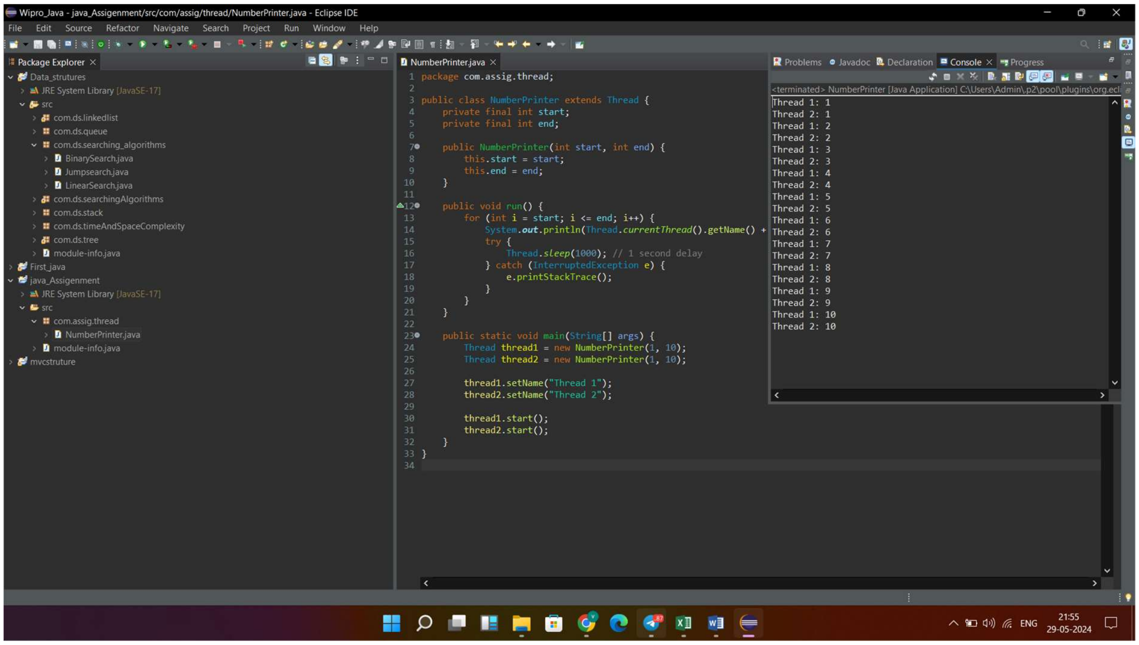
Day:18

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

```java
package com.assig.thread;

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();
    }
}
```

OP:-



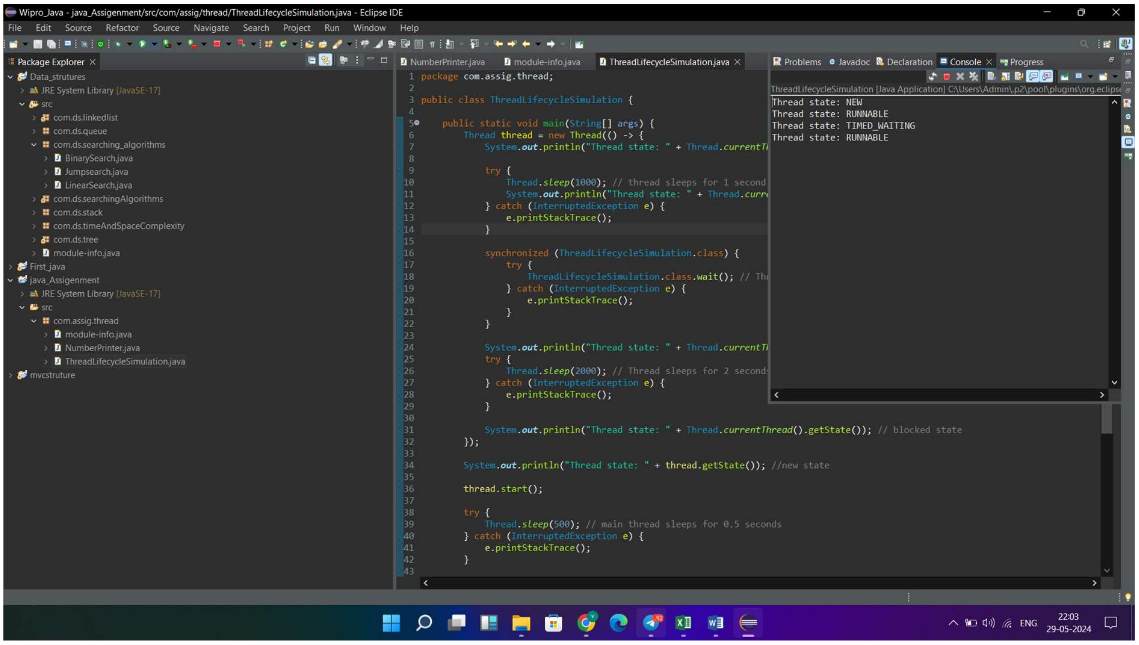
```java
package com.assig.thread;

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() +
            try {
                Thread.sleep(1000); // 1 second delay
            } 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();
    }
}
```

```
<terminated> NumberPrinter [Java Application] C:\Users\Admin\.p2\pool\plugins\org.ecl
Thread 1: 1
Thread 2: 1
Thread 1: 2
Thread 2: 2
Thread 1: 3
Thread 2: 3
Thread 1: 4
Thread 2: 4
Thread 1: 5
Thread 2: 5
Thread 1: 6
Thread 2: 6
Thread 1: 7
Thread 2: 7
Thread 1: 8
Thread 2: 8
Thread 1: 9
Thread 2: 9
Thread 1: 10
Thread 2: 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
package com.assig.thread;

public class ThreadLifecycleSimulation {

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

            try {
                Thread.sleep(1000); // thread sleeps for 1 second
                System.out.println("Thread state: " +
Thread.currentThread().getState()); //  state runneable
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            synchronized (ThreadLifecycleSimulation.class) {
                try {
                    ThreadLifecycleSimulation.class.wait(); // Thread enters
waiting state
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println("Thread state: " +
Thread.currentThread().getState()); // timewaiting state
            try {
                Thread.sleep(2000); // Thread sleeps for 2 seconds
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

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

        System.out.println("Thread state: " + thread.getState()); //new state

        thread.start();

        try {
            Thread.sleep(500); // main thread sleeps for 0.5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Thread state: " + thread.getState()); // runnable
state

        synchronized (ThreadLifecycleSimulation.class) {
            ThreadLifecycleSimulation.class.notify(); // Thread transitions from
waiting to time waiting state
        }

        try {
            thread.join(); // Main thread waits for the child thread to terminate
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

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


Wipro_Java - java_Assignement/src/com/assig/thread/ThreadLifecycleSimulation.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer
Data_strutures
JRE System Library [JavaSE-17]
src
com.ds.linkedlist
com.ds.queue
com.ds.searching_algorithms
BinarySearch.java
Jumpsearch.java
LinearSearch.java
com.ds.searchingAlgorithms
com.ds.stack
com.ds.timeAndSpaceComplexity
com.ds.tree
module-info.java
First_java
java_Assignement
JRE System Library [JavaSE-17]
src
com.assig.thread
module-info.java
NumberPrinter.java
ThreadLifecycleSimulation.java
mvcstruture
NumberPrinter.java
module-info.java
ThreadLifecycleSimulation.java
Problems
Javadoc
Declaration
Console
Progress
ThreadLifecycleSimulation [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse
ENG
22:03
29-05-2024

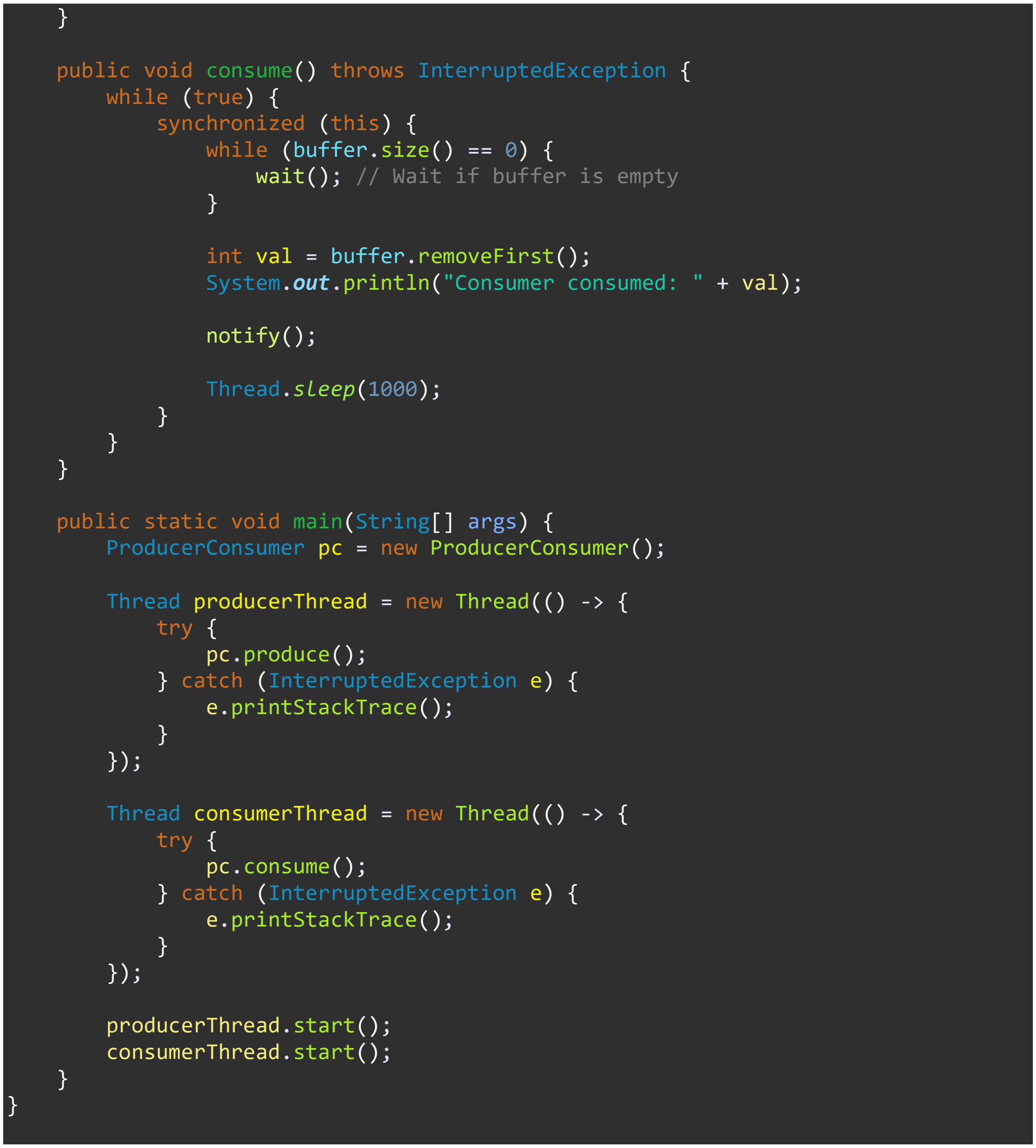
```java
package com.assig.thread;

public class ThreadLifecycleSimulation {

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

            try {
                Thread.sleep(1000); // thread sleeps for 1 second
                System.out.println("Thread state: " + Thread.curr
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            synchronized (ThreadLifecycleSimulation.class) {
                try {
                    ThreadLifecycleSimulation.class.wait(); // Th
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

            System.out.println("Thread state: " + Thread.currentTh
            try {
                Thread.sleep(2000); // Thread sleeps for 2 second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

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

        System.out.println("Thread state: " + thread.getState()); //new state

        thread.start();

        try {
            Thread.sleep(500); // main thread sleeps for 0.5 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

```

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

Task 3: Synchronization and Inter-thread Communication

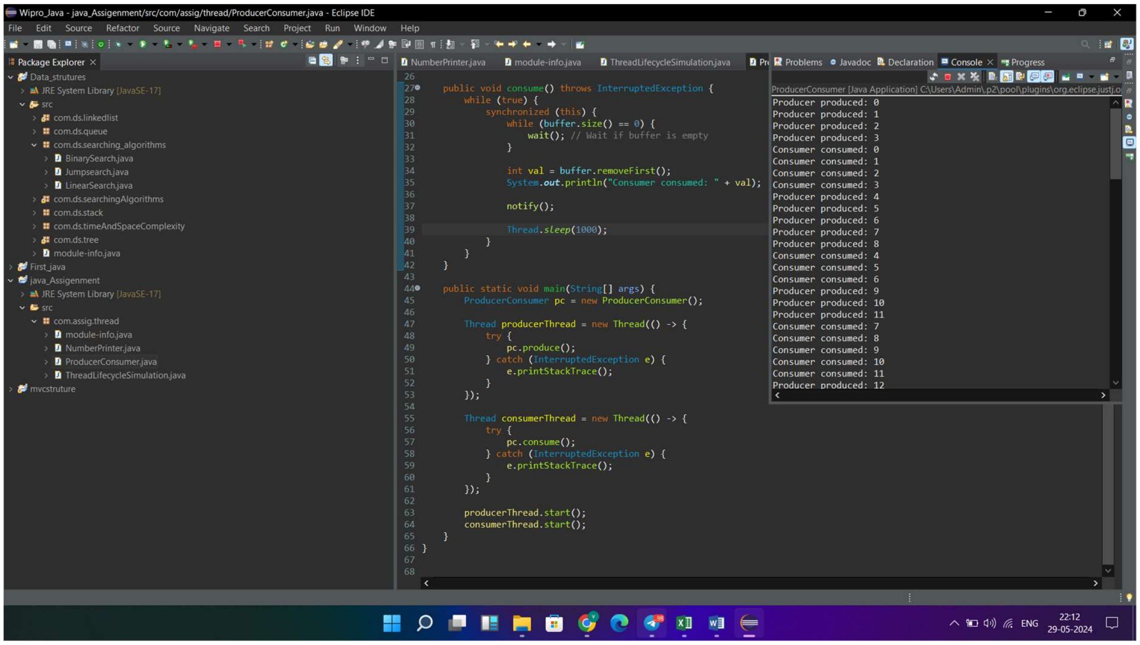
Implement a producer-consumer problem using wait() and notify() methods to handle the correct processing sequence between threads.

```java
package com.assig.thread;

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 {
                pc.produce();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

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

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

Op:


Wipro_Java - java_Assignement/src/com/assig/thread/ProducerConsumer.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer
NumberPrinter.java
module-info.java
ThreadLifecycleSimulation.java
Problems
Javadoc
Declaration
Console
Progress

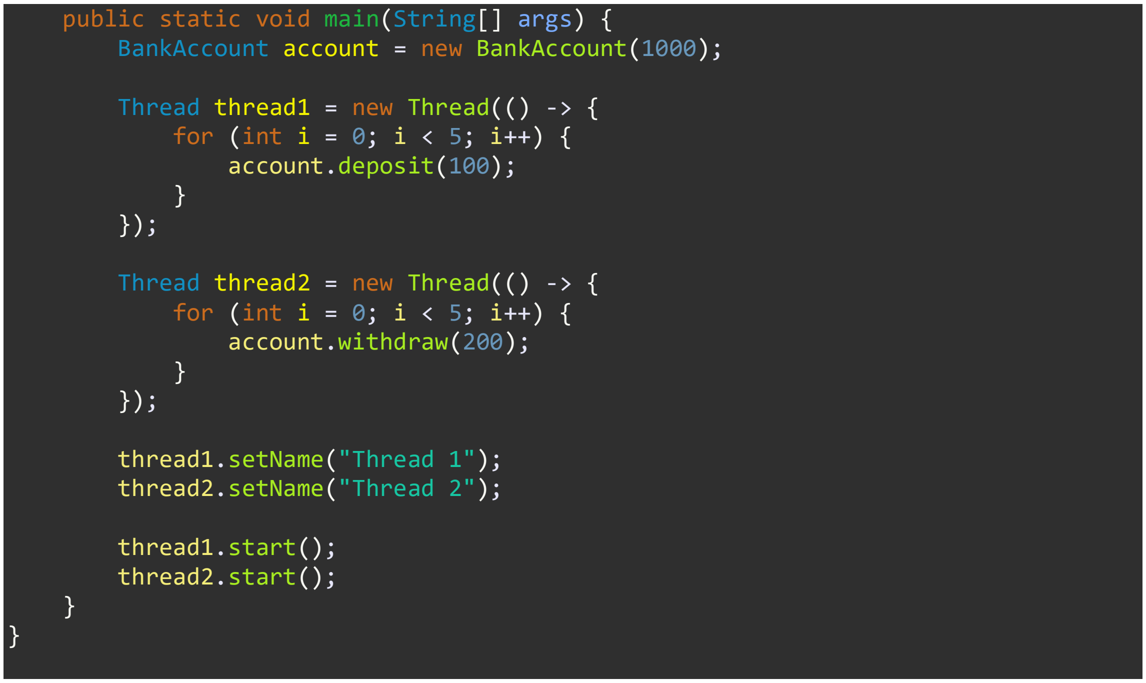
```java
    public void consume() throws InterruptedException {
        while (true) {
            synchronized (this) {
                while (buffer.size() == 0) {
                    wait(); // Wait if buffer is empty
                }

                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 {
                pc.produce();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

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

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

```
ProducerConsumer [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.o
Producer produced: 0
Producer produced: 1
Producer produced: 2
Producer produced: 3
Consumer consumed: 0
Consumer consumed: 1
Consumer consumed: 2
Consumer consumed: 3
Producer produced: 4
Producer produced: 5
Producer produced: 6
Producer produced: 7
Producer produced: 8
Consumer consumed: 4
Consumer consumed: 5
Consumer consumed: 6
Producer produced: 9
Producer produced: 10
Producer produced: 11
Consumer consumed: 7
Consumer consumed: 8
Consumer consumed: 9
Consumer consumed: 10
Consumer consumed: 11
Producer produced: 12
```

Task 4: Synchronized Blocks and Methods

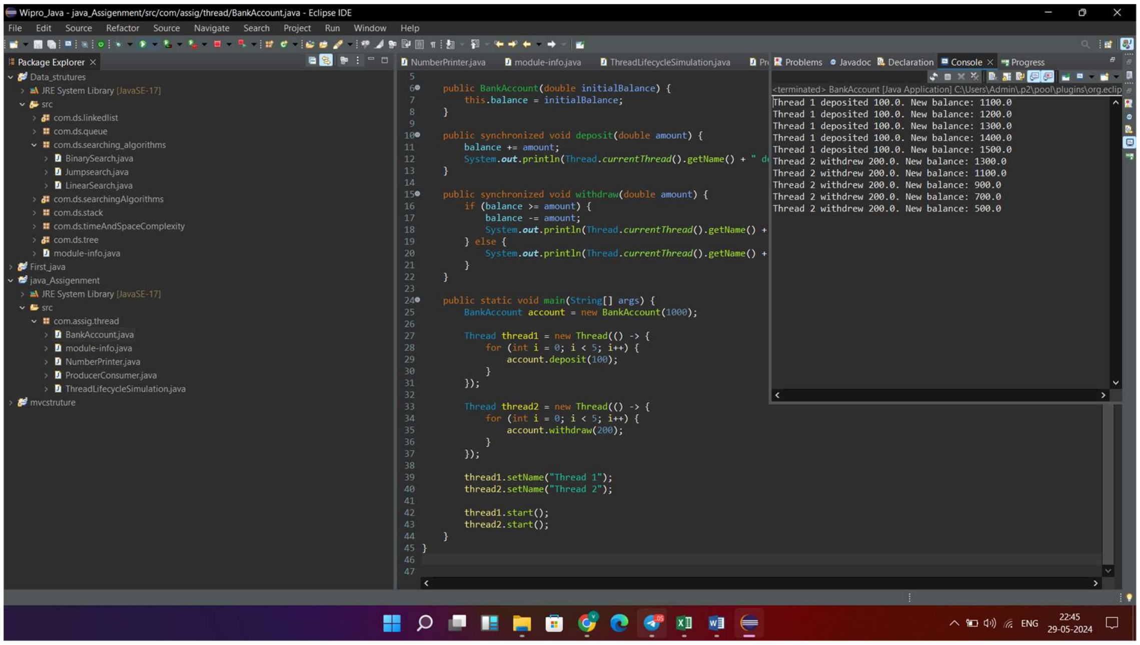
Write a program that simulates a bank account being accessed by multiple threads to perform deposits and withdrawals using synchronized methods to prevent race conditions.

```java
package com.assig.thread;

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.");
        }
    }
```



```java
    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");
        thread2.setName("Thread 2");

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

Op:-



```java

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

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

    public synchronized void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println(Thread.currentThread().getName() +
        } else {
            System.out.println(Thread.currentThread().getName() +
        }
    }

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

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

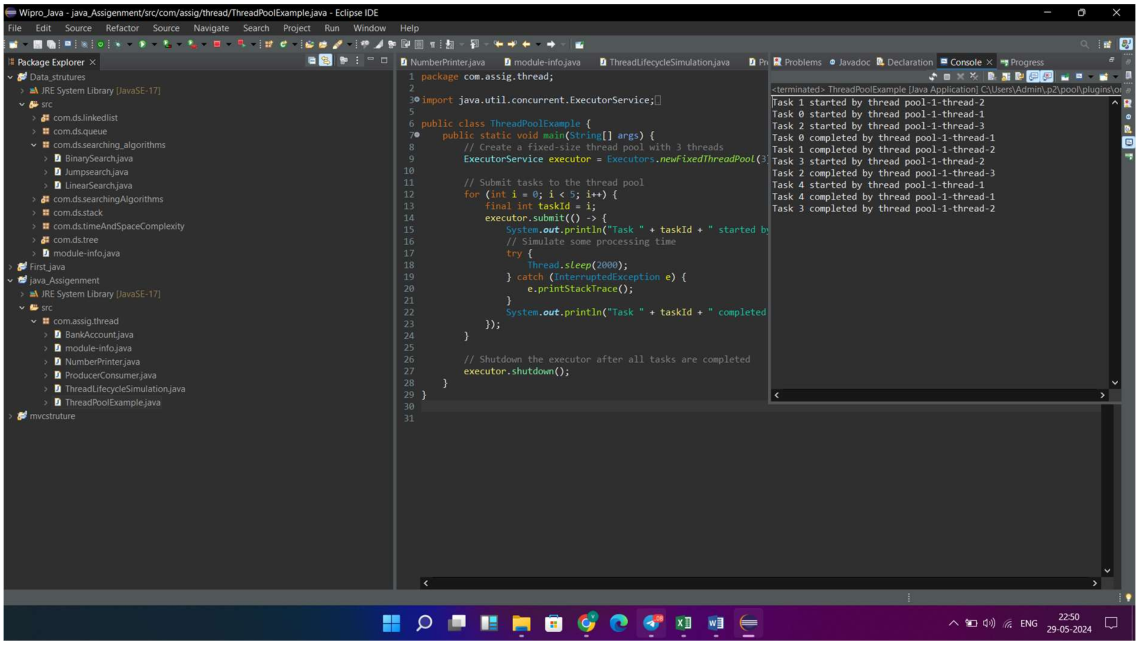
```
<terminated> BankAccount [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclip
Thread 1 deposited 100.0. New balance: 1100.0
Thread 1 deposited 100.0. New balance: 1200.0
Thread 1 deposited 100.0. New balance: 1300.0
Thread 1 deposited 100.0. New balance: 1400.0
Thread 1 deposited 100.0. New 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

```
package com.assig.thread;


import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;


public class ThreadPoolExample {

    public static void main(String[] args) {

        // Create a fixed-size thread pool with 3 threads

        ExecutorService executor = Executors.newFixedThreadPool(3);


        // Submit tasks to the thread pool

        for (int i = 0; i < 5; i++) {

            final int taskId = i;

            executor.submit(() -> {

                System.out.println("Task " + taskId + " started by thread " +
Thread.currentThread().getName());

                // Simulate some processing time

                try {

                    Thread.sleep(2000);

                } catch (InterruptedException e) {

                    e.printStackTrace();
                }
                System.out.println("Task " + taskId + " completed by thread " + Thread.currentThread().getName());
            });
        }
        executor.shutdown();
    }
}
```

Op:-



Wipro_Java - java_Assignement/src/com/assig/thread/ThreadPoolExample.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer
NumberPrinter.java
module-info.java
ThreadLifecycleSimulation.java
Problems
Javadoc
Declaration
Console
Progress

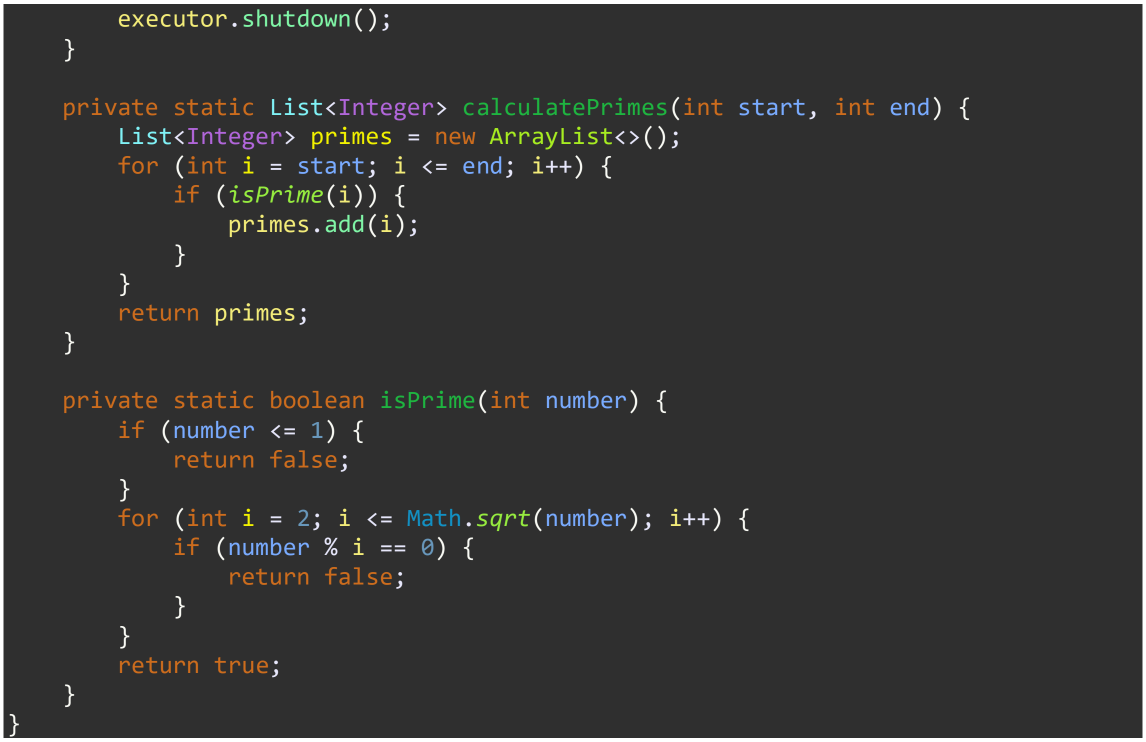
```
package com.assig.thread;

import java.util.concurrent.ExecutorService;

public class ThreadPoolExample {
    public static void main(String[] args) {
        // Create a fixed-size thread pool with 3 threads
        ExecutorService executor = Executors.newFixedThreadPool(3
        
        // Submit tasks to the thread pool
        for (int i = 0; i < 5; i++) {
            final int taskId = i;
            executor.submit(() -> {
                System.out.println("Task " + taskId + " started by
                // Simulate some processing time
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Task " + taskId + " completed
            });
        }
        
        // Shutdown the executor after all tasks are completed
        executor.shutdown();
    }
}
```

```
<terminated> ThreadPoolExample [Java Application] C:\Users\Admin\.p2\pool\plugins\or
Task 1 started by thread pool-1-thread-2
Task 0 started by thread pool-1-thread-1
Task 2 started by thread pool-1-thread-3
Task 0 completed by thread pool-1-thread-1
Task 1 completed by thread pool-1-thread-2
Task 3 started by thread pool-1-thread-2
Task 2 completed by thread pool-1-thread-3
Task 4 started by thread pool-1-thread-1
Task 4 completed by thread pool-1-thread-1
Task 3 completed by thread pool-1-thread-2
```

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.

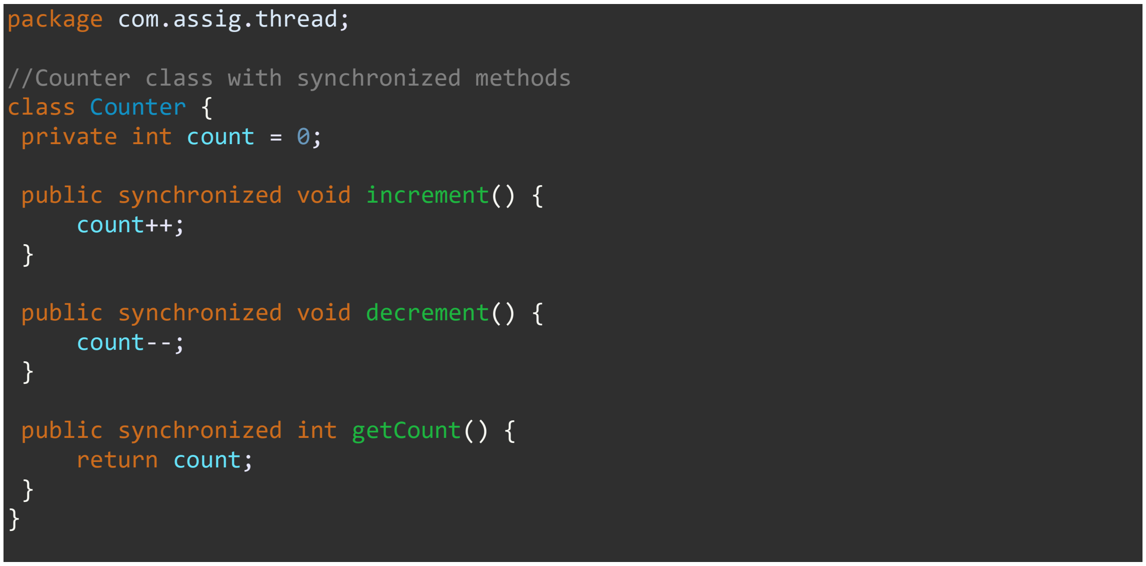
```
package com.assig.thread;

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;

    public static void main(String[] args) {
        int maxNumber = 100;
        ExecutorService executor = Executors.newFixedThreadPool(THREAD_COUNT);

        // Calculate prime numbers in parallel
        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);
        }

        // Combine results from all threads
        CompletableFuture<List<Integer>> combinedFuture = CompletableFuture.allOf(
                futures.toArray(new CompletableFuture[0]))
                .thenApply(v -> futures.stream()
                        .map(CompletableFuture::join)
                        .flatMap(List::stream)
                        .collect(Collectors.toList()));

        // Write results to file asynchronously
        combinedFuture.thenAcceptAsync(primes -> {
            try (BufferedWriter writer = new BufferedWriter(new
FileWriter("primes.txt"))) {
                for (Integer prime : primes) {
                    writer.write(prime.toString());
                    writer.newLine();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }, executor);

        // Shutdown the executor
```



```
        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.

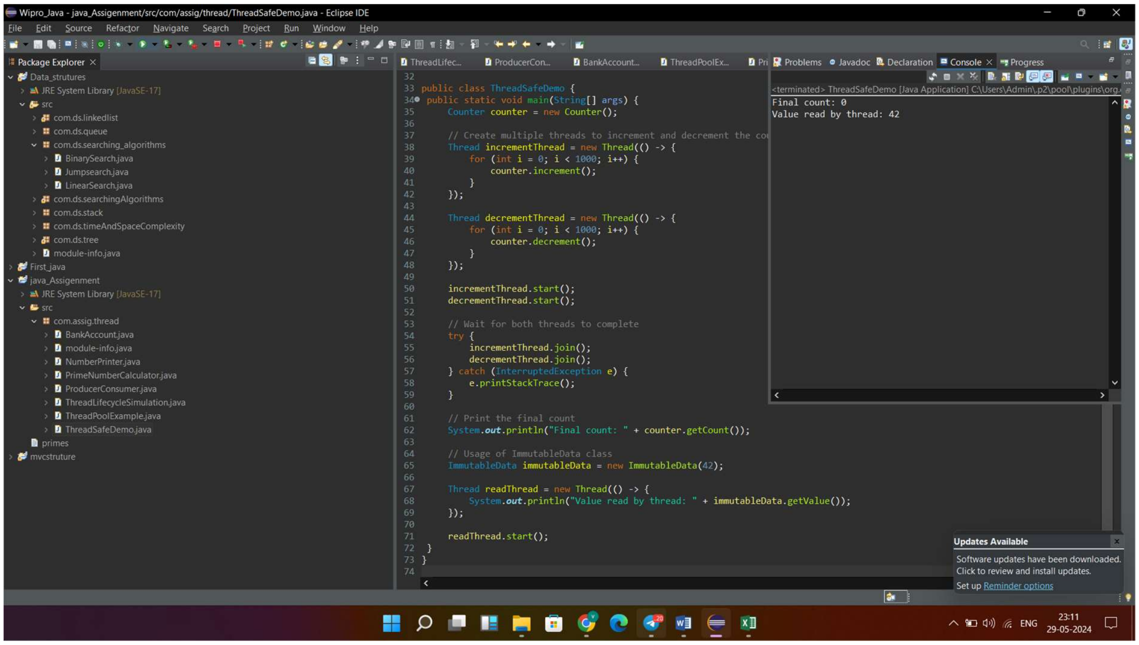


```
package com.assig.thread;

//Counter class with synchronized methods
class Counter {
 private int count = 0;

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

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

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

```
//Immutable class to share data between threads
final class ImmutableData {
 private final int value;

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

 public int getValue() {
     return value;
 }
}

public class ThreadSafeDemo {
 public static void main(String[] args) {
     Counter counter = new Counter();

     // Create multiple threads to increment and decrement the 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();

     // Wait for both threads to complete
     try {
         incrementThread.join();
         decrementThread.join();
     } catch (InterruptedException e) {
         e.printStackTrace();
     }

     // Print the final count
     System.out.println("Final count: " + counter.getCount());

     // Usage of ImmutableData class
     ImmutableData immutableData = new ImmutableData(42);

     Thread readThread = new Thread(() -> {
         System.out.println("Value read by thread: " + immutableData.getValue());
     });

     readThread.start();
 }
}
```

Op

Vijay patil Thread assignment 1-7



```java

public class ThreadSafeDemo {
 public static void main(String[] args) {
     Counter counter = new Counter();

     // Create multiple threads to increment and decrement the co
     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();

     // Wait for both threads to complete
     try {
         incrementThread.join();
         decrementThread.join();
     } catch (InterruptedException e) {
         e.printStackTrace();
     }

     // Print the final count
     System.out.println("Final count: " + counter.getCount());

     // Usage of ImmutableData class
     ImmutableData immutableData = new ImmutableData(42);

     Thread readThread = new Thread(() -> {
         System.out.println("Value read by thread: " + immutableData.getValue());
     });

     readThread.start();
 }
}

```

```
<terminated> ThreadSafeDemo [Java Application] C:\Users\Admin\.p2\pool\plugins\org.
Final count: 0
Value read by thread: 42
```

Updates Available

Software updates have been downloaded. Click to review and install updates.

Set up Reminder options

Vijay patil Thread assignment 1-7