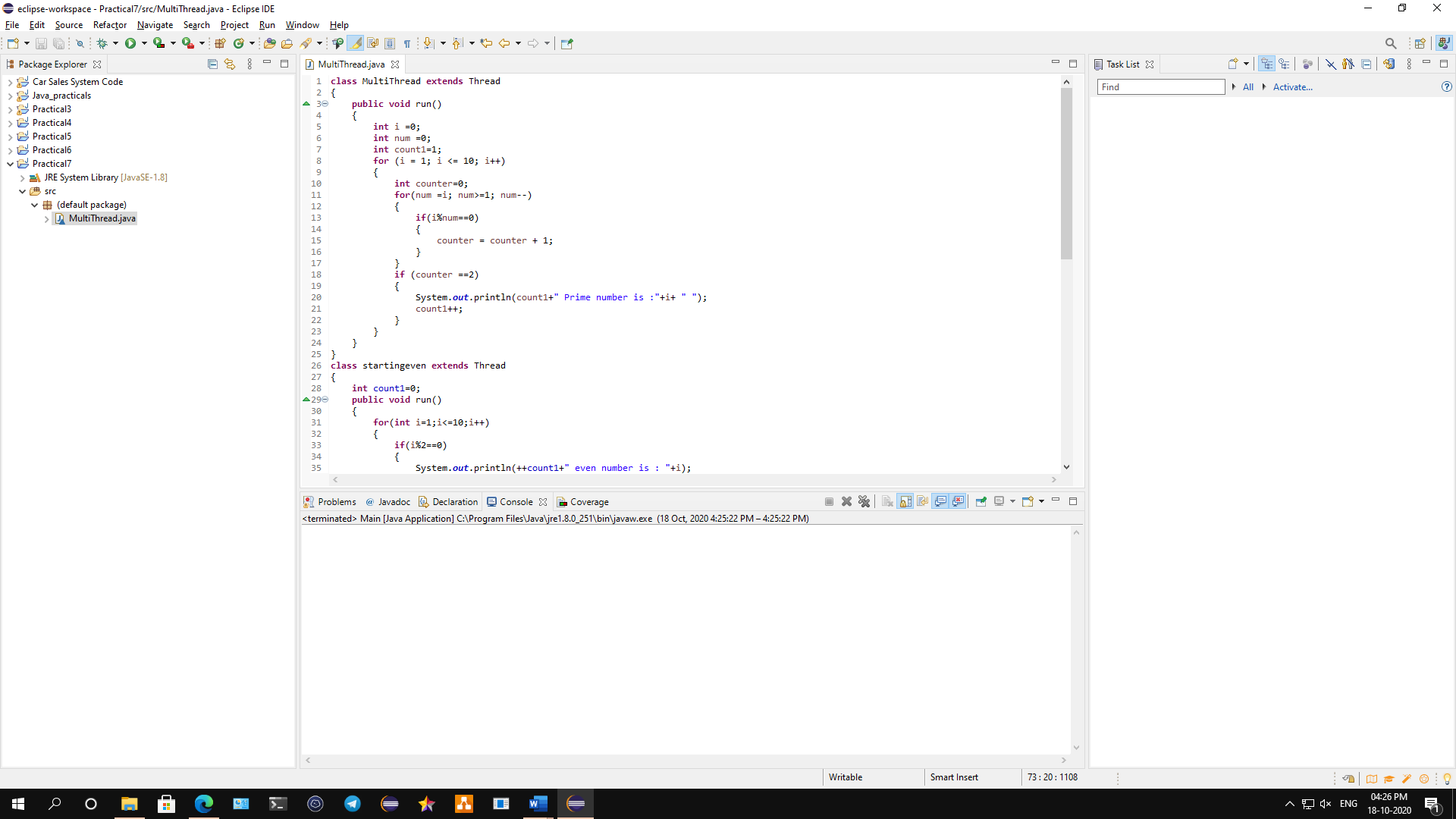
**Practical No 7**

**A) To implement a program to demonstrate multi-threading: even numbers between 1 to 100, Prime Numbers between 1 to 100 and Fibonacci Series.**

**Aim: Write a program to demonstrate multi-threading: even numbers between 1 to 100, Prime Numbers between 1 to 100 and Fibonacci Series.**

**Description:**

Multithreading in Java is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation, etc. In this program we have define the class multithread which extends thread classes. Then we use run as public method and we defined int as I, num and count. It will print the prime number. Then we defined stringeven class that extends thread which will also count the even number. Then we defined fibo class which will extend thread which will print the fib no followed by the main class and the output will be shown.



**Conclusion: We have written a program to demonstrate multi-threading: even numbers between 1 to 100, Prime Numbers between 1 to 100 and Fibonacci Series.**

**Code:**

**class** MultiThread **extends** Thread

{

**public** **void** run()

{

**int** i =0;

**int** num =0;

**int** count1=1;

**for** (i = 1; i <= 10; i++)

{

**int** counter=0;

**for**(num =i; num>=1; num--)

{

**if**(i%num==0)

{

counter = counter + 1;

}

}

**if** (counter ==2)

{

System.***out***.println(count1+" Prime number is :"+i+ " ");

count1++;

}

}

}

}

**class** startingeven **extends** Thread

{

**int** count1=0;

**public** **void** run()

{

**for**(**int** i=1;i<=10;i++)

{

**if**(i%2==0)

{

System.***out***.println(++count1+" even number is : "+i);

}

}

}

}

**class** fibo **extends** Thread

{

**long** a=0,b=1,c=0,n=1;

**public** **void** run()

{

**while**(a<10)

{

System.***out***.println(n+"th" +" Fib no: = "+a);

n++;

c=a+b;

a=b;

b=c;

**try**

{

**if**(a==21)

{

Thread.*sleep*(500);

}

}

**catch**(Exception e)

{}

}

}

}

**class** Main

{

**public** **static** **void** main(String[] args)

{

MultiThread sp = **new** MultiThread();

startingeven se = **new** startingeven();

fibo fb = **new** fibo();

sp.start();

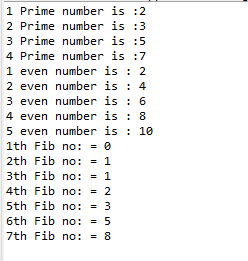
se.start();

fb.start();

}

}

**Output:**

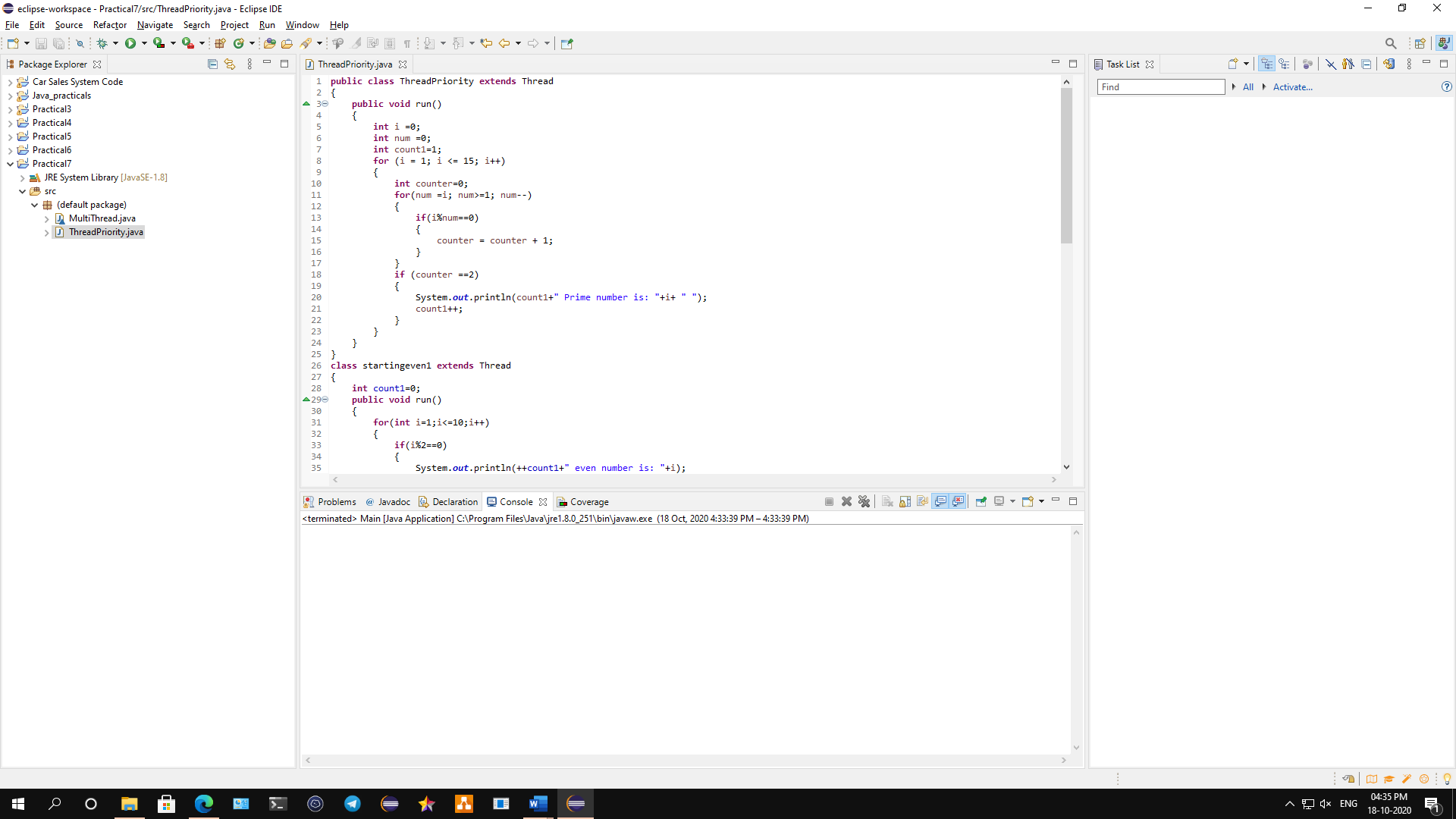


**B)** **To implement a program to demonstrate Thread Priority in above program.**

**Aim: Write a java program to implement a program to demonstrate Thread Priority in above program.**

**Description:**

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as pre-emptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses. 3 constants defined in Thread class are public static int MIN\_PRIORITY, public static int NORM\_PRIORITY, public static int MAX\_PRIORITY. Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10. In this program we have define the class multithread which extends thread classes. Then we use run as public method and we defined int as I, num and count. It will print the prime number. Then we defined stringeven class that extends thread which will also count the even number. Then we defined fibo class which will extend thread which will print the fib no. The we defined prior class which will show the default priority and current priority.



**Conclusion: We have implemented a program to demonstrate Thread Priority in above program.**

**Code:**

**public** **class** ThreadPriority **extends** Thread

{

**public** **void** run()

{

**int** i =0;

**int** num =0;

**int** count1=1;

**for** (i = 1; i <= 15; i++)

{

**int** counter=0;

**for**(num =i; num>=1; num--)

{

**if**(i%num==0)

{

counter = counter + 1;

}

}

**if** (counter ==2)

{

System.***out***.println(count1+" Prime number is: "+i+ " ");

count1++;

}

}

}

}

**class** startingeven1 **extends** Thread

{

**int** count1=0;

**public** **void** run()

{

**for**(**int** i=1;i<=10;i++)

{

**if**(i%2==0)

{

System.***out***.println(++count1+" even number is: "+i);

}

}

}

}

**class** fibo1 **extends** Thread

{

**long** a=0,b=1,c=0,n=1;

**public** **void** run()

{

**while**(a<40)

{

System.***out***.println(n+"th" +" Fib no: = "+a);

n++;

c=a+b;

a=b;

b=c;

**try**

{

**if**(a==21)

{

Thread.*sleep*(500);

}

}

**catch**(Exception e)

{}

}

}

}

**class** thread\_prior

{

**public** **static** **void** main(String[] args)

{

ThreadPriority sp = **new** ThreadPriority();

startingeven1 se = **new** startingeven1();

fibo1 fb = **new** fibo1();

sp.start();

System.***out***.println("Default priority of "+ sp.getName() + " : " + sp.getPriority());

se.start();

System.***out***.println("Default priority of "+ se.getName() + " : " + se.getPriority());

fb.start();

System.***out***.println("Default priority of "+ fb.getName() + " : " + fb.getPriority());

se.setPriority(Thread.***MAX\_PRIORITY***);

sp.setPriority(Thread.***MIN\_PRIORITY***);

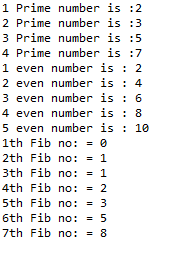
System.***out***.println("Current priority of "+ sp.getName() + " : " + sp.getPriority());

System.***out***.println("Current priority of "+ se.getName() + " : " + se.getPriority());

}

}

**Output:**

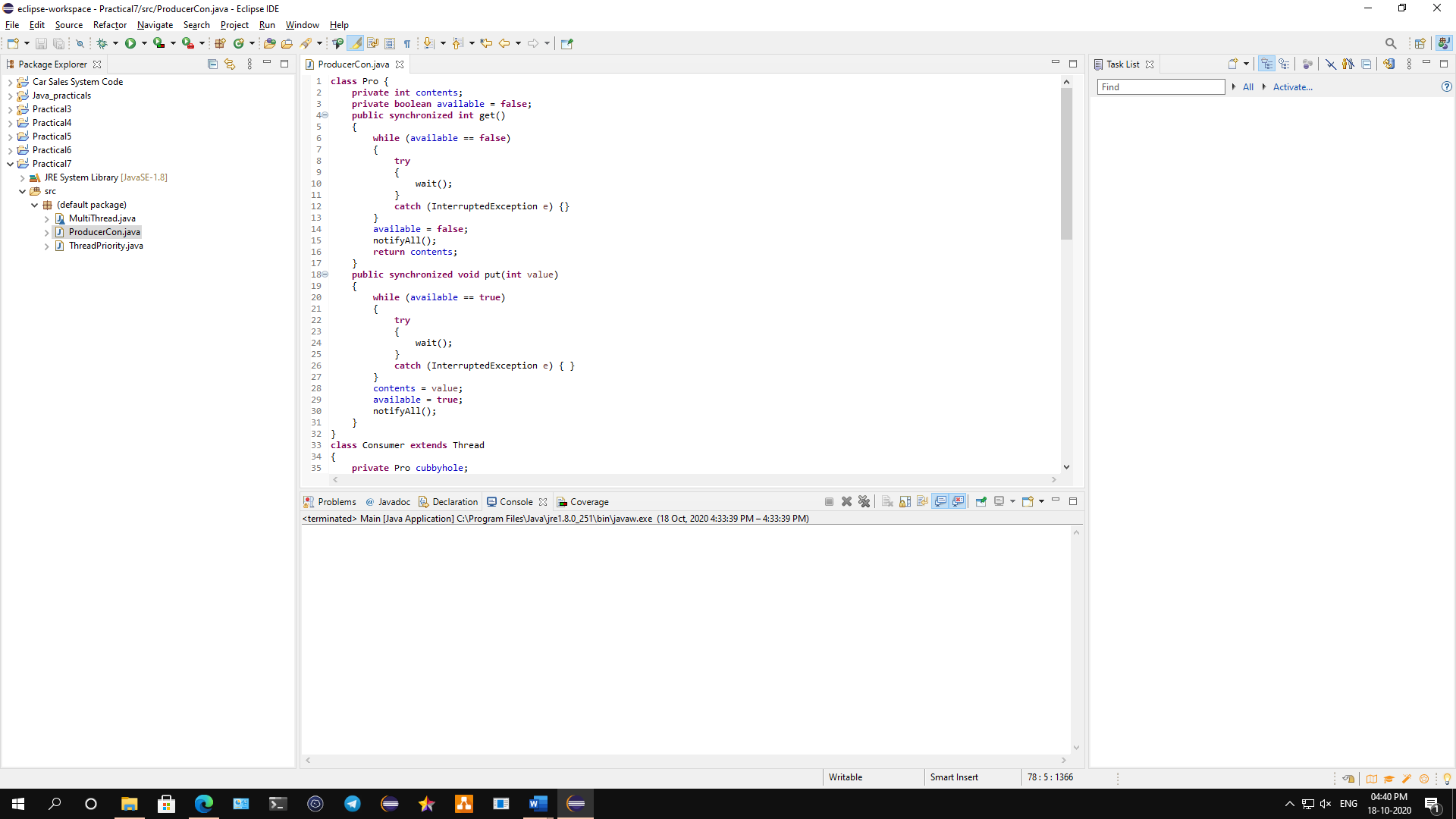


**C) To implement Multi-threaded Producer Consumer Application.**

**Aim: Write a java program to implement Multi-threaded Producer Consumer Application.**

**Description:**

In computing, the producer-consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue. The producer’s job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e. removing it from the buffer), one piece at a time. In this program we have defined pro class and the use private and public classes. Then we defined try block followed by catch block. Then we define class consumer which extends thread class followed by producer class and it main functions and the output will be displayed.



**Conclusion: We have implemented a program on Multi-threaded Producer Consumer Application.**

**Code:**

**class** Pro {

**private** **int** contents;

**private** **boolean** available = **false**;

**public** **synchronized** **int** get()

{

**while** (available == **false**)

{

**try**

{

wait();

}

**catch** (InterruptedException e) {}

}

available = **false**;

notifyAll();

**return** contents;

}

**public** **synchronized** **void** put(**int** value)

{

**while** (available == **true**)

{

**try**

{

wait();

}

**catch** (InterruptedException e) { }

}

contents = value;

available = **true**;

notifyAll();

}

}

**class** Consumer **extends** Thread

{

**private** Pro cubbyhole;

**private** **int** number;

**public** Consumer(Pro c, **int** number)

{

cubbyhole = c;

**this**.number = number;

}

**public** **void** run()

{

**int** value = 0;

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

{

value = cubbyhole.get();

System.***out***.println("Consumer #" + **this**.number + " got: " + value);

}

}

}

**class** Producer **extends** Thread

{

**private** Pro cubbyhole;

**private** **int** number;

**public** Producer(Pro c, **int** number)

{

cubbyhole = c;

**this**.number = number;

}

**public** **void** run()

{

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

{

cubbyhole.put(i);

System.***out***.println("Producer #" + **this**.number + " put: " + i);

**try**

{

*sleep*((**int**)(Math.*random*() \* 100));

}

**catch** (InterruptedException e) { }

}

}

}

**public** **class** ProducerCon

{

**public** **static** **void** main(String[] args)

{

Pro c = **new** Pro();

Producer p1 = **new** Producer(c, 1);

Consumer c1 = **new** Consumer(c, 1);

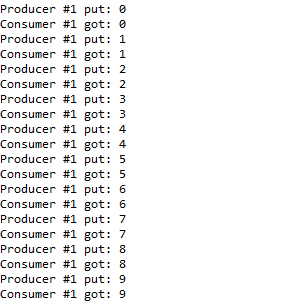
p1.start();

c1.start();

}

}

**Output:**

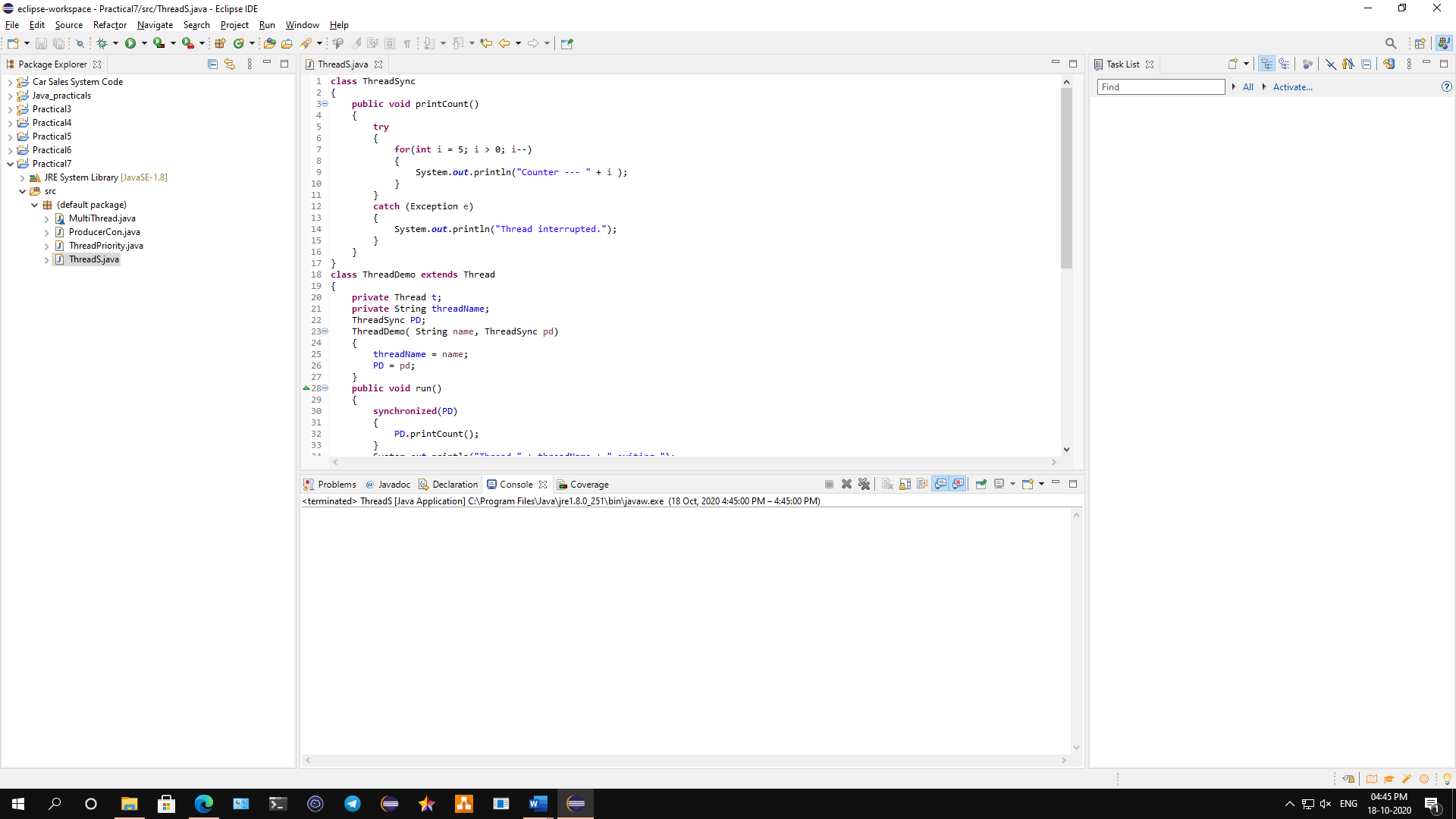


**D) Program on Thread Synchronization.**

**Aim: Write a java program to implement Thread Synchronization.**

**Description:**

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results. So, it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time. Java provides a way of creating threads and synchronizing their task by using synchronized blocks. Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block. We have defined ThreadSync class followed by counter and catch exception, it will throw the exception if the thread is interrupted. The class threaddemo will extend the thread class followed by public and private class. Then the main function will declared the output will be displayed.



**Conclusion: We have implemented a program on Thread Synchronization.**

**Code:**

**class** ThreadSync

{

**public** **void** printCount()

{

**try**

{

**for**(**int** i = 5; i > 0; i--)

{

System.***out***.println("Counter --- " + i );

}

}

**catch** (Exception e)

{

System.***out***.println("Thread interrupted.");

}

}

}

**class** ThreadDemo **extends** Thread

{

**private** Thread t;

**private** String threadName;

ThreadSync PD;

ThreadDemo( String name, ThreadSync pd)

{

threadName = name;

PD = pd;

}

**public** **void** run()

{

**synchronized**(PD)

{

PD.printCount();

}

System.***out***.println("Thread " + threadName + " exiting.");

}

**public** **void** start ()

{

System.***out***.println("Starting " + threadName );

**if** (t == **null**)

{

t = **new** Thread (**this**, threadName);

t.start ();

}

}

}

**public** **class** ThreadS

{

**public** **static** **void** main(String args[])

{

ThreadSync PD = **new** ThreadSync();

ThreadDemo T1 = **new** ThreadDemo( "Thread - 1 ", PD );

ThreadDemo T2 = **new** ThreadDemo( "Thread - 2 ", PD );

T1.start();

T2.start();

**try**

{

T1.join();

T2.join();

}

**catch** ( Exception e)

{

System.***out***.println("Interrupted");

}

}

}

**Output:**

