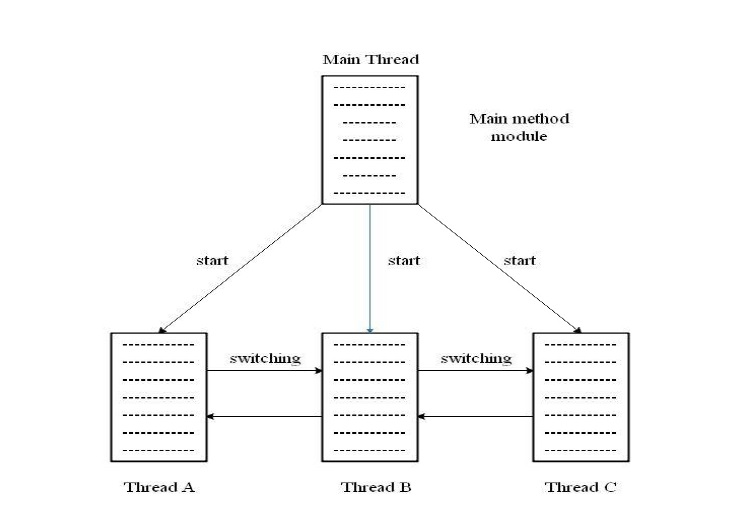
1. **Practical Outcome:**

Develop a program for implementation of multithreading operation.

1. **Minimum Theoretical Background:**

**Multithreading:** Multithreading is a small program (process) which is divided into two or more subprograms (processes), that can be implemented simultaneously.

**Thread:**  It is small process which is used to divide a program into number of sub parts and each part can be executed in parallel.



**Creating Threads**

Threads are implemented using a method called run().

**Syntax:**

Public void run()

{

……….

……….

}

**A thread can be created in two ways:**

1. **By creating a thread class:**

Define a class thar extends Thread class and override its run()

**Extending the thread Class**

Class MyThread extends Thread

{

……….

……….

}

1. **By converting a class to a thread:**

Define a class that implements Runnable interface. The Runnable interface has only one method, run()

class A implements Runnable

{

…………………

…………………

}

**Synchronization**

It means only a single thread can execute a block of code at the same time.

**Example:**

The method that will read information form a file and the method that will update the same file may be declared as synchronized.

Synchronized void update()

{

………………

……………… //synchronized code

}

**Java Thread Method:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.**  **No.** | **Type** | **Method** | **Description** |
| 1 | void | start() | It is used to start the execution of the thread. |
| 2 | void | run() | It is used to perform action for a thread. |
| 3 | static void | sleep() | It sleeps a thread for specified amount of time. |
| 4 | static thread | currentThread() | It returns a reference to the currently executing thread object. |
| 5 | void | join() | It waits for a thread to die. |
| 6 | int | getPriority() | It returns the priority of the thread. |
| 7 | void | setPriority() | It changes the priority of the thread. |
| 8 | string | getName() | It returns the name of the thread. |
| 9 | void | setName() | It changes the name of the thread. |
| 10 | long | getID() | It returns the id of the thread. |
| 11 | boolean | isAlive() | It tests if the thread is alive. |
| 12 | static void | yield() | It causes the currently executing thread object to temporarily pause, allow other threads to execute. |
| 13 | void | suspend() | It is used to suspend the thread. |
| 14 | void | resume() | It is used to resume the suspended thread |
| 15 | void | stop() | It is used to stop the thread. |
| 16 | void | destroy() | It is used to destroy thread group, all of its subgrp. |
| 17 | void | notify() | It is to give the notification for only one thread which is waiting for a particular object. |
| 18 | void | notifyAll() | It is used to give the notification to all waiting threads of a particular object. |

1. **Program Code:**

class AdmissionCounter extends Thread

{

public int available\_seats = 10;

synchronized void bookAdmission(String name, int number\_of\_Admission)

{

if(available\_seats >= number\_of\_Admission && number\_of\_Admission > 0)

{

available\_seats = available\_seats - number\_of\_Admission;

System.out.println("Hello " + name + " seats are available : " + number\_of\_Admission);

}

else

{

System.out.println("Admission is closed because seats are not available.");

}

}

}

class Admission

{

public static void main(String[] args)

{

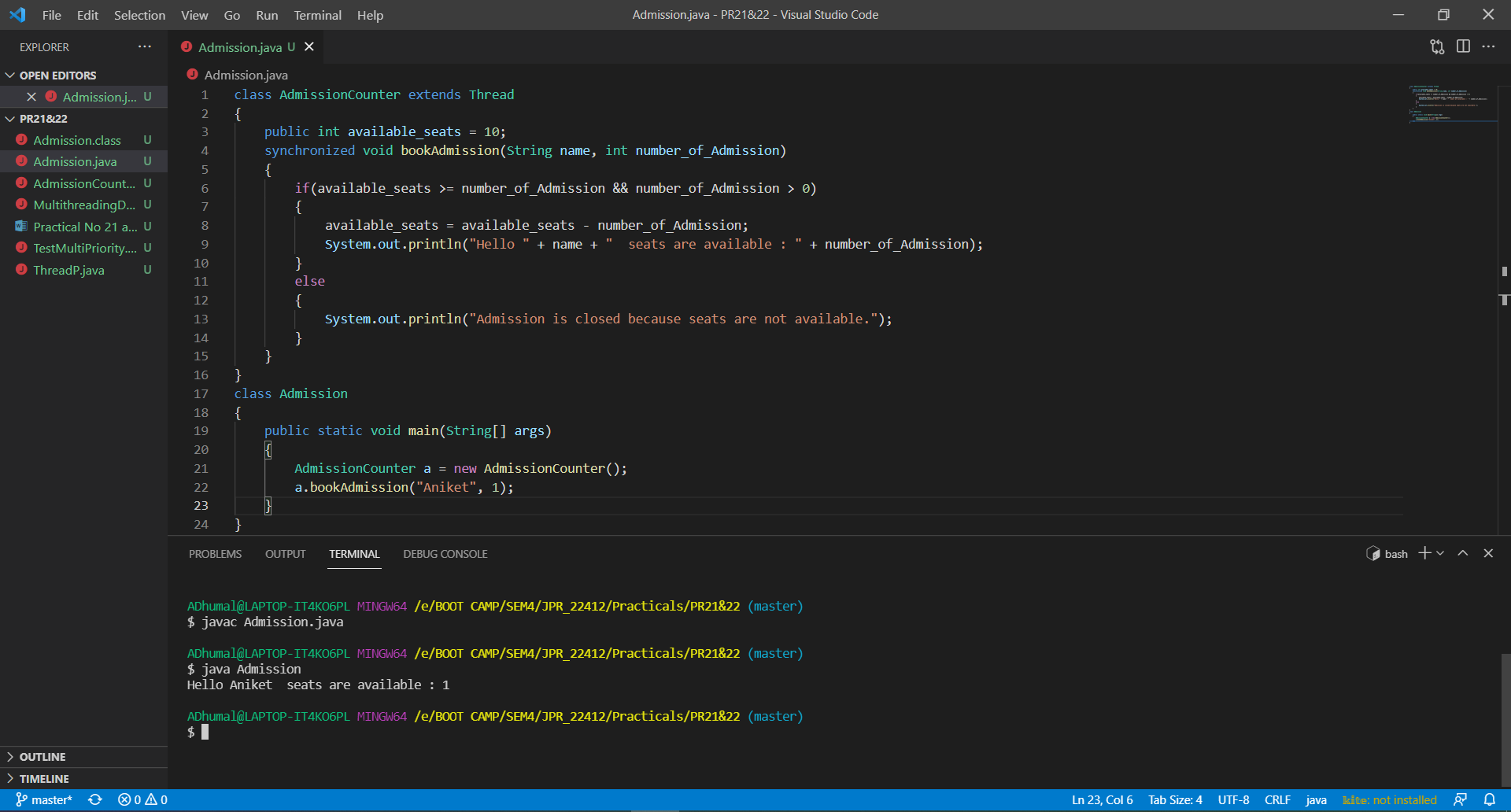
AdmissionCounter a = new AdmissionCounter();

a.bookAdmission("Aniket", 1);

}

}

1. **Result:**



1. **Practical Related Questions:**
2. **Is it possible to start a thread twice?**

No. After starting a thread, it can never be started again. In such case, thread will run once but for second time, it will throw exception.

1. **Can we call the run() method instead of start()?**

No, you cannot directly call run method to start a thread. If you call run method directly, it won't create a new thread and it will be in same stack as main.

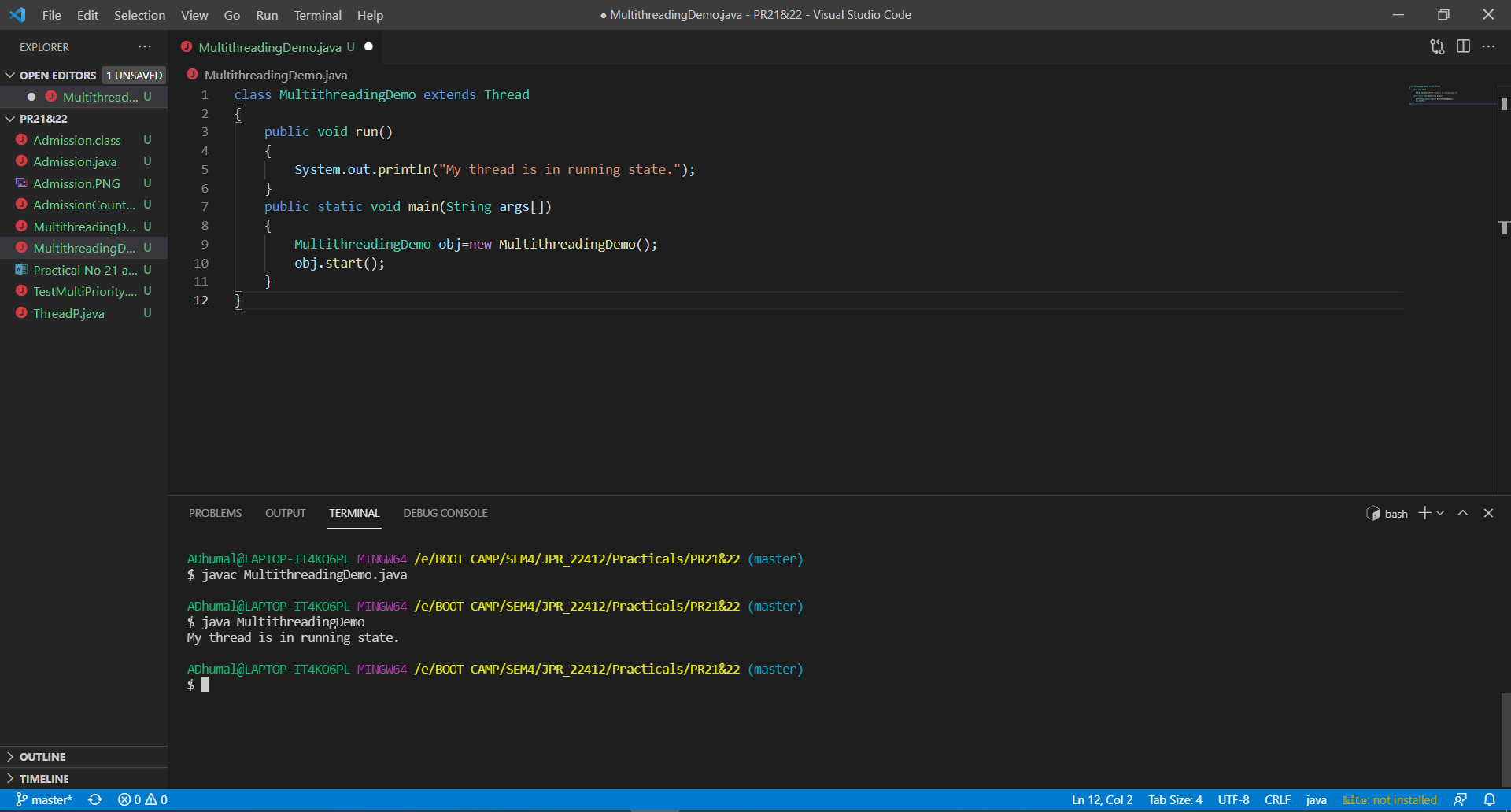
1. **Differentiate between notify() and notifyAll()?**

In case of multithreading notify() method sends the notification to only one thread among the multiple waiting threads which are waiting for lock. While notifyAll() methods in the same context sends the notification to all waiting threads instead of single one thread.

1. **Explain the use of keyword synchronized.**

To avoid such issues, Java provides us with the synchronized keyword, which acts like a lock to a particular resource. This helps achieve communication between threads such that only one thread accesses the synchronized resource and other threads wait for the resource to become free.

1. **Exercise:**
2. **Implement multithreading to perform simultaneous processes.**



1. **The code give below calls the run() method of two threads while setting their priority. Will this code compile successfully? If not, correct the code.**

class t1 extends Thread

{

public void run()

{

System.out.println("This is Thread1 class");

}

}

class t2 extends Thread

{

public void run()

{

System.out.println("This is Thread2 class");

}

}

public class ThreadP

{

public static void main(String args[])

{

t1 t = new t1();

t2 tt = new t2();

t.setPriority(Thread.MIN\_PRIORITY);

tt.setPriority(Thread.MIN\_PRIORITY);

t1.run();

t2.run();

}

class t1 extends Thread

{

public void run()

{

System.out.println("This is Thread1 class");

}

}

class t2 extends Thread

{

public void run()

{

System.out.println("This is Thread2 class");

}

}

public class ThreadP

{

public static void main(String args[])

{

t1 t = new t1();

t2 tt = new t2();

t.setPriority(Thread.MIN\_PRIORITY);

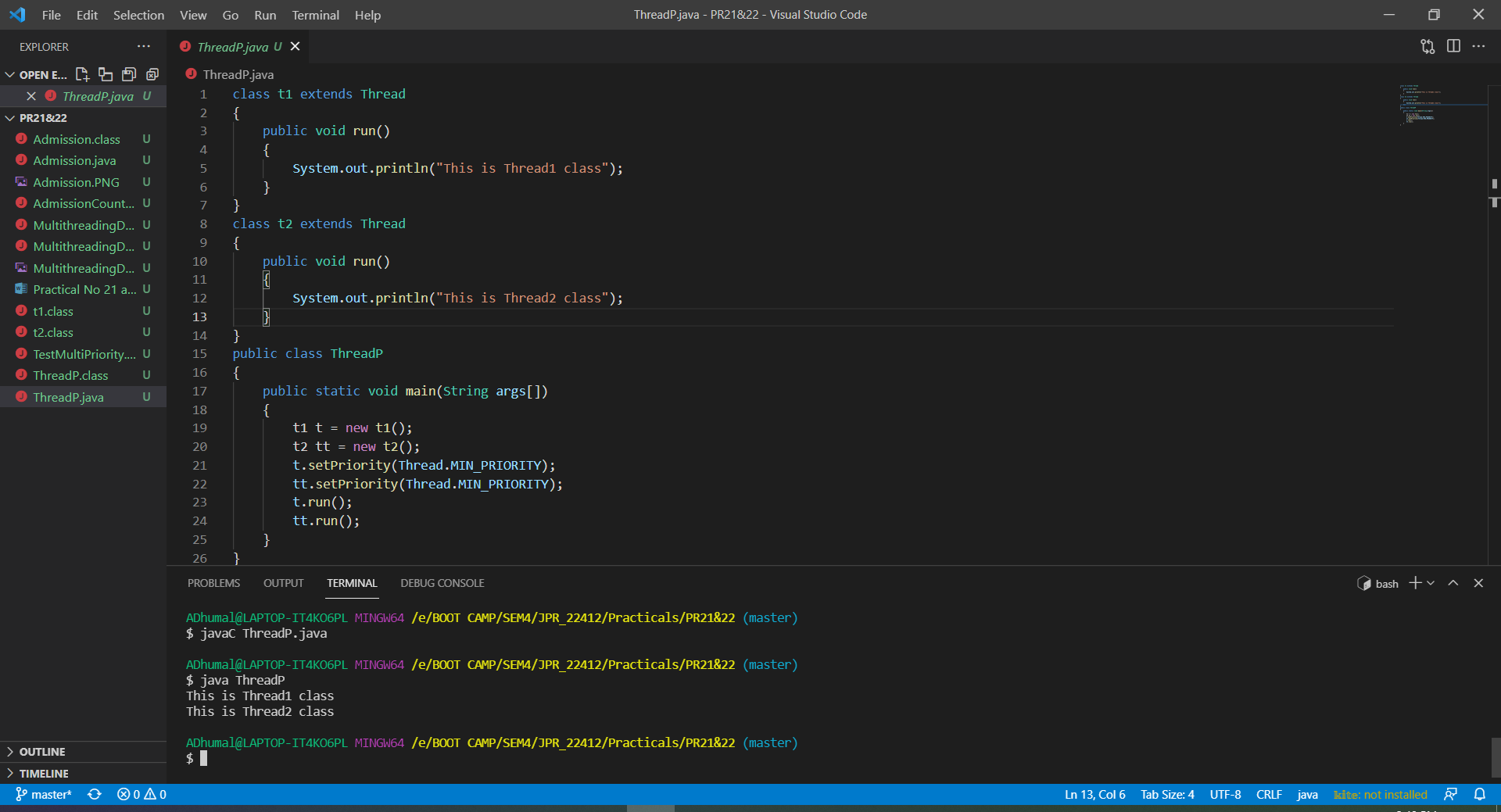
tt.setPriority(Thread.MIN\_PRIORITY);

t.run();

tt.run();

}

}



1. **Create three threads and run these threads according to set priority.**

class TestMultiPriority extends Thread

{

public void run()

{

System.out.println("running thread name is: " + Thread.currentThread().getName());

System.out.println("running thread priority is: " + Thread.currentThread().getPriority());

}

public static void main(String args[])

{

TestMultiPriority m1 = new TestMultiPriority();

TestMultiPriority m2 = new TestMultiPriority();

TestMultiPriority m3 = new TestMultiPriority();

m1.setPriority(Thread.MIN\_PRIORITY);

m1.start();

m2.setPriority(Thread.NORM\_PRIORITY);

m2.start();

m3.setPriority(Thread.MAX\_PRIORITY);

m3.start();

}

}

