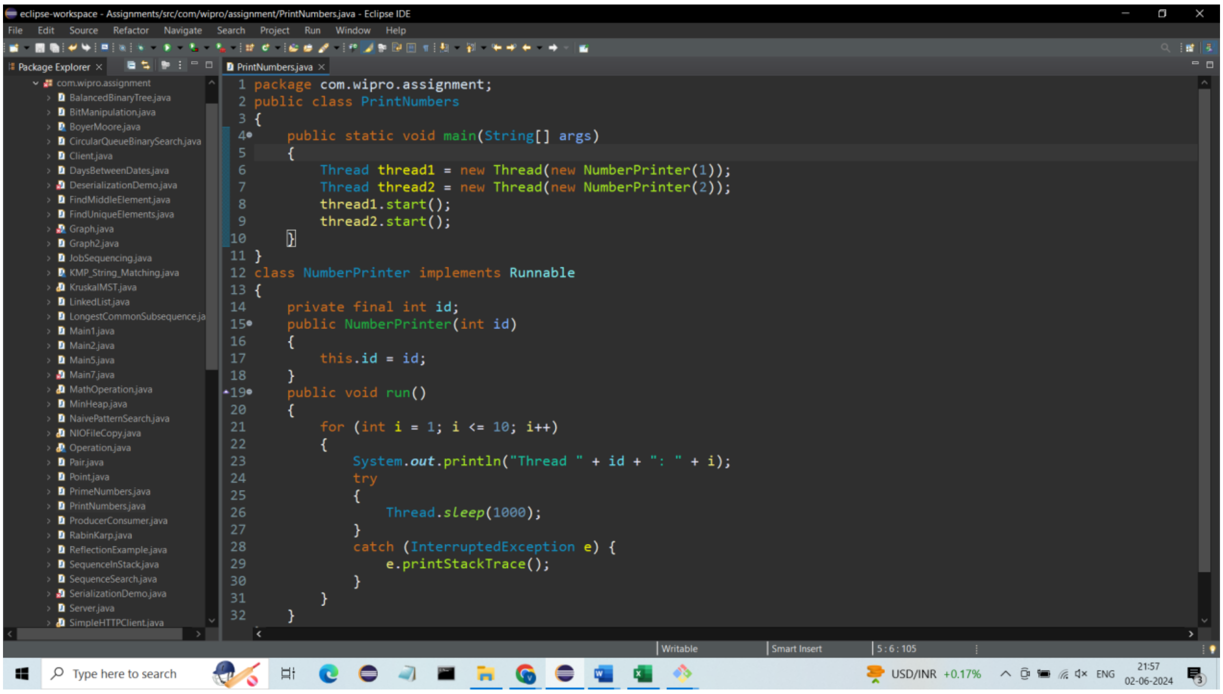
## Task 1: Creating and Managing Threads

Write a program that starts two threads, where each thread prints numbers from 1 to 10 with a 1-second delay between each number.

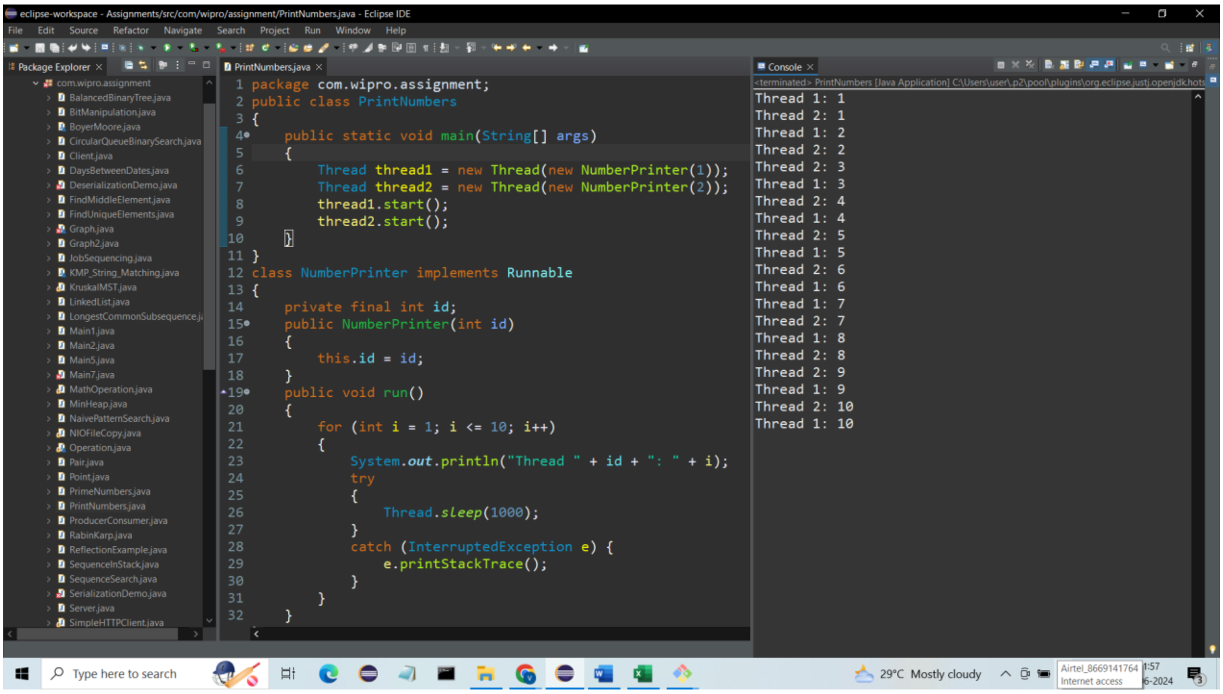
## Program: -

Here's the Java program that starts two threads, where each thread prints numbers from 1 to 10 with a 1second delay between each number:



```java
package com.wipro.assignment;
public class PrintNumbers
{
    public static void main(String[] args)
    {
        Thread thread1 = new Thread(new NumberPrinter(1));
        Thread thread2 = new Thread(new NumberPrinter(2));
        thread1.start();
        thread2.start();
    }
}
class NumberPrinter implements Runnable
{
    private final int id;
    public NumberPrinter(int id)
    {
        this.id = id;
    }
    public void run()
    {
        for (int i = 1; i <= 10; i++)
        {
            System.out.println("Thread " + id + ": " + i);
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

## Output: -



eclipse-workspace - Assignments/src/com/wipro/assignment/PrintNumbers.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
PrintNumbers.java
Console
<terminated> PrintNumbers [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hots
Type here to search
29°C Mostly cloudy

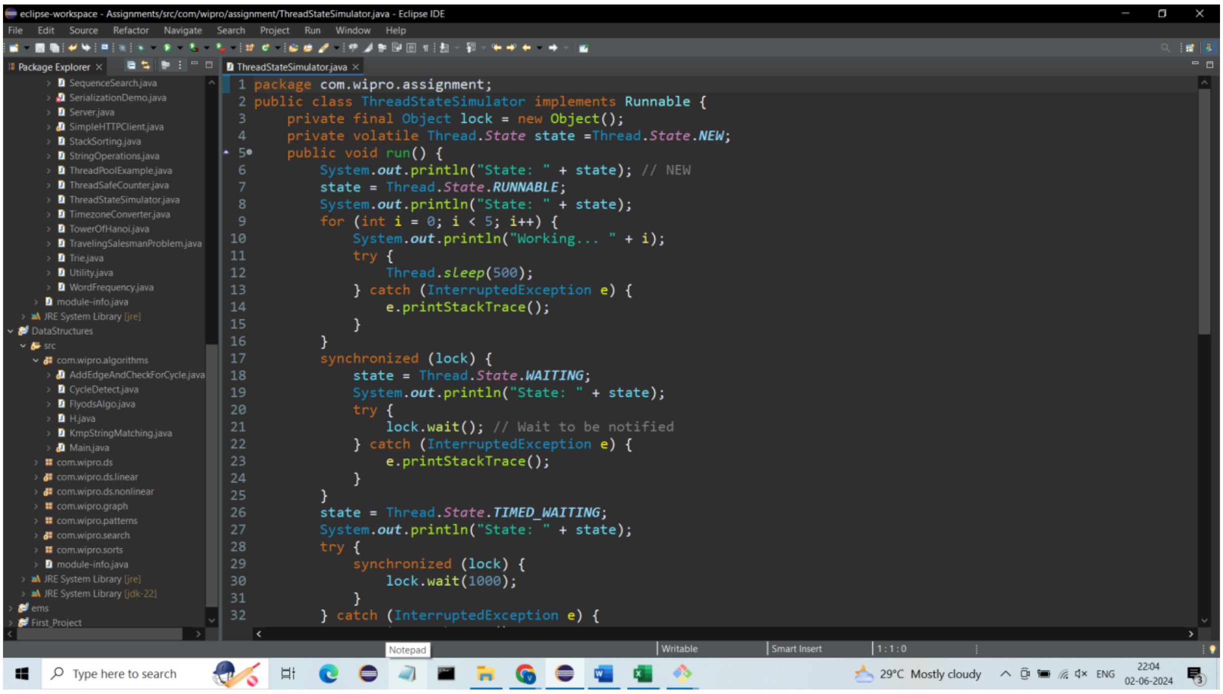
```java
package com.wipro.assignment;
public class PrintNumbers
{
    public static void main(String[] args)
    {
        Thread thread1 = new Thread(new NumberPrinter(1));
        Thread thread2 = new Thread(new NumberPrinter(2));
        thread1.start();
        thread2.start();
    }
}
class NumberPrinter implements Runnable
{
    private final int id;
    public NumberPrinter(int id)
    {
        this.id = id;
    }
    public void run()
    {
        for (int i = 1; i <= 10; i++)
        {
            System.out.println("Thread " + id + ": " + i);
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

```
Thread 1: 1
Thread 2: 1
Thread 1: 2
Thread 2: 2
Thread 2: 3
Thread 1: 3
Thread 2: 4
Thread 1: 4
Thread 2: 5
Thread 1: 5
Thread 2: 6
Thread 1: 6
Thread 1: 7
Thread 2: 7
Thread 1: 8
Thread 2: 8
Thread 2: 9
Thread 1: 9
Thread 2: 10
Thread 1: 10
```

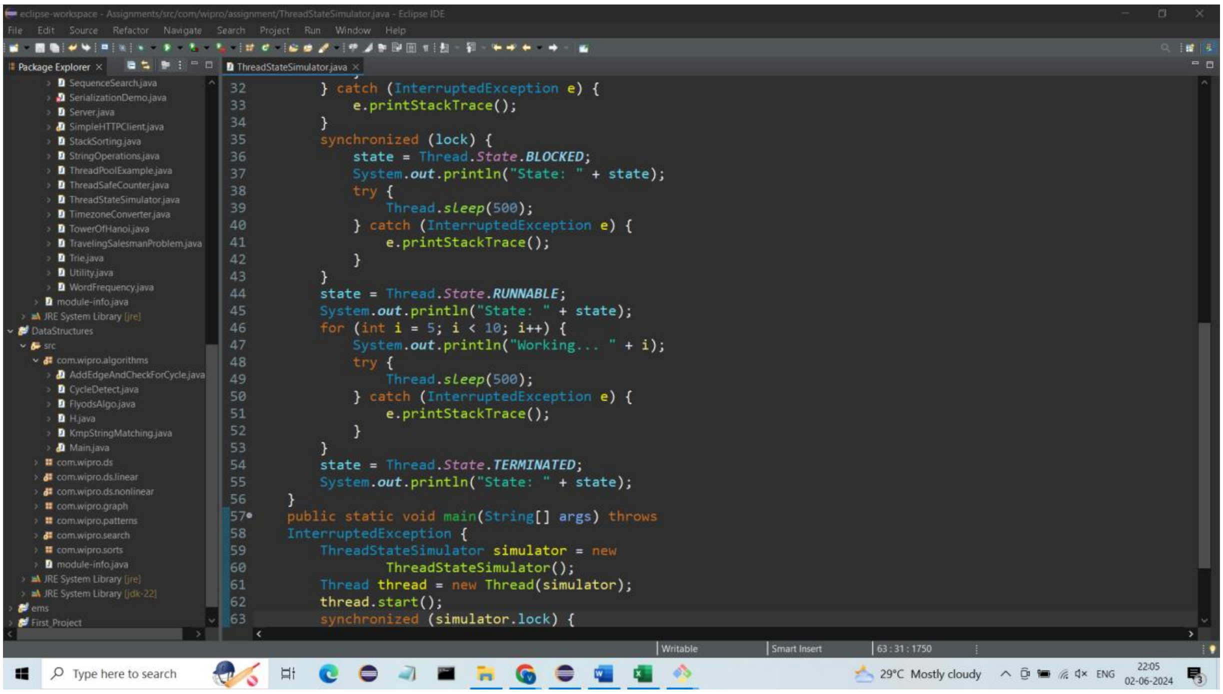
## Task 2: States and Transitions

Create a Java class that simulates a thread going through different lifecycle states: NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCKED, and TERMINATED. Use methods like sleep(), wait(), notify(), and join() to demonstrate these states..

## Java Class: -

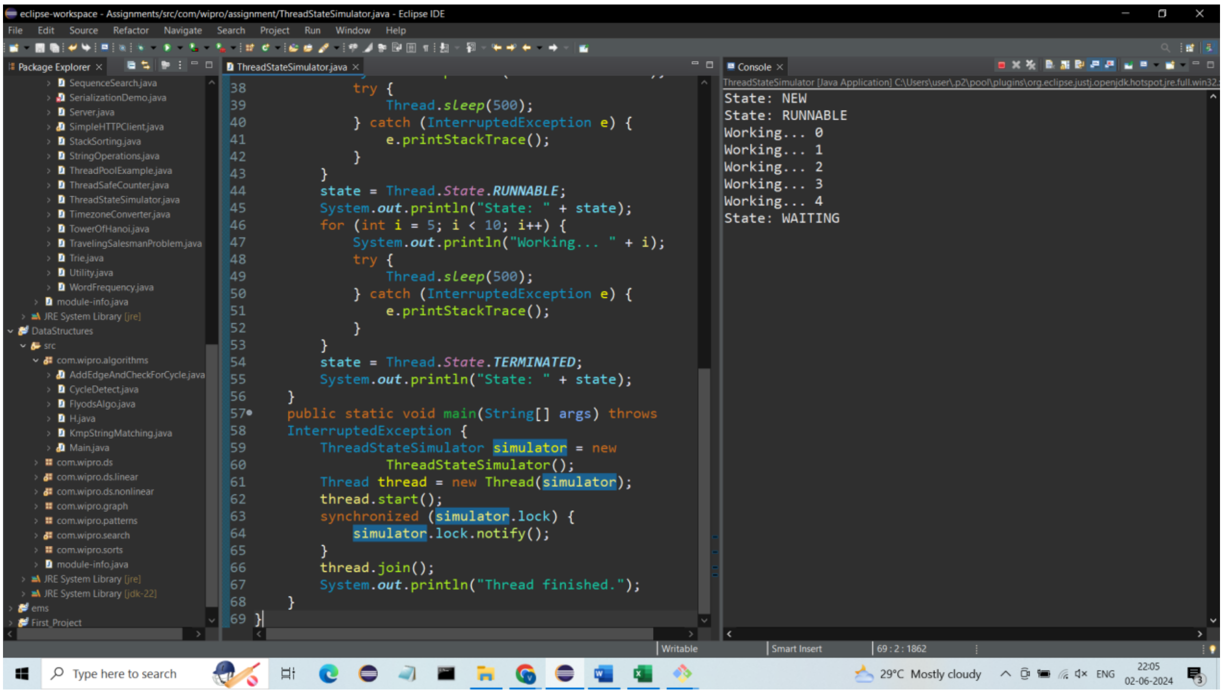

eclipse-workspace - Assignments/src/com/wipro/assignment/ThreadStateSimulator.java - Eclipse IDE
Package Explorer
ThreadStateSimulator.java
Writable
Smart Insert
1 : 1 : 0

```java
package com.wipro.assignment;
public class ThreadStateSimulator implements Runnable {
    private final Object lock = new Object();
    private volatile Thread.State state =Thread.State.NEW;
    public void run() {
        System.out.println("State: " + state); // NEW
        state = Thread.State.RUNNABLE;
        System.out.println("State: " + state);
        for (int i = 0; i < 5; i++) {
            System.out.println("Working... " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        synchronized (lock) {
            state = Thread.State.WAITING;
            System.out.println("State: " + state);
            try {
                lock.wait(); // Wait to be notified
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        state = Thread.State.TIMED_WAITING;
        System.out.println("State: " + state);
        try {
            synchronized (lock) {
                lock.wait(1000);
            }
        } catch (InterruptedException e) {
```


eclipse-workspace - Assignments/src/com/wipro/assignment/ThreadStateSimulator.java - Eclipse IDE
Package Explorer
ThreadStateSimulator.java
Writable
Smart Insert
63 : 31 : 1750

```java
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        synchronized (lock) {
            state = Thread.State.BLOCKED;
            System.out.println("State: " + state);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        state = Thread.State.RUNNABLE;
        System.out.println("State: " + state);
        for (int i = 5; i < 10; i++) {
            System.out.println("Working... " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        state = Thread.State.TERMINATED;
        System.out.println("State: " + state);
    }
    public static void main(String[] args) throws
    InterruptedException {
        ThreadStateSimulator simulator = new
                ThreadStateSimulator();
        Thread thread = new Thread(simulator);
        thread.start();
        synchronized (simulator.lock) {
```

# Output: -


eclipse-workspace - Assignments/src/com/wipro/assignment/ThreadStateSimulator.java - Eclipse IDE
ThreadStateSimulator.java
Console

```java
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        state = Thread.State.RUNNABLE;
        System.out.println("State: " + state);
        for (int i = 5; i < 10; i++) {
            System.out.println("Working... " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        state = Thread.State.TERMINATED;
        System.out.println("State: " + state);
    }
    public static void main(String[] args) throws
    InterruptedException {
        ThreadStateSimulator simulator = new
                ThreadStateSimulator();
        Thread thread = new Thread(simulator);
        thread.start();
        synchronized (simulator.lock) {
            simulator.lock.notify();
        }
        thread.join();
        System.out.println("Thread finished.");
    }
}
```

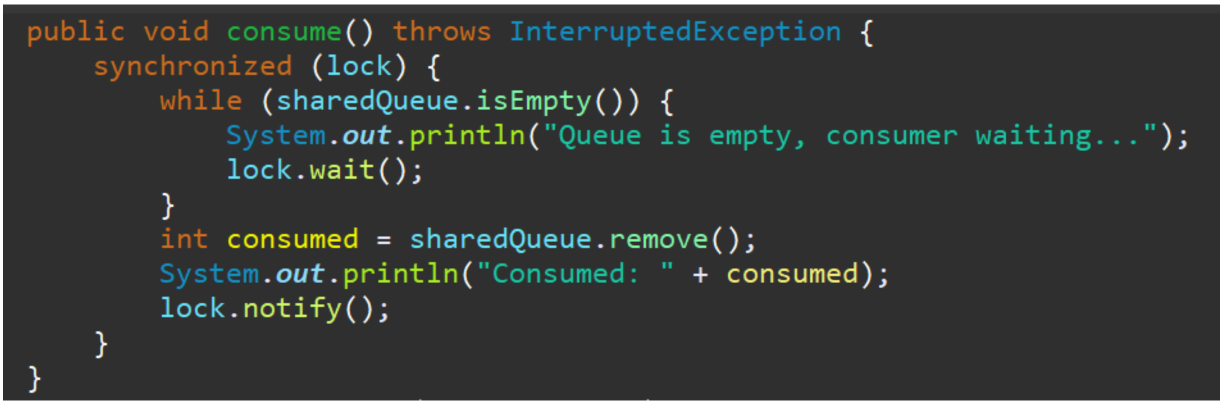
```
State: NEW
State: RUNNABLE
Working... 0
Working... 1
Working... 2
Working... 3
Working... 4
State: WAITING
```

## Task 3: Synchronization and Inter-thread Communication

Implement a producer-consumer problem using wait() and notify() methods to handle the correct processing sequence between threads.

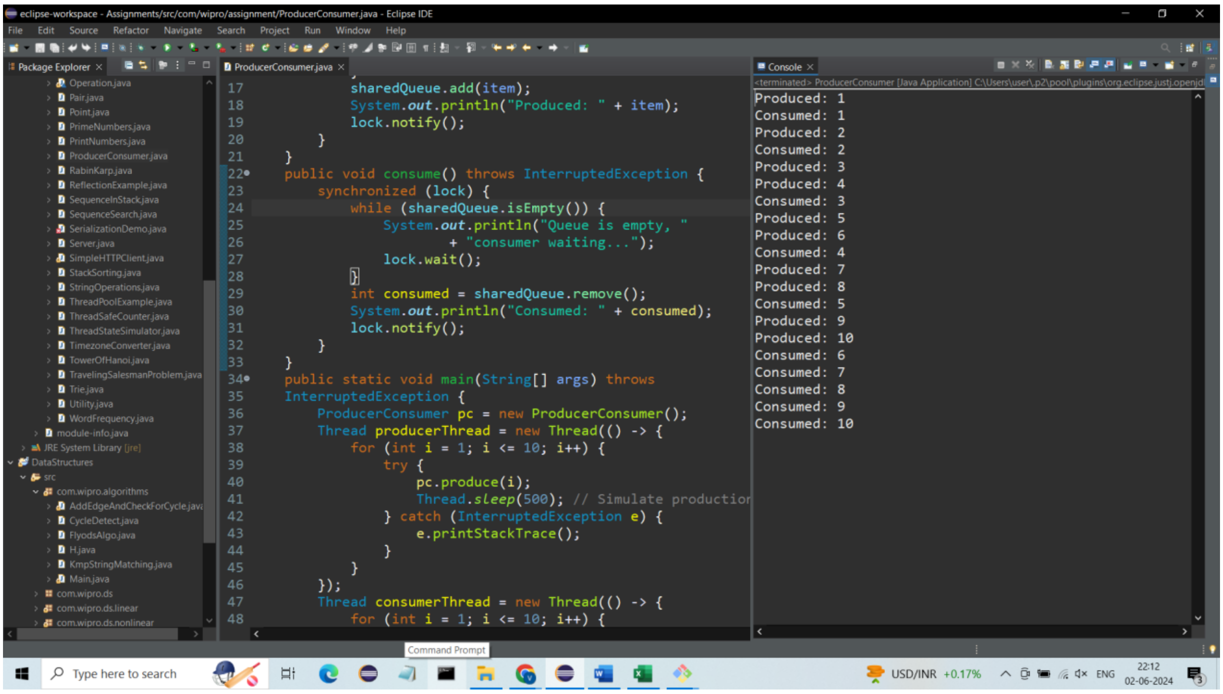
**Methods: -**

```java
public void produce(int item) throws InterruptedException {
    synchronized (lock) {
        while (sharedQueue.size() == MAX_SIZE) {
            System.out.println("Queue is full, producer waiting...");
            lock.wait();
        }
        sharedQueue.add(item);
        System.out.println("Produced: " + item);
        lock.notify();
    }
}
```



```
public void consume() throws InterruptedException {
    synchronized (lock) {
        while (sharedQueue.isEmpty()) {
            System.out.println("Queue is empty, consumer waiting...");
            lock.wait();
        }
        int consumed = sharedQueue.remove();
        System.out.println("Consumed: " + consumed);
        lock.notify();
    }
}
```

## Output: -



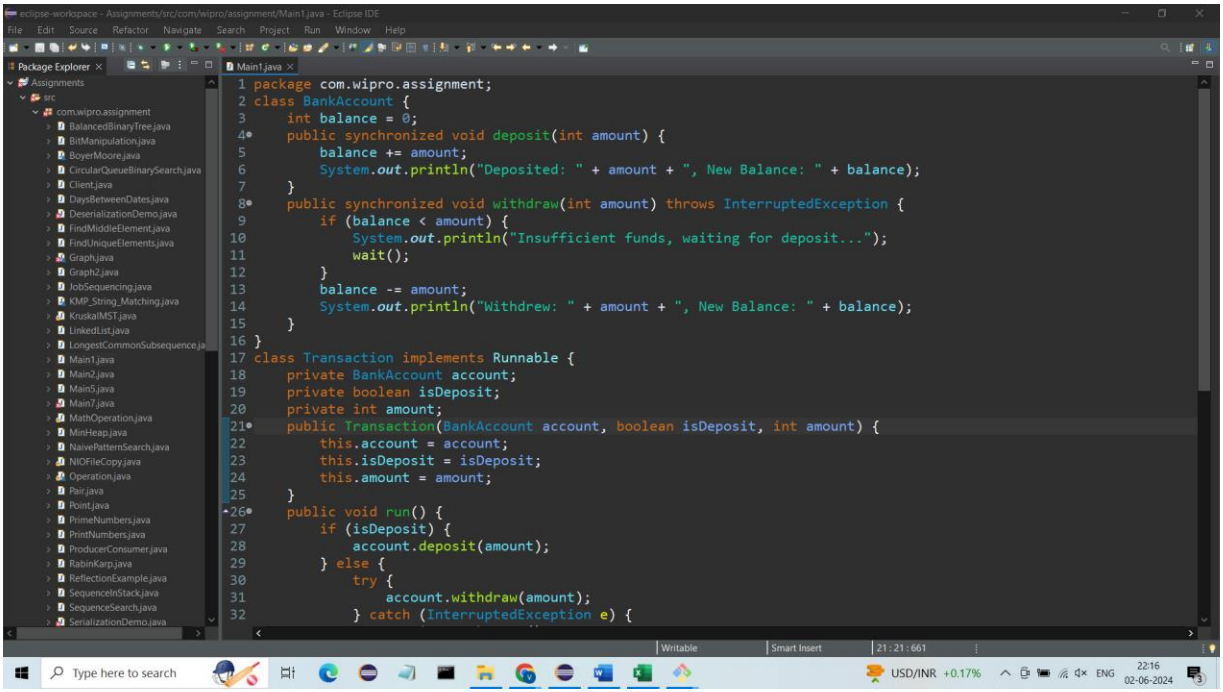
```
            sharedQueue.add(item);
            System.out.println("Produced: " + item);
            lock.notify();
        }
    }
    public void consume() throws InterruptedException {
        synchronized (lock) {
            while (sharedQueue.isEmpty()) {
                System.out.println("Queue is empty, "
                        + "consumer waiting...");
                lock.wait();
            }
            int consumed = sharedQueue.remove();
            System.out.println("Consumed: " + consumed);
            lock.notify();
        }
    }
    public static void main(String[] args) throws
    InterruptedException {
        ProducerConsumer pc = new ProducerConsumer();
        Thread producerThread = new Thread(() -> {
            for (int i = 1; i <= 10; i++) {
                try {
                    pc.produce(i);
                    Thread.sleep(500); // Simulate production
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        Thread consumerThread = new Thread(() -> {
            for (int i = 1; i <= 10; i++) {
```

```
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Produced: 4
Consumed: 3
Produced: 5
Produced: 6
Consumed: 4
Produced: 7
Produced: 8
Consumed: 5
Produced: 9
Produced: 10
Consumed: 6
Consumed: 7
Consumed: 8
Consumed: 9
Consumed: 10
```

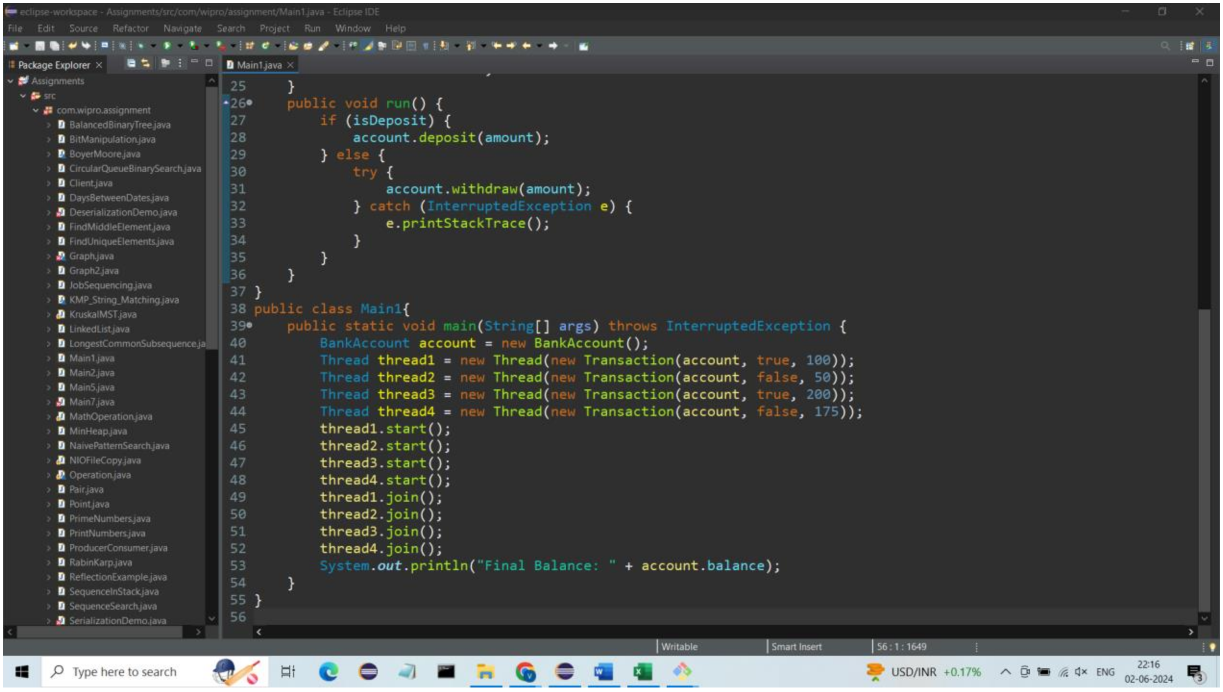
## Task 4: Synchronized Blocks and Methods

Write a program that simulates a bank account being accessed by multiple threads to perform deposits and withdrawals using synchronized methods to prevent race conditions.

## Program: -

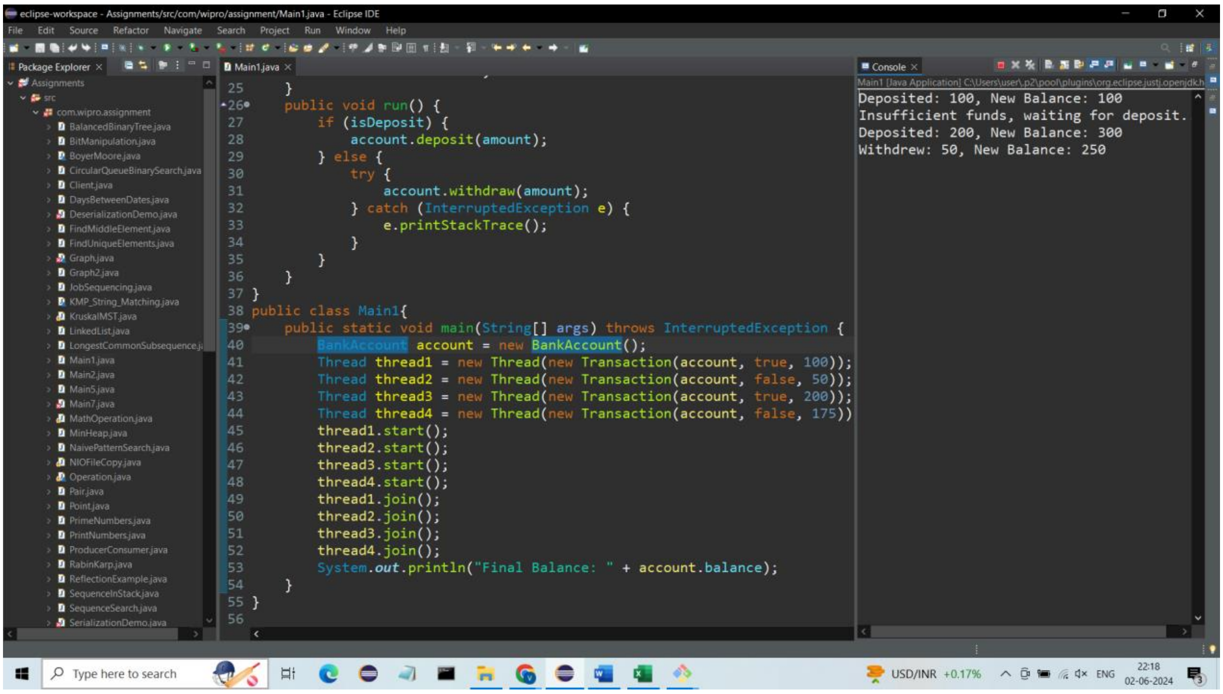

eclipse-workspace - Assignments/src/com/wipro/assignment/Main1.java - Eclipse IDE
Package Explorer
Main1.java
Writable
Smart Insert
21 : 21 : 661

```java
package com.wipro.assignment;
class BankAccount {
    int balance = 0;
    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println("Deposited: " + amount + ", New Balance: " + balance);
    }
    public synchronized void withdraw(int amount) throws InterruptedException {
        if (balance < amount) {
            System.out.println("Insufficient funds, waiting for deposit...");
            wait();
        }
        balance -= amount;
        System.out.println("Withdrew: " + amount + ", New Balance: " + balance);
    }
}
class Transaction implements Runnable {
    private BankAccount account;
    private boolean isDeposit;
    private int amount;
    public Transaction(BankAccount account, boolean isDeposit, int amount) {
        this.account = account;
        this.isDeposit = isDeposit;
        this.amount = amount;
    }
    public void run() {
        if (isDeposit) {
            account.deposit(amount);
        } else {
            try {
                account.withdraw(amount);
            } catch (InterruptedException e) {
```


eclipse-workspace - Assignments/src/com/wipro/assignment/Main1.java - Eclipse IDE
Package Explorer
Main1.java
Writable
Smart Insert
56 : 1 : 1649

```java
    }
    public void run() {
        if (isDeposit) {
            account.deposit(amount);
        } else {
            try {
                account.withdraw(amount);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
public class Main1{
    public static void main(String[] args) throws InterruptedException {
        BankAccount account = new BankAccount();
        Thread thread1 = new Thread(new Transaction(account, true, 100));
        Thread thread2 = new Thread(new Transaction(account, false, 50));
        Thread thread3 = new Thread(new Transaction(account, true, 200));
        Thread thread4 = new Thread(new Transaction(account, false, 175));
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        thread1.join();
        thread2.join();
        thread3.join();
        thread4.join();
        System.out.println("Final Balance: " + account.balance);
    }
}
```

# Output: -

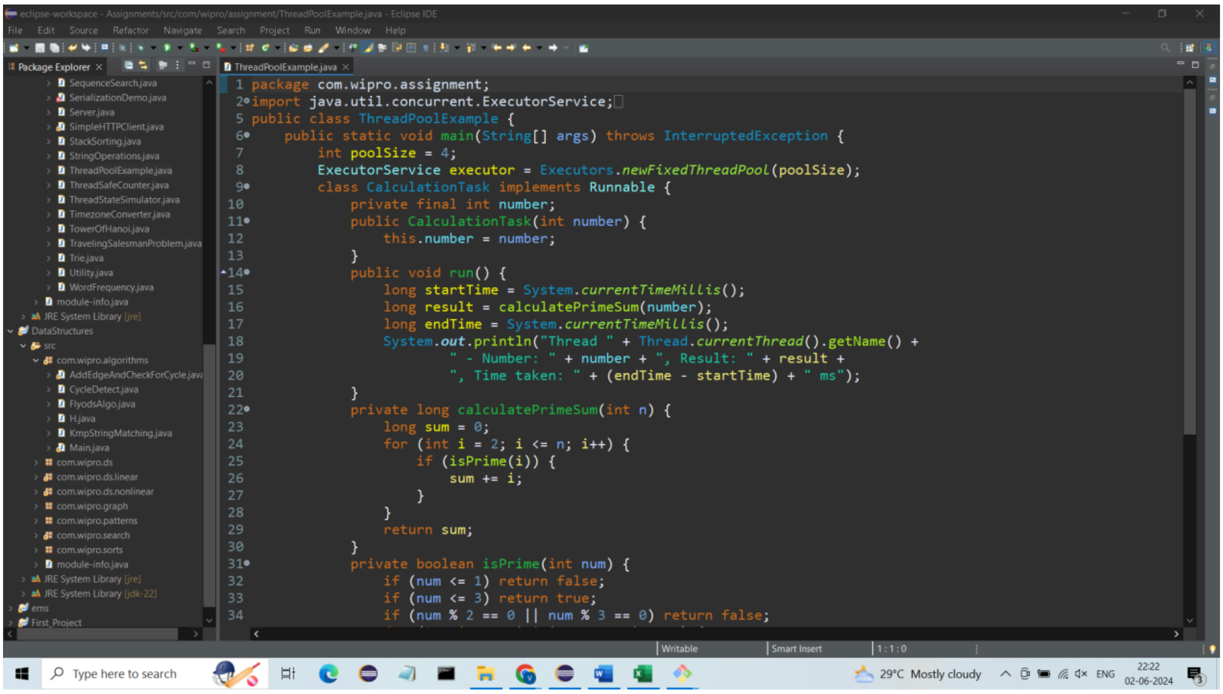


```java
        }
    public void run() {
        if (isDeposit) {
            account.deposit(amount);
        } else {
            try {
                account.withdraw(amount);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
public class Main1{
    public static void main(String[] args) throws InterruptedException {
        BankAccount account = new BankAccount();
        Thread thread1 = new Thread(new Transaction(account, true, 100));
        Thread thread2 = new Thread(new Transaction(account, false, 50));
        Thread thread3 = new Thread(new Transaction(account, true, 200));
        Thread thread4 = new Thread(new Transaction(account, false, 175))
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        thread1.join();
        thread2.join();
        thread3.join();
        thread4.join();
        System.out.println("Final Balance: " + account.balance);
    }
}
```

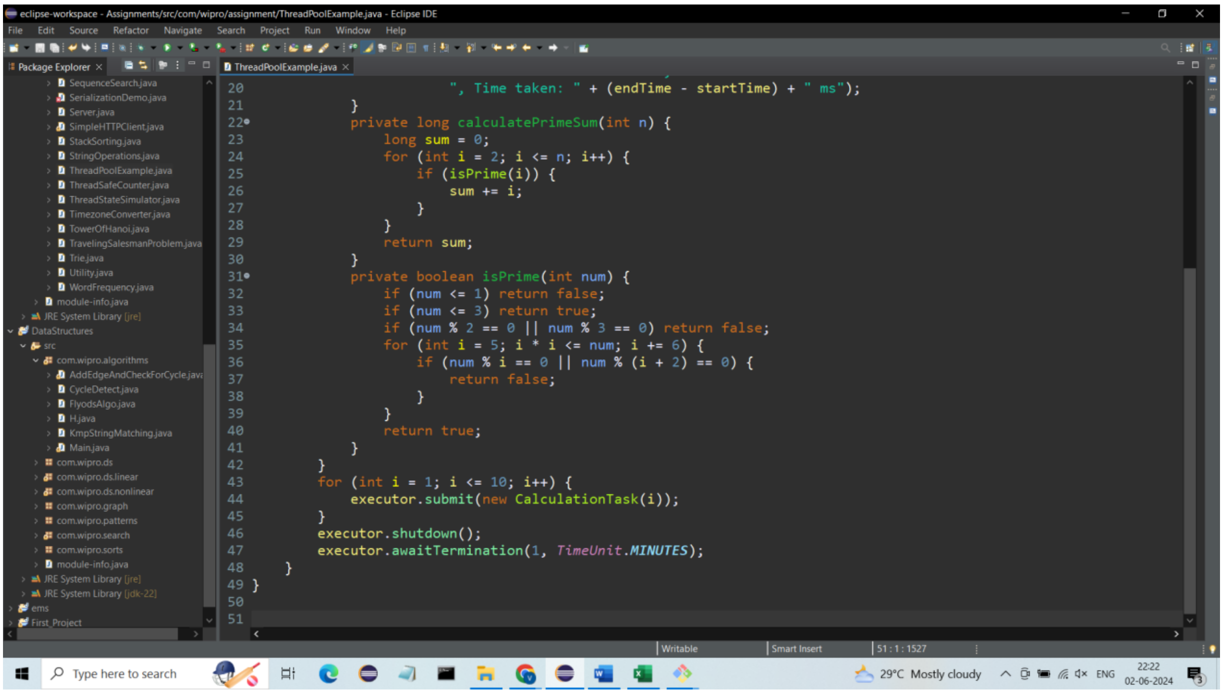
```
Deposited: 100, New Balance: 100
Insufficient funds, waiting for deposit.
Deposited: 200, New Balance: 300
Withdrew: 50, New Balance: 250
```

## Task 5: Thread Pools and Concurrency Utilities

Create a fixed-size thread pool and submit multiple tasks that perform complex calculations or I/O operations and observe the execution.

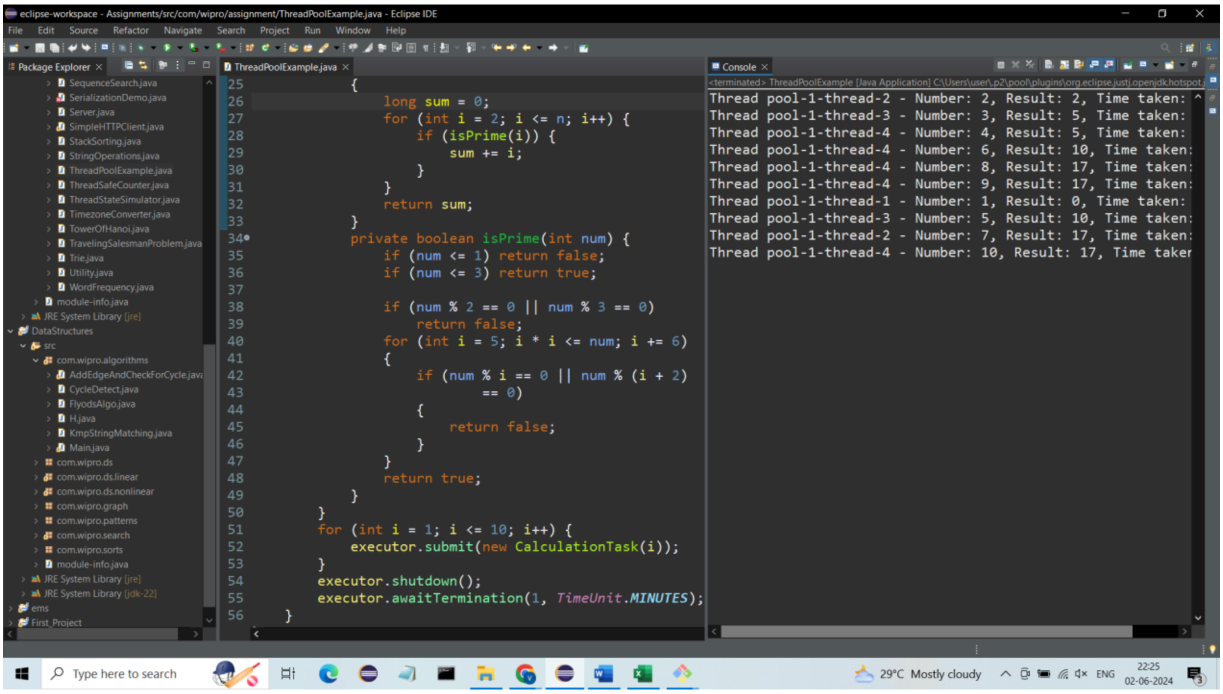


```java
package com.wipro.assignment;
import java.util.concurrent.ExecutorService;
public class ThreadPoolExample {
    public static void main(String[] args) throws InterruptedException {
        int poolSize = 4;
        ExecutorService executor = Executors.newFixedThreadPool(poolSize);
        class CalculationTask implements Runnable {
            private final int number;
            public CalculationTask(int number) {
                this.number = number;
            }
            public void run() {
                long startTime = System.currentTimeMillis();
                long result = calculatePrimeSum(number);
                long endTime = System.currentTimeMillis();
                System.out.println("Thread " + Thread.currentThread().getName() +
                        " - Number: " + number + ", Result: " + result +
                        ", Time taken: " + (endTime - startTime) + " ms");
            }
            private long calculatePrimeSum(int n) {
                long sum = 0;
                for (int i = 2; i <= n; i++) {
                    if (isPrime(i)) {
                        sum += i;
                    }
                }
                return sum;
            }
            private boolean isPrime(int num) {
                if (num <= 1) return false;
                if (num <= 3) return true;
                if (num % 2 == 0 || num % 3 == 0) return false;
```


eclipse-workspace - Assignments/src/com/wipro/assignment/ThreadPoolExample.java - Eclipse IDE
Package Explorer
ThreadPoolExample.java
Writable
Smart Insert
51 : 1 : 1527

```java
                        ", Time taken: " + (endTime - startTime) + " ms");
            }
            private long calculatePrimeSum(int n) {
                long sum = 0;
                for (int i = 2; i <= n; i++) {
                    if (isPrime(i)) {
                        sum += i;
                    }
                }
                return sum;
            }
            private boolean isPrime(int num) {
                if (num <= 1) return false;
                if (num <= 3) return true;
                if (num % 2 == 0 || num % 3 == 0) return false;
                for (int i = 5; i * i <= num; i += 6) {
                    if (num % i == 0 || num % (i + 2) == 0) {
                        return false;
                    }
                }
                return true;
            }
        }
        for (int i = 1; i <= 10; i++) {
            executor.submit(new CalculationTask(i));
        }
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.MINUTES);
    }
}
```

# Output: -

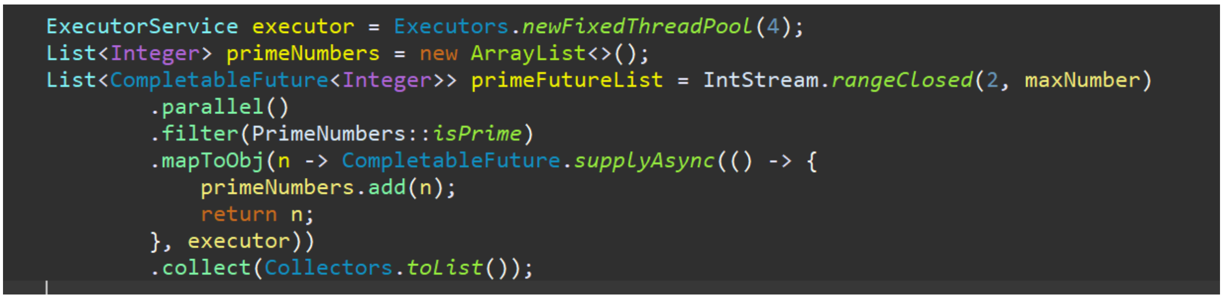

eclipse-workspace - Assignments/src/com/wipro/assignment/ThreadPoolExample.java - Eclipse IDE
Package Explorer
ThreadPoolExample.java
Console

```java
            {
                long sum = 0;
                for (int i = 2; i <= n; i++) {
                    if (isPrime(i)) {
                        sum += i;
                    }
                }
                return sum;
            }
            private boolean isPrime(int num) {
                if (num <= 1) return false;
                if (num <= 3) return true;

                if (num % 2 == 0 || num % 3 == 0)
                    return false;
                for (int i = 5; i * i <= num; i += 6)
                {
                    if (num % i == 0 || num % (i + 2)
                        == 0)
                    {
                        return false;
                    }
                }
                return true;
            }
        }
        for (int i = 1; i <= 10; i++) {
            executor.submit(new CalculationTask(i));
        }
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.MINUTES);
    }
```

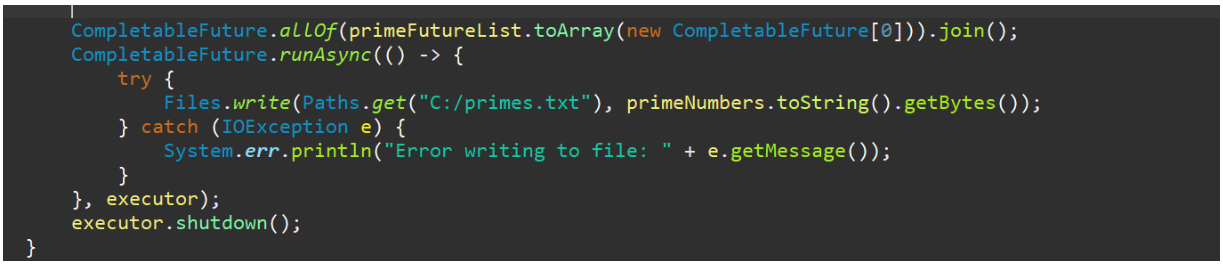
```
<terminated> ThreadPoolExample [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.j
Thread pool-1-thread-2 - Number: 2, Result: 2, Time taken:
Thread pool-1-thread-3 - Number: 3, Result: 5, Time taken:
Thread pool-1-thread-4 - Number: 4, Result: 5, Time taken:
Thread pool-1-thread-4 - Number: 6, Result: 10, Time taken:
Thread pool-1-thread-4 - Number: 8, Result: 17, Time taken:
Thread pool-1-thread-4 - Number: 9, Result: 17, Time taken:
Thread pool-1-thread-1 - Number: 1, Result: 0, Time taken:
Thread pool-1-thread-3 - Number: 5, Result: 10, Time taken:
Thread pool-1-thread-2 - Number: 7, Result: 17, Time taken:
Thread pool-1-thread-4 - Number: 10, Result: 17, Time taker
```

# Task 6: Executors, Concurrent Collections, CompletableFuture

Use an ExecutorService to parallelize a task that calculates prime numbers up to a given number and then use CompletableFuture to write the results to a file asynchronously.

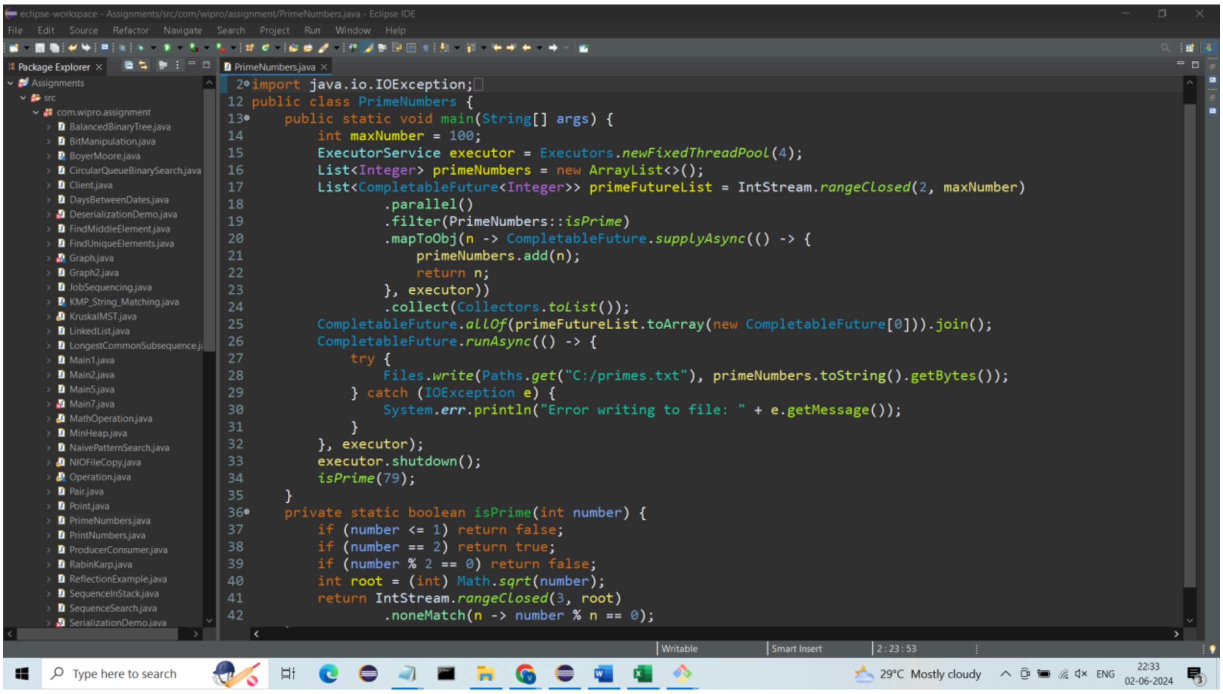


```java
ExecutorService executor = Executors.newFixedThreadPool(4);
List<Integer> primeNumbers = new ArrayList<>();
List<CompletableFuture<Integer>> primeFutureList = IntStream.rangeClosed(2, maxNumber)
        .parallel()
        .filter(PrimeNumbers::isPrime)
        .mapToObj(n -> CompletableFuture.supplyAsync(() -> {
            primeNumbers.add(n);
            return n;
        }, executor))
        .collect(Collectors.toList());
```



```java
    CompletableFuture.allOf(primeFutureList.toArray(new CompletableFuture[0])).join();
    CompletableFuture.runAsync(() -> {
        try {
            Files.write(Paths.get("C:/primes.txt"), primeNumbers.toString().getBytes());
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }
    }, executor);
    executor.shutdown();
}
```

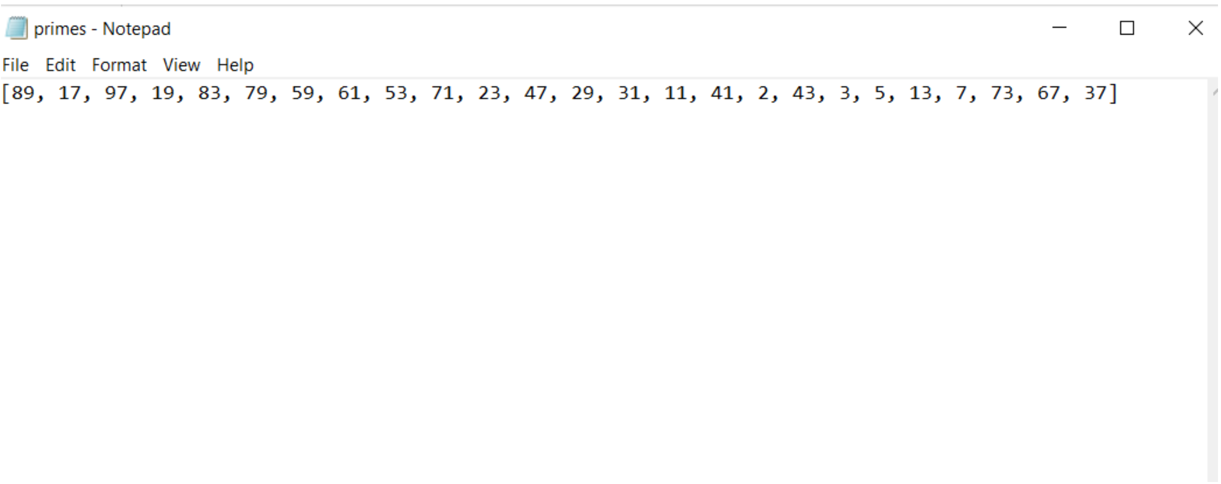
## Program: -



eclipse-workspace - Assignments/src/com/wipro/assignment/PrimeNumbers.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
PrimeNumbers.java
Writable
Smart Insert
2 : 23 : 53
29°C Mostly cloudy
22:33
02-06-2024

```java
import java.io.IOException;
public class PrimeNumbers {
    public static void main(String[] args) {
        int maxNumber = 100;
        ExecutorService executor = Executors.newFixedThreadPool(4);
        List<Integer> primeNumbers = new ArrayList<>();
        List<CompletableFuture<Integer>> primeFutureList = IntStream.rangeClosed(2, maxNumber)
                .parallel()
                .filter(PrimeNumbers::isPrime)
                .mapToObj(n -> CompletableFuture.supplyAsync(() -> {
                    primeNumbers.add(n);
                    return n;
                }, executor))
                .collect(Collectors.toList());
        CompletableFuture.allOf(primeFutureList.toArray(new CompletableFuture[0])).join();
        CompletableFuture.runAsync(() -> {
            try {
                Files.write(Paths.get("C:/primes.txt"), primeNumbers.toString().getBytes());
            } catch (IOException e) {
                System.err.println("Error writing to file: " + e.getMessage());
            }
        }, executor);
        executor.shutdown();
        isPrime(79);
    }
    private static boolean isPrime(int number) {
        if (number <= 1) return false;
        if (number == 2) return true;
        if (number % 2 == 0) return false;
        int root = (int) Math.sqrt(number);
        return IntStream.rangeClosed(3, root)
                .noneMatch(n -> number % n == 0);
```

## Output: -

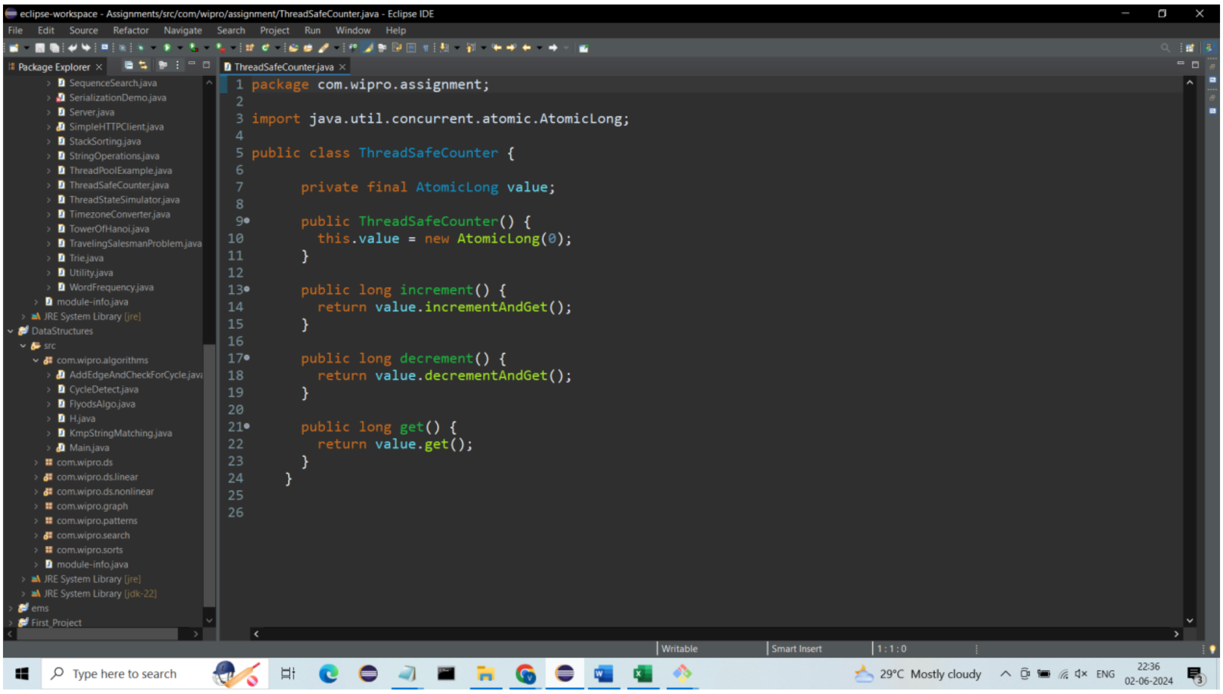


```
[89, 17, 97, 19, 83, 79, 59, 61, 53, 71, 23, 47, 29, 31, 11, 41, 2, 43, 3, 5, 13, 7, 73, 67, 37]
```

## Task 7: Writing Thread-Safe Code, Immutable Objects

Design a thread-safe Counter class with increment and decrement methods. Then demonstrate its usage from multiple threads. Also, implement and use an immutable class to share data between threads.

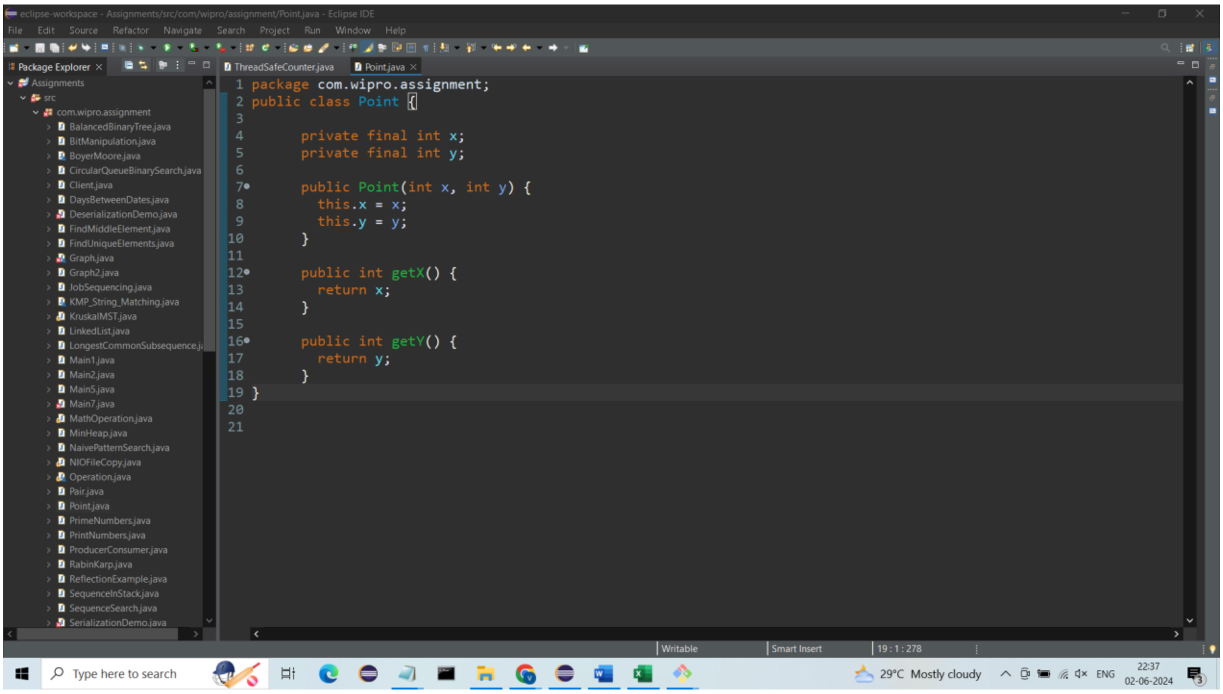
### 1. ThreadSafeCounter Class:



eclipse-workspace - Assignments/src/com/wipro/assignment/ThreadSafeCounter.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
ThreadSafeCounter.java
Writable
Smart Insert
1 : 1 : 0
Type here to search
29°C Mostly cloudy
ENG
22:36
02-06-2024

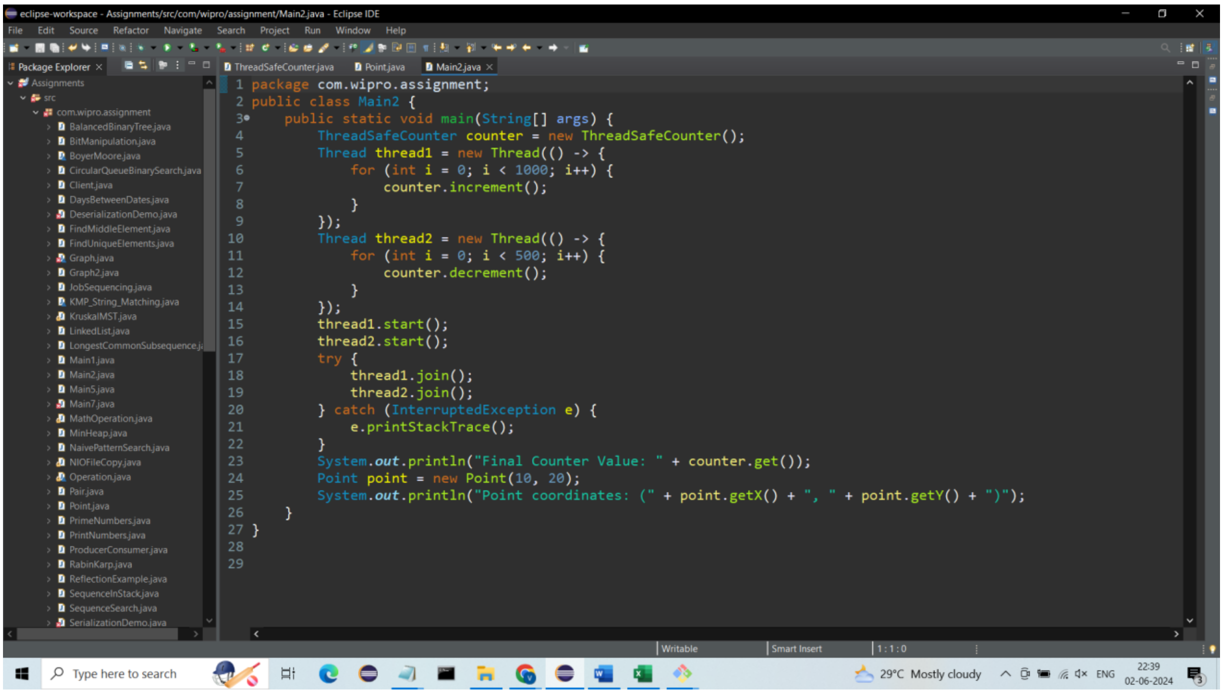
```java
package com.wipro.assignment;

import java.util.concurrent.atomic.AtomicLong;

public class ThreadSafeCounter {

	private final AtomicLong value;

	public ThreadSafeCounter() {
	  this.value = new AtomicLong(0);
	}

	public long increment() {
	  return value.incrementAndGet();
	}

	public long decrement() {
	  return value.decrementAndGet();
	}

	public long get() {
	  return value.get();
	}
    }
```

## 2. Point Class (Immutable):



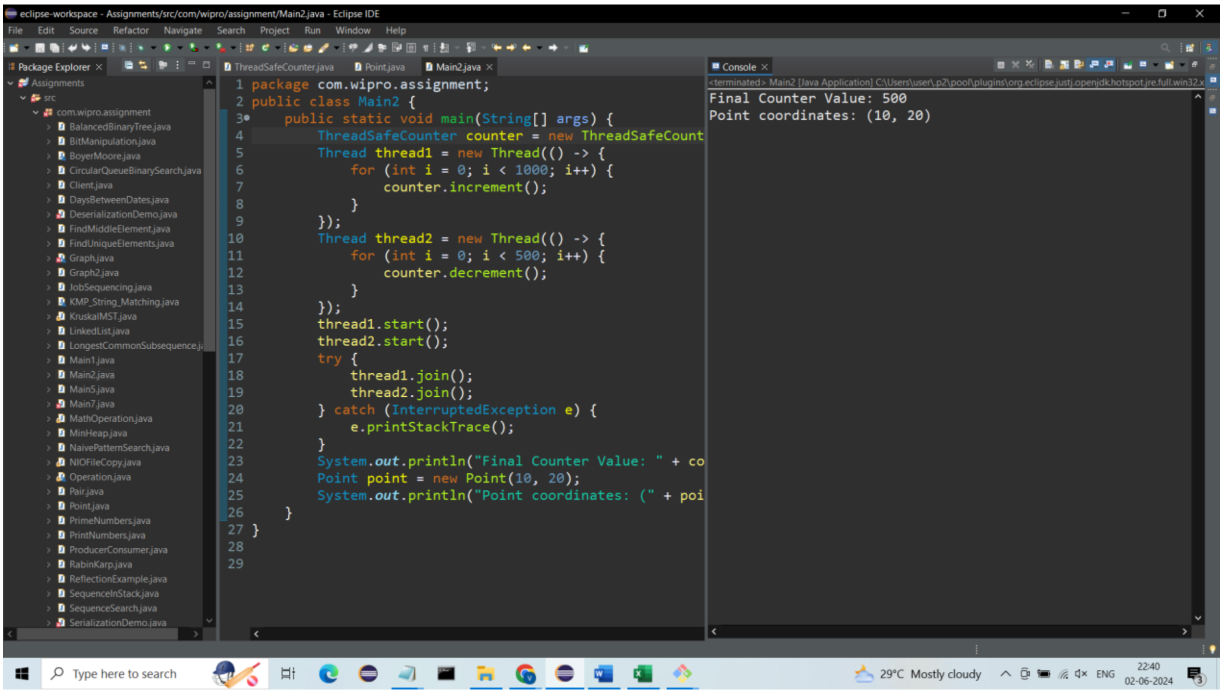
```java
package com.wipro.assignment;
public class Point {

      private final int x;
      private final int y;

      public Point(int x, int y) {
        this.x = x;
        this.y = y;
      }

      public int getX() {
        return x;
      }

      public int getY() {
        return y;
      }
}
```

## 3. Usage Example



```java
package com.wipro.assignment;
public class Main2 {
    public static void main(String[] args) {
        ThreadSafeCounter counter = new ThreadSafeCounter();
        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });
        Thread thread2 = new Thread(() -> {
            for (int i = 0; i < 500; i++) {
                counter.decrement();
            }
        });
        thread1.start();
        thread2.start();
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Final Counter Value: " + counter.get());
        Point point = new Point(10, 20);
        System.out.println("Point coordinates: (" + point.getX() + ", " + point.getY() + ")");
    }
}
```

## Output: -



```java
package com.wipro.assignment;
public class Main2 {
    public static void main(String[] args) {
        ThreadSafeCounter counter = new ThreadSafeCount
        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });
        Thread thread2 = new Thread(() -> {
            for (int i = 0; i < 500; i++) {
                counter.decrement();
            }
        });
        thread1.start();
        thread2.start();
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Final Counter Value: " + co
        Point point = new Point(10, 20);
        System.out.println("Point coordinates: (" + poi
    }
}
```

```
Final Counter Value: 500
Point coordinates: (10, 20)
```