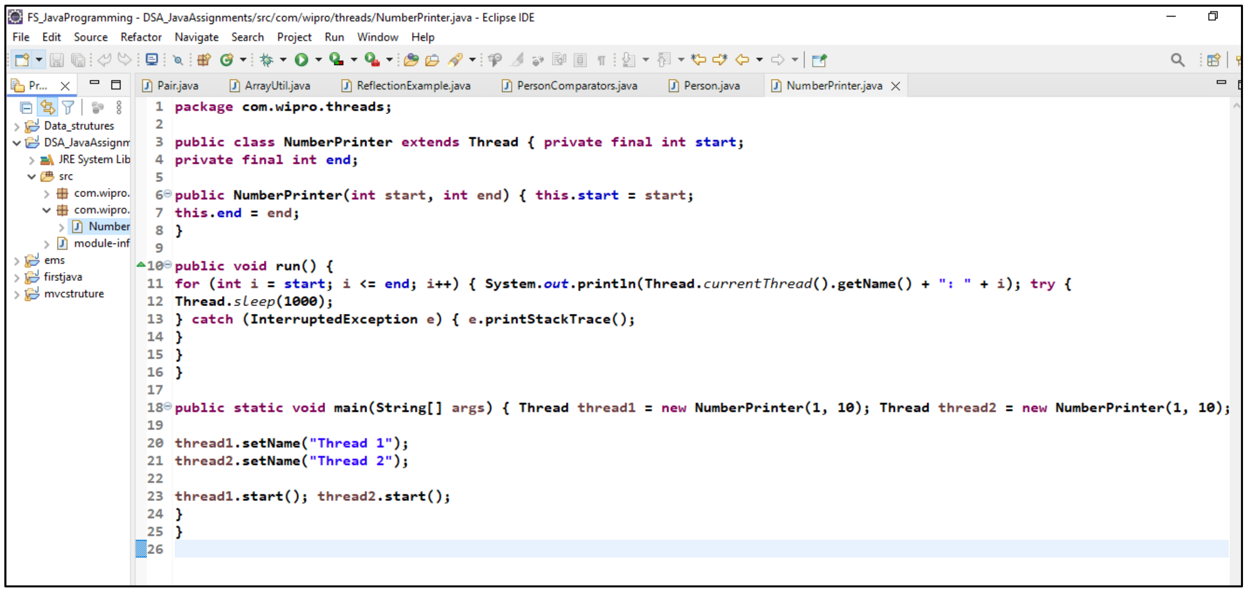
Name: Tejaswini Anil Kamble
Email : teju000kamble@gmail.com

Threads Assignments: Day 18

Task 1: Creating and Managing Threads

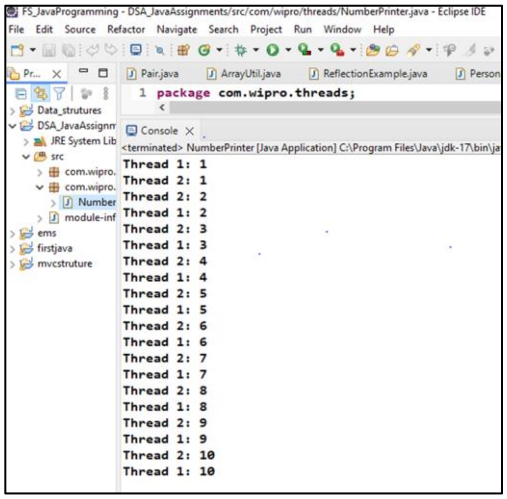
Write a program that starts two threads, where each thread prints numbers from 1 to 10 with a 1-second delay between each number

Ans : Source Code



```java
package com.wipro.threads;

public class NumberPrinter extends Thread { private final int start;
private final int end;

public NumberPrinter(int start, int end) { this.start = start;
this.end = end;
}

public void run() {
for (int i = start; i <= end; i++) { System.out.println(Thread.currentThread().getName() + ": " + i); try {
Thread.sleep(1000);
} catch (InterruptedException e) { e.printStackTrace();
}
}
}

public static void main(String[] args) { Thread thread1 = new NumberPrinter(1, 10); Thread thread2 = new NumberPrinter(1, 10);

thread1.setName("Thread 1");
thread2.setName("Thread 2");

thread1.start(); thread2.start();
}
}
```

Output:

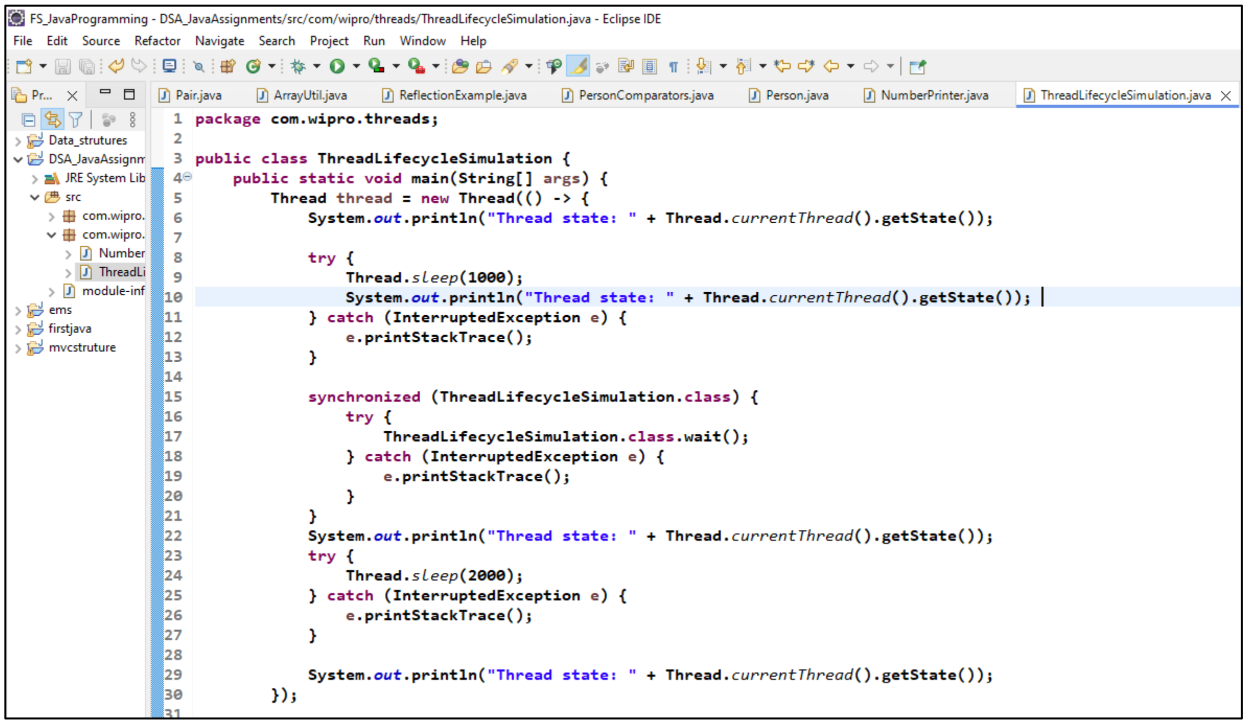


```
<terminated> NumberPrinter [Java Application] C:\Program Files\Java\jdk-17\bin\ja
Thread 1: 1
Thread 2: 1
Thread 2: 2
Thread 1: 2
Thread 2: 3
Thread 1: 3
Thread 2: 4
Thread 1: 4
Thread 2: 5
Thread 1: 5
Thread 2: 6
Thread 1: 6
Thread 2: 7
Thread 1: 7
Thread 2: 8
Thread 1: 8
Thread 2: 9
Thread 1: 9
Thread 2: 10
Thread 1: 10
```

Task 2: States and Transitions

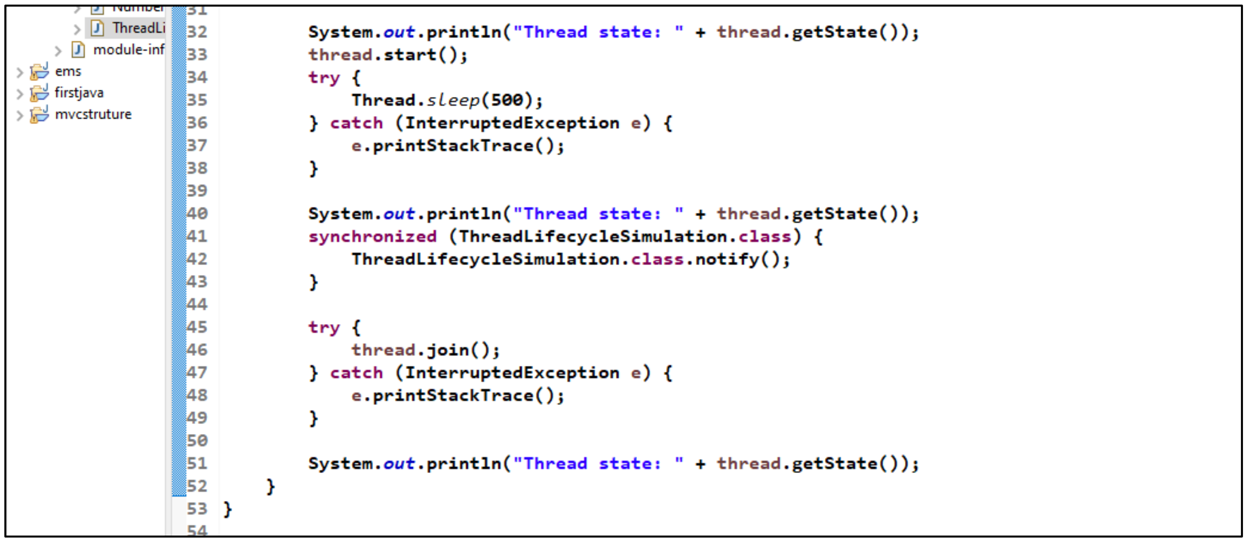
Create a Java class that simulates a thread going through different lifecycle states: NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCKED, and TERMINATED. Use methods like sleep(), wait(), notify(), and join() to demonstrate these states

Ans : Source code



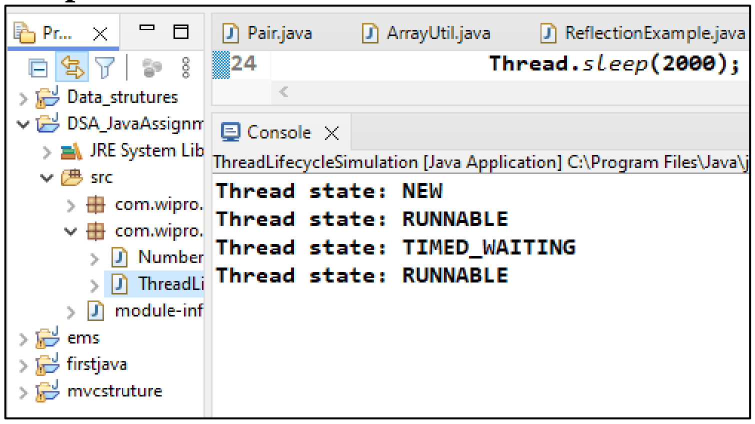
FS_JavaProgramming - DSA_JavaAssignments/src/com/wipro/threads/ThreadLifecycleSimulation.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Pair.java ArrayUtil.java ReflectionExample.java PersonComparators.java Person.java NumberPrinter.java ThreadLifecycleSimulation.java
Data_strutures
DSA_JavaAssignm
JRE System Lib
src
com.wipro.
com.wipro.
Number
ThreadLi
module-inf
ems
firstjava
mvcstruture

```java
package com.wipro.threads;

public class ThreadLifecycleSimulation {
    public static void main(String[] args) {
        Thread thread = new Thread(() -> {
            System.out.println("Thread state: " + Thread.currentThread().getState());

            try {
                Thread.sleep(1000);
                System.out.println("Thread state: " + Thread.currentThread().getState());
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            synchronized (ThreadLifecycleSimulation.class) {
                try {
                    ThreadLifecycleSimulation.class.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("Thread state: " + Thread.currentThread().getState());
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            System.out.println("Thread state: " + Thread.currentThread().getState());
        });
```



```java
            System.out.println("Thread state: " + thread.getState());
            thread.start();
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            System.out.println("Thread state: " + thread.getState());
            synchronized (ThreadLifecycleSimulation.class) {
                ThreadLifecycleSimulation.class.notify();
            }

            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            System.out.println("Thread state: " + thread.getState());
        }
}
```

Output:



```
24                Thread.sleep(2000);
```

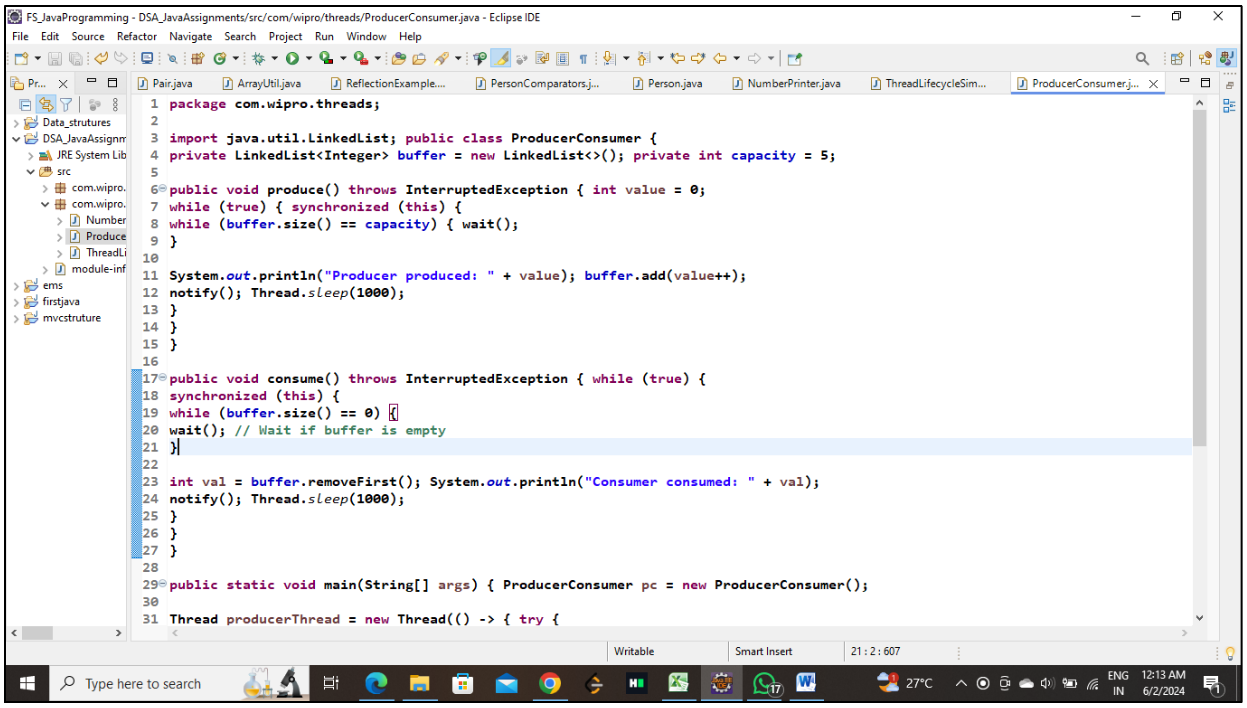
ThreadLifecycleSimulation [Java Application] C:\Program Files\Java\j

```
Thread state: NEW
Thread state: RUNNABLE
Thread state: TIMED_WAITING
Thread state: RUNNABLE
```

Task 3: Synchronization and Inter-thread Communication

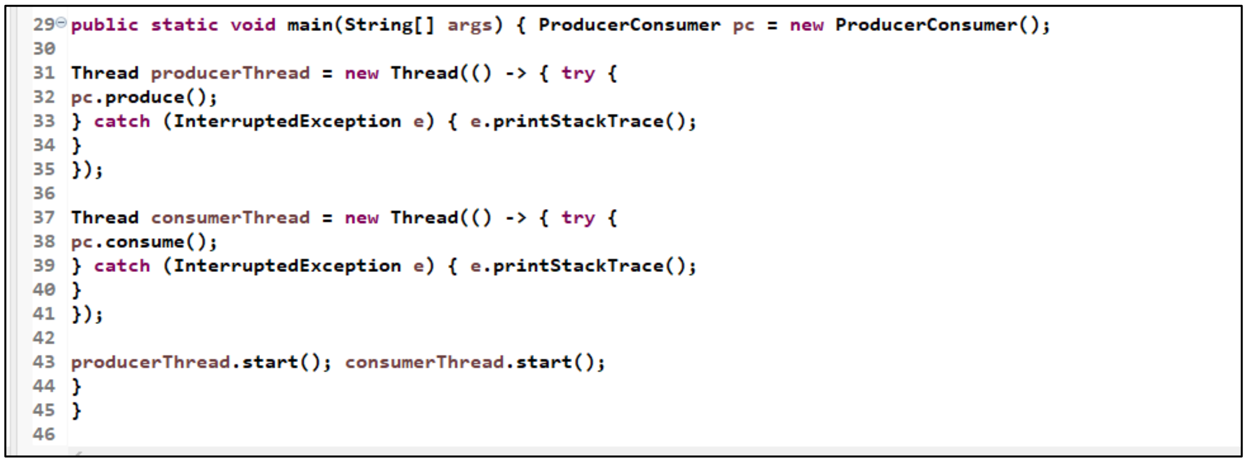
Implement a producer-consumer problem using wait() and notify() methods to handle the correct processing sequence between threads.

Ans: Sourcecode



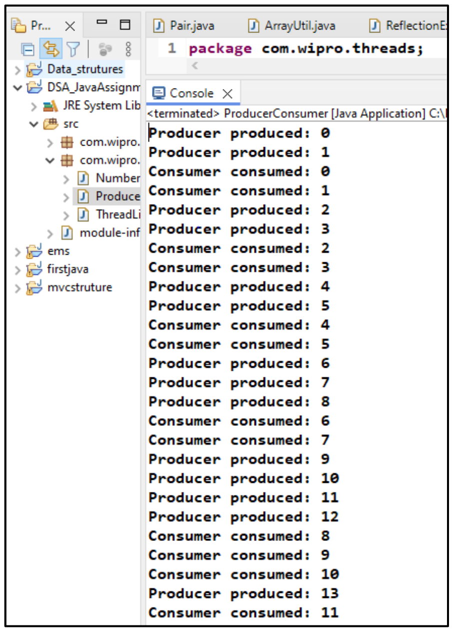
FS_JavaProgramming - DSA_JavaAssignments/src/com/wipro/threads/ProducerConsumer.java - Eclipse IDE

```java
package com.wipro.threads;

import java.util.LinkedList; public class ProducerConsumer {
private LinkedList<Integer> buffer = new LinkedList<>(); private int capacity = 5;

public void produce() throws InterruptedException { int value = 0;
while (true) { synchronized (this) {
while (buffer.size() == capacity) { wait();
}

System.out.println("Producer produced: " + value); buffer.add(value++);
notify(); Thread.sleep(1000);
}
}
}

public void consume() throws InterruptedException { while (true) {
synchronized (this) {
while (buffer.size() == 0) {
wait(); // Wait if buffer is empty
}

int val = buffer.removeFirst(); System.out.println("Consumer consumed: " + val);
notify(); Thread.sleep(1000);
}
}
}

public static void main(String[] args) { ProducerConsumer pc = new ProducerConsumer();

Thread producerThread = new Thread(() -> { try {
```



```java
public static void main(String[] args) { ProducerConsumer pc = new ProducerConsumer();

Thread producerThread = new Thread(() -> { try {
pc.produce();
} catch (InterruptedException e) { e.printStackTrace();
}
});

Thread consumerThread = new Thread(() -> { try {
pc.consume();
} catch (InterruptedException e) { e.printStackTrace();
}
});

producerThread.start(); consumerThread.start();
}
}
```

Output:



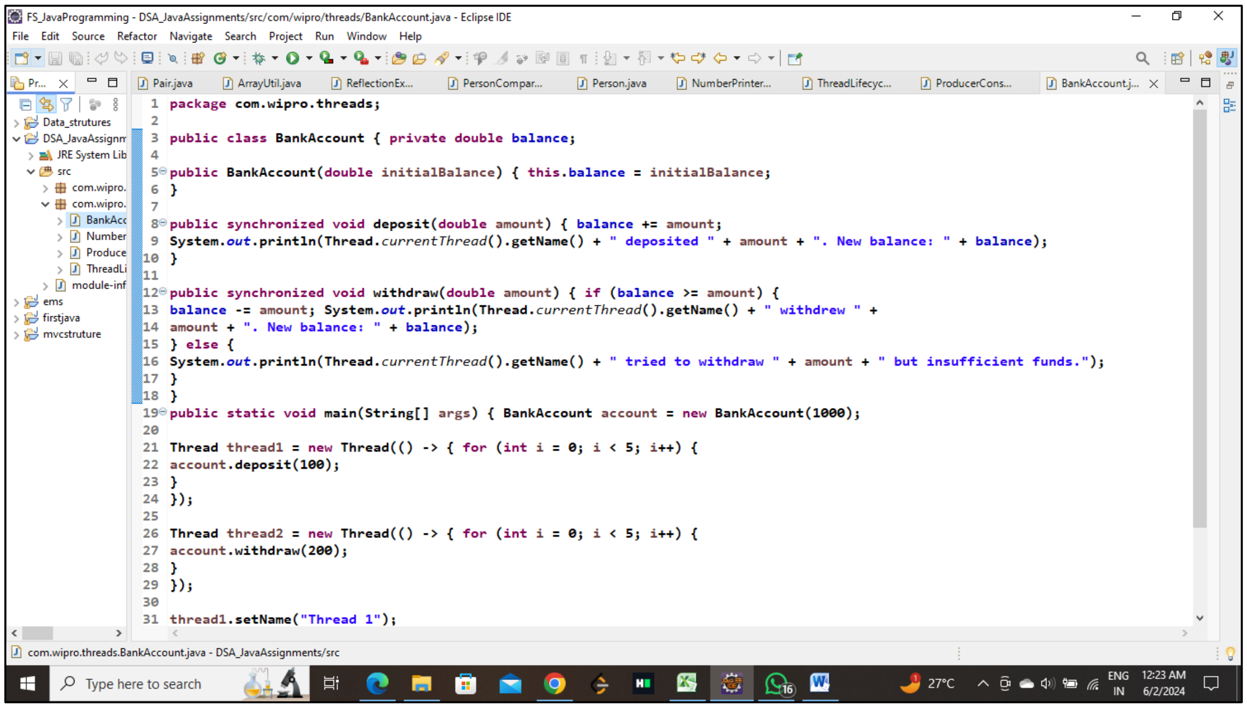
```
1 package com.wipro.threads;
```

```
<terminated> ProducerConsumer [Java Application] C:\
Producer produced: 0
Producer produced: 1
Consumer consumed: 0
Consumer consumed: 1
Producer produced: 2
Producer produced: 3
Consumer consumed: 2
Consumer consumed: 3
Producer produced: 4
Producer produced: 5
Consumer consumed: 4
Consumer consumed: 5
Producer produced: 6
Producer produced: 7
Producer produced: 8
Consumer consumed: 6
Consumer consumed: 7
Producer produced: 9
Producer produced: 10
Producer produced: 11
Producer produced: 12
Consumer consumed: 8
Consumer consumed: 9
Consumer consumed: 10
Producer produced: 13
Consumer consumed: 11
```

Task 4: Synchronized Blocks and Methods

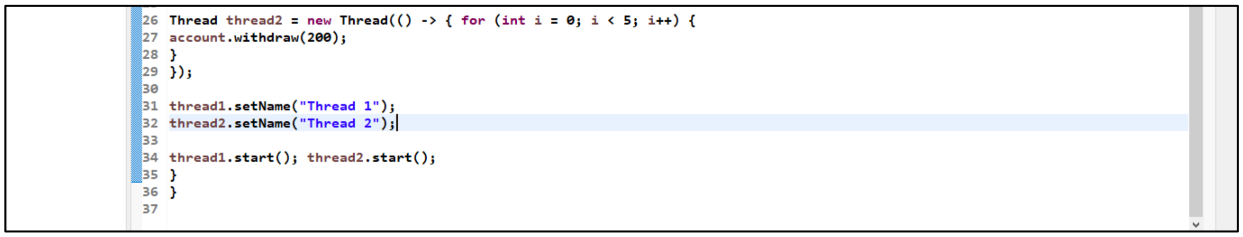
Write a program that simulates a bank account being accessed by multiple threads to perform deposits and withdrawals using synchronized methods to prevent race conditions.

Ans: Source Code


FS_JavaProgramming - DSA_JavaAssignments/src/com/wipro/threads/BankAccount.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

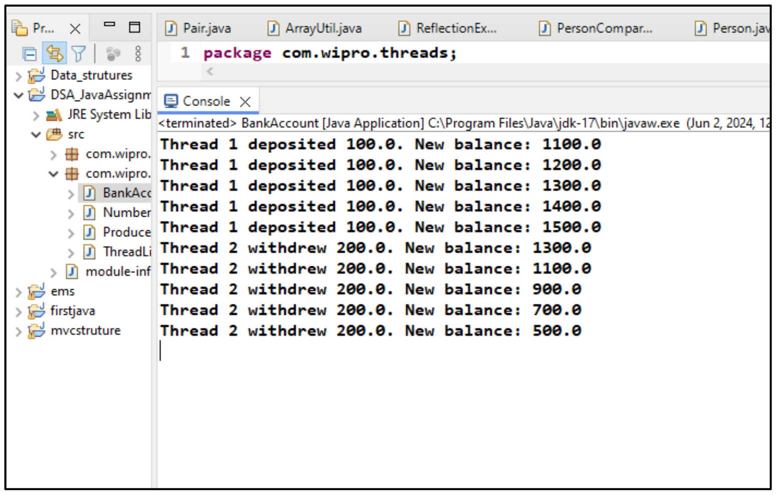
```java
package com.wipro.threads;

public class BankAccount { private double balance;

public BankAccount(double initialBalance) { this.balance = initialBalance;
}

public synchronized void deposit(double amount) { balance += amount;
System.out.println(Thread.currentThread().getName() + " deposited " + amount + ". New balance: " + balance);
}

public synchronized void withdraw(double amount) { if (balance >= amount) {
balance -= amount; System.out.println(Thread.currentThread().getName() + " withdrew " +
amount + ". New balance: " + balance);
} else {
System.out.println(Thread.currentThread().getName() + " tried to withdraw " + amount + " but insufficient funds.");
}
}
public static void main(String[] args) { BankAccount account = new BankAccount(1000);

Thread thread1 = new Thread(() -> { for (int i = 0; i < 5; i++) {
account.deposit(100);
}
});

Thread thread2 = new Thread(() -> { for (int i = 0; i < 5; i++) {
account.withdraw(200);
}
});

thread1.setName("Thread 1");
```

com.wipro.threads.BankAccount.java - DSA_JavaAssignments/src



```java
Thread thread2 = new Thread(() -> { for (int i = 0; i < 5; i++) {
account.withdraw(200);
}
});

thread1.setName("Thread 1");
thread2.setName("Thread 2");

thread1.start(); thread2.start();
}
}
```

Output:

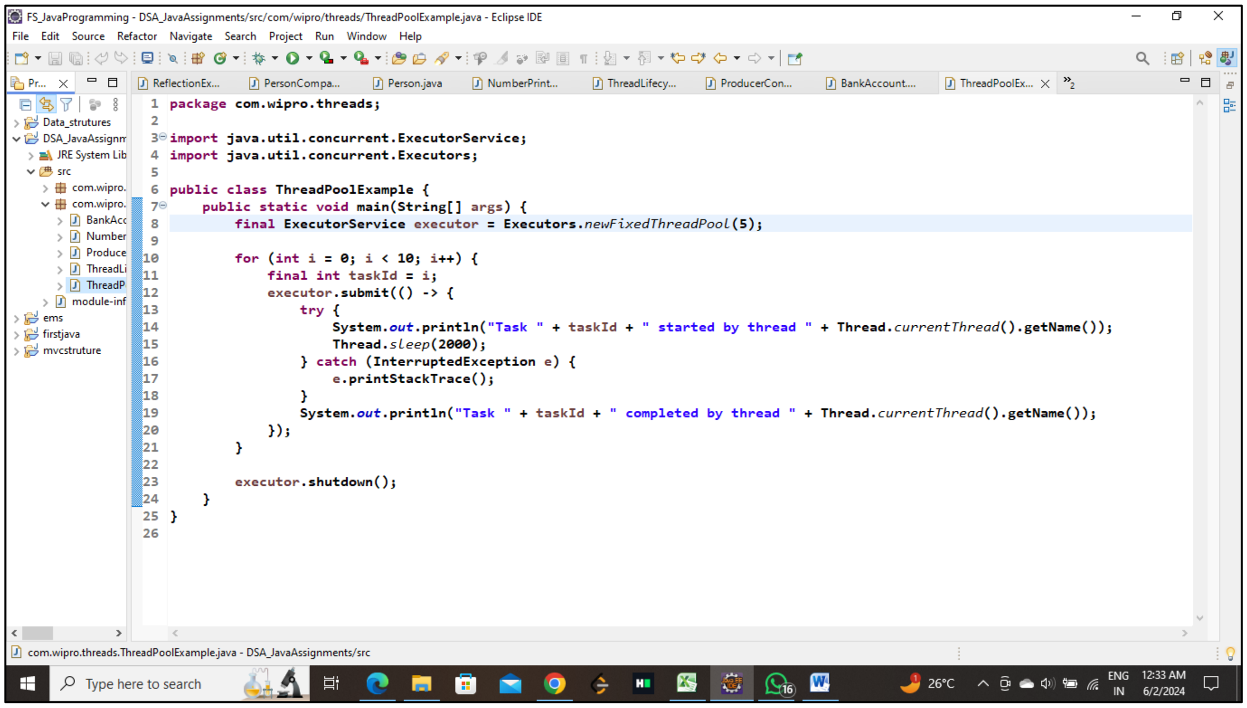


```
<terminated> BankAccount [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Jun 2, 2024, 12
Thread 1 deposited 100.0. New balance: 1100.0
Thread 1 deposited 100.0. New balance: 1200.0
Thread 1 deposited 100.0. New balance: 1300.0
Thread 1 deposited 100.0. New balance: 1400.0
Thread 1 deposited 100.0. New balance: 1500.0
Thread 2 withdrew 200.0. New balance: 1300.0
Thread 2 withdrew 200.0. New balance: 1100.0
Thread 2 withdrew 200.0. New balance: 900.0
Thread 2 withdrew 200.0. New balance: 700.0
Thread 2 withdrew 200.0. New balance: 500.0
```

Task 5: Thread Pools and Concurrency Utilities

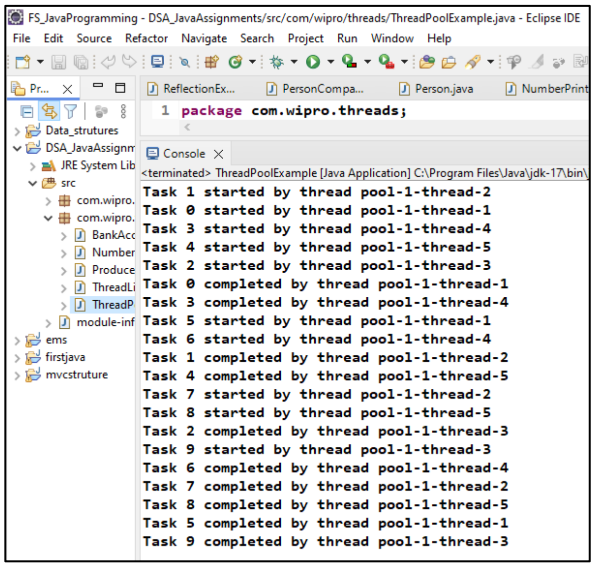
Create a fixed-size thread pool and submit multiple tasks that perform complex calculations or I/O operations and observe the execution

Ans: Source code



```java
package com.wipro.threads;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ThreadPoolExample {
    public static void main(String[] args) {
        final ExecutorService executor = Executors.newFixedThreadPool(5);

        for (int i = 0; i < 10; i++) {
            final int taskId = i;
            executor.submit(() -> {
                try {
                    System.out.println("Task " + taskId + " started by thread " + Thread.currentThread().getName());
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Task " + taskId + " completed by thread " + Thread.currentThread().getName());
            });
        }

        executor.shutdown();
    }
}
```

Output:

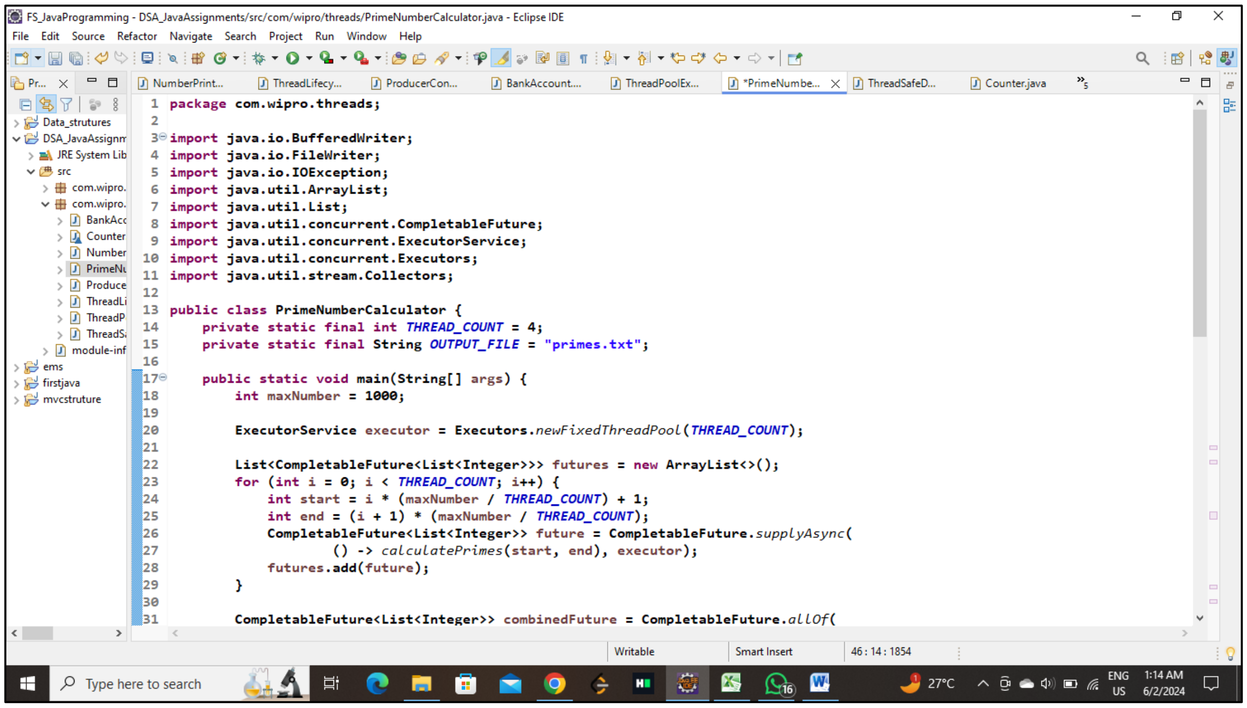


```
<terminated> ThreadPoolExample [Java Application] C:\Program Files\Java\jdk-17\bin\
Task 1 started by thread pool-1-thread-2
Task 0 started by thread pool-1-thread-1
Task 3 started by thread pool-1-thread-4
Task 4 started by thread pool-1-thread-5
Task 2 started by thread pool-1-thread-3
Task 0 completed by thread pool-1-thread-1
Task 3 completed by thread pool-1-thread-4
Task 5 started by thread pool-1-thread-1
Task 6 started by thread pool-1-thread-4
Task 1 completed by thread pool-1-thread-2
Task 4 completed by thread pool-1-thread-5
Task 7 started by thread pool-1-thread-2
Task 8 started by thread pool-1-thread-5
Task 2 completed by thread pool-1-thread-3
Task 9 started by thread pool-1-thread-3
Task 6 completed by thread pool-1-thread-4
Task 7 completed by thread pool-1-thread-2
Task 8 completed by thread pool-1-thread-5
Task 5 completed by thread pool-1-thread-1
Task 9 completed by thread pool-1-thread-3
```

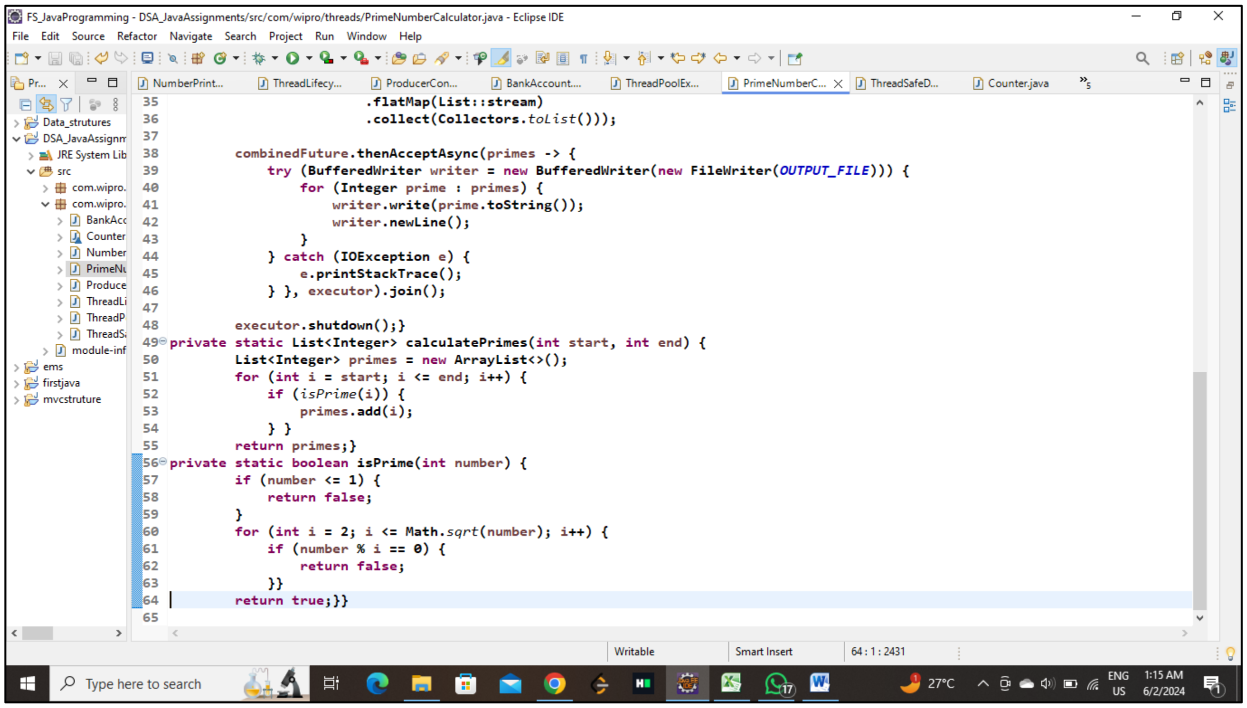
Task 6: Executors, Concurrent Collections, CompletableFuture

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.

Ans : Source Code


FS_JavaProgramming - DSA_JavaAssignments/src/com/wipro/threads/PrimeNumberCalculator.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
NumberPrint... ThreadLifecy... ProducerCon... BankAccount.... ThreadPoolEx... *PrimeNumbe... ThreadSafeD... Counter.java
Writable
Smart Insert
46 : 14 : 1854

```java
package com.wipro.threads;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.stream.Collectors;

public class PrimeNumberCalculator {
    private static final int THREAD_COUNT = 4;
    private static final String OUTPUT_FILE = "primes.txt";

    public static void main(String[] args) {
        int maxNumber = 1000;

        ExecutorService executor = Executors.newFixedThreadPool(THREAD_COUNT);

        List<CompletableFuture<List<Integer>>> futures = new ArrayList<>();
        for (int i = 0; i < THREAD_COUNT; i++) {
            int start = i * (maxNumber / THREAD_COUNT) + 1;
            int end = (i + 1) * (maxNumber / THREAD_COUNT);
            CompletableFuture<List<Integer>> future = CompletableFuture.supplyAsync(
                    () -> calculatePrimes(start, end), executor);
            futures.add(future);
        }

        CompletableFuture<List<Integer>> combinedFuture = CompletableFuture.allOf(
```

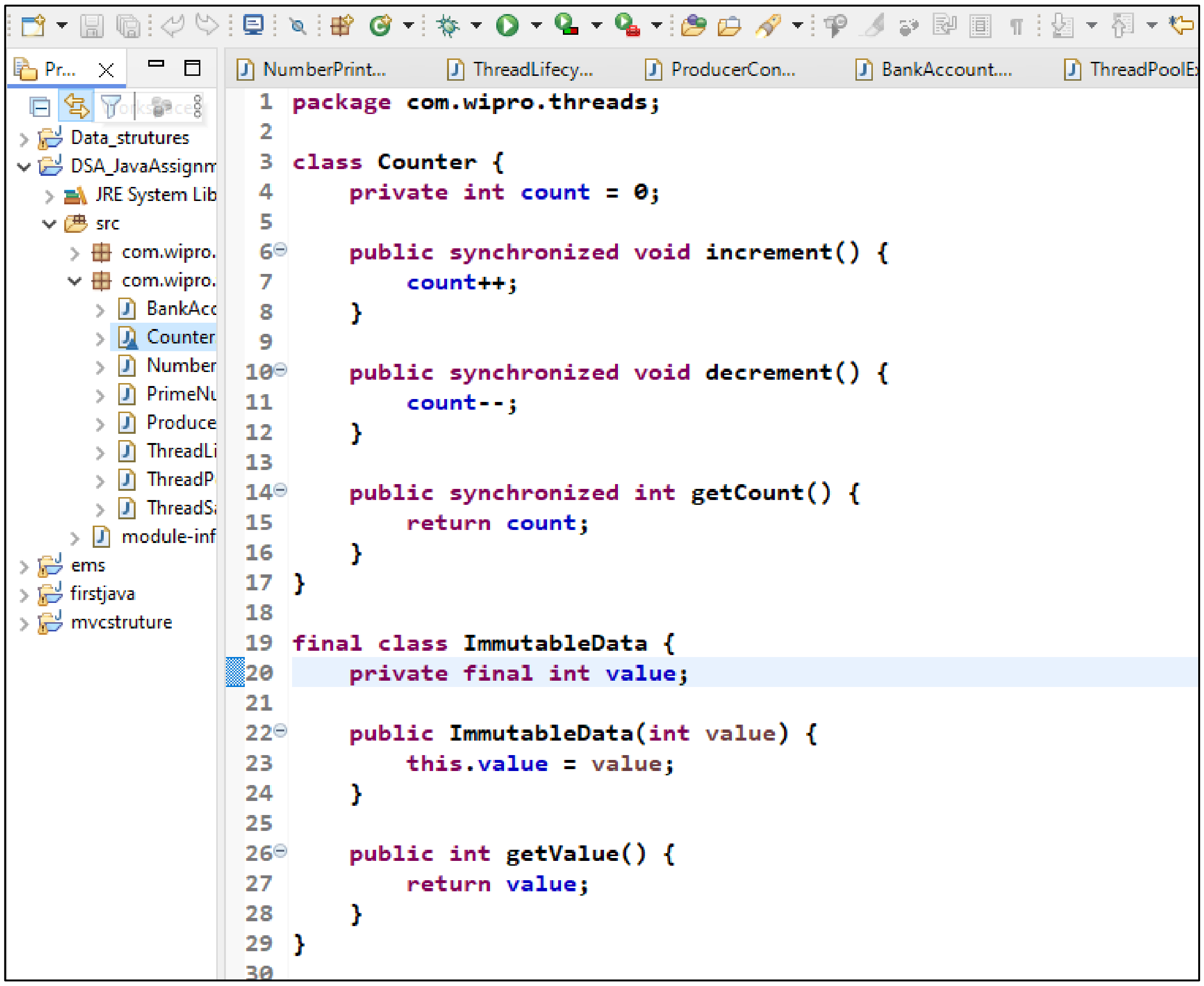

FS_JavaProgramming - DSA_JavaAssignments/src/com/wipro/threads/PrimeNumberCalculator.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
NumberPrint... ThreadLifecy... ProducerCon... BankAccount.... ThreadPoolEx... PrimeNumberC... ThreadSafeD... Counter.java
Data_strutures
DSA_JavaAssignm
JRE System Lib
src
ems
firstjava
mvcstruture
Writable
Smart Insert
64 : 1 : 2431

```java
                    .flatMap(List::stream)
                    .collect(Collectors.toList()));

        combinedFuture.thenAcceptAsync(primes -> {
            try (BufferedWriter writer = new BufferedWriter(new FileWriter(OUTPUT_FILE))) {
                for (Integer prime : primes) {
                    writer.write(prime.toString());
                    writer.newLine();
                }
            } catch (IOException e) {
                e.printStackTrace();
            } }, executor).join();

        executor.shutdown();}
private static List<Integer> calculatePrimes(int start, int end) {
        List<Integer> primes = new ArrayList<>();
        for (int i = start; i <= end; i++) {
            if (isPrime(i)) {
                primes.add(i);
            } }
        return primes;}
private static boolean isPrime(int number) {
        if (number <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                return false;
            }}
        return true;}}
```

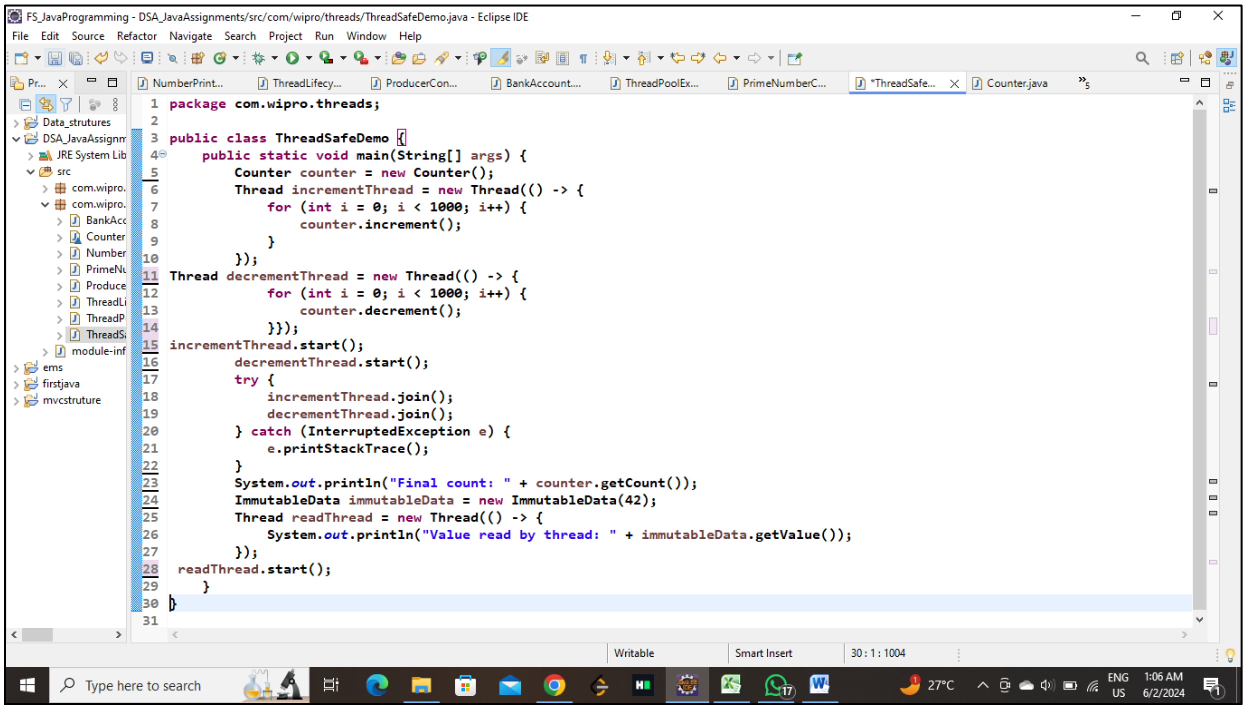
Task 7: Writing Thread-Safe Code, Immutable Objects

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.

Ans: Source code

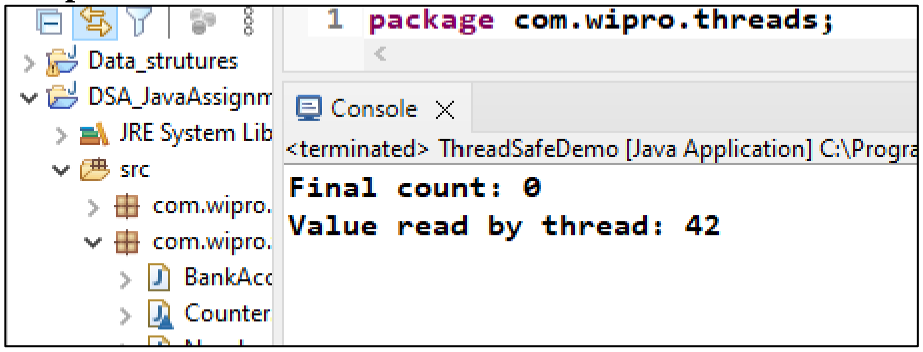


```java
package com.wipro.threads;

class Counter {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

    public synchronized void decrement() {
        count--;
    }

    public synchronized int getCount() {
        return count;
    }
}

final class ImmutableData {
    private final int value;

    public ImmutableData(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```


FS_JavaProgramming - DSA_JavaAssignments/src/com/wipro/threads/ThreadSafeDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*ThreadSafe...
Counter.java
Writable
Smart Insert
30 : 1 : 1004

```java
package com.wipro.threads;

public class ThreadSafeDemo {
    public static void main(String[] args) {
        Counter counter = new Counter();
        Thread incrementThread = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });
Thread decrementThread = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.decrement();
            }});
incrementThread.start();
        decrementThread.start();
        try {
            incrementThread.join();
            decrementThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Final count: " + counter.getCount());
        ImmutableData immutableData = new ImmutableData(42);
        Thread readThread = new Thread(() -> {
            System.out.println("Value read by thread: " + immutableData.getValue());
        });
 readThread.start();
    }
}
```

Output :



```
package com.wipro.threads;
```

```
<terminated> ThreadSafeDemo [Java Application] C:\Progra
Final count: 0
Value read by thread: 42
```