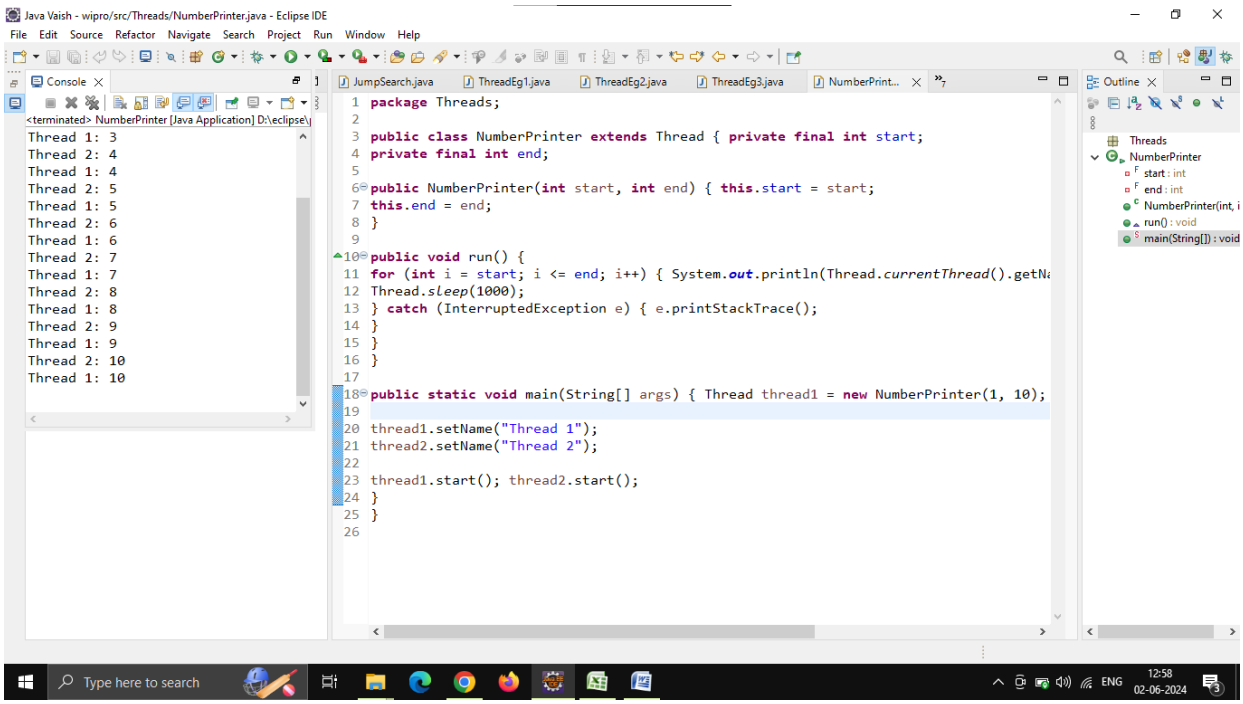
Day:18

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

```
package 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();
}
}
```


Java Vaish - wipro/src/Threads/NumberPrinter.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Console
<terminated> NumberPrinter [Java Application] D:\eclipse\
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
JumpSearch.java
ThreadEg1.java
ThreadEg2.java
ThreadEg3.java
NumberPrint...
1 package Threads;
2
3 public class NumberPrinter extends Thread { private final int start;
4 private final int end;
5
6 public NumberPrinter(int start, int end) { this.start = start;
7 this.end = end;
8 }
9
10 public void run() {
11 for (int i = start; i <= end; i++) { System.out.println(Thread.currentThread().getNa
12 Thread.sleep(1000);
13 } catch (InterruptedException e) { e.printStackTrace();
14 }
15 }
16 }
17
18 public static void main(String[] args) { Thread thread1 = new NumberPrinter(1, 10);
19
20 thread1.setName("Thread 1");
21 thread2.setName("Thread 2");
22
23 thread1.start(); thread2.start();
24 }
25 }
26
Outline
Threads
NumberPrinter
start : int
end : int
NumberPrinter(int, i
run() : void
main(String[]) : void
Type here to search
ENG
12:58
02-06-2024

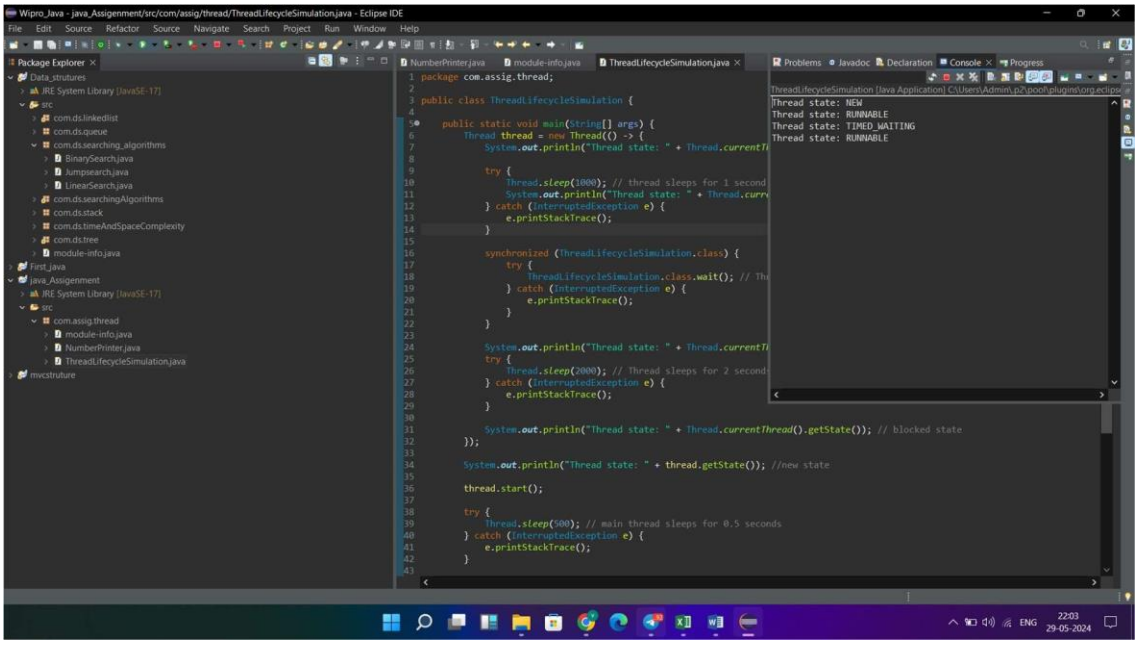
Task 2: States and Transitions

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

```
package com.assig.thread;


public class ThreadLifecycleSimulation {

    public static void main(String[] args) {

        Thread thread = new Thread(() -> {

            // Initial state of the thread

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


            try {

                // Thread sleeps for 1 second

                Thread.sleep(1000);

                // After sleeping, thread state is RUNNABLE

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

            } catch (InterruptedException e) {

                e.printStackTrace();

            }


            // Thread enters waiting state

            synchronized (ThreadLifecycleSimulation.class) {

                try {
```

```
                ThreadLifecycleSimulation.class.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // After waking up, thread state is TIMED_WAITING
        System.out.println("Thread state: " +
Thread.currentThread().getState()); // TIMED_WAITING state
        try {
            // Thread sleeps for 2 seconds
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // After sleeping, thread state is BLOCKED
        System.out.println("Thread state: " +
Thread.currentThread().getState()); // BLOCKED state
    });

    // Initial state of the thread
    System.out.println("Thread state: " + thread.getState()); // NEW state
    // Starting the thread
    thread.start();

    try {
        // Main thread sleeps for 0.5 seconds
        Thread.sleep(500);
```

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // After sleeping, thread state is RUNNABLE
        System.out.println("Thread state: " + thread.getState()); // RUNNABLE
state

        // Notify the waiting thread to continue
        synchronized (ThreadLifecycleSimulation.class) {
            ThreadLifecycleSimulation.class.notify();
        }

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

        // After the thread terminates, its state is TERMINATED
        System.out.println("Thread state: " + thread.getState()); // TERMINATED
state
    }
}
```

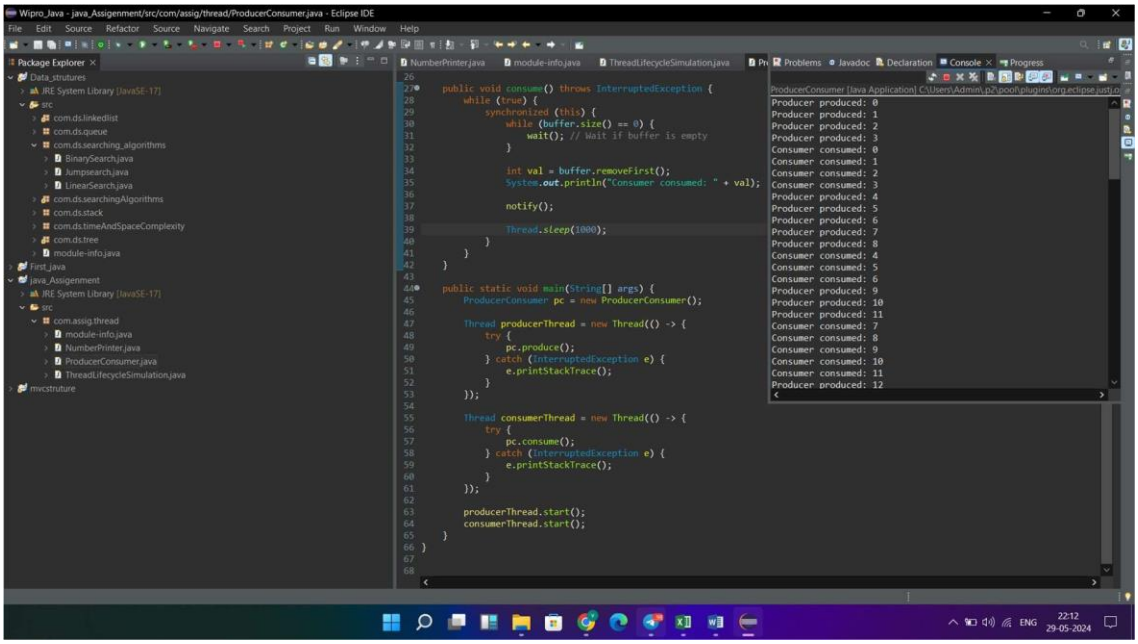

Wipro_Java - java_Assignment/src/com/assig/thread/ThreadLifecycleSimulation.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Package Explorer
Data_strutures
JRE System Library [JavaSE-17]
src
com.ds.linkedlist
com.ds.queue
com.ds.searching_algorithms
BinarySearch.java
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
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.curre
} 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();
}
Problems Javadoc Declaration Console Progress
ThreadLifecycleSimulation [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse
Thread state: NEW
Thread state: RUNNABLE
Thread state: TIMED_WAITING
Thread state: RUNNABLE
ENG
22:03
29-05-2024

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.

```
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) {
                // Wait while buffer is full
                while (buffer.size() == capacity) {
                    wait();
                }

                // Produce an item and add it to the buffer
                System.out.println("Producer produced: " + value);
                buffer.add(value++);

                // Notify consumer that an item is available
```

```
            notify();

            // Simulate some processing time
            Thread.sleep(1000);
        }
    }
}
```

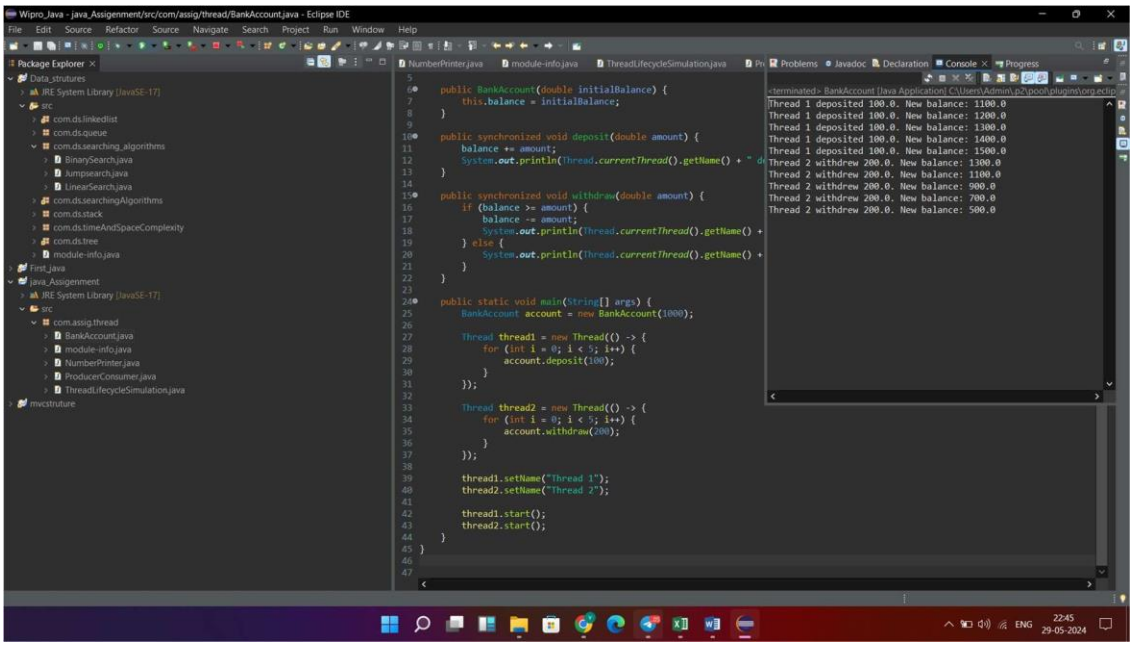


Task 4: Synchronized Blocks and Methods

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.

```
package com.assig.thread;


public class BankAccount {
    private double balance;

    // Constructor to initialize the initial balance
    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    // Synchronized method to deposit money into the account
    public synchronized void deposit(double amount) {
        balance += amount;
        System.out.println(Thread.currentThread().getName() + " deposited " +
                amount + ". New balance: " + balance);
    }

    // Synchronized method to withdraw money from the account
    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 "
```



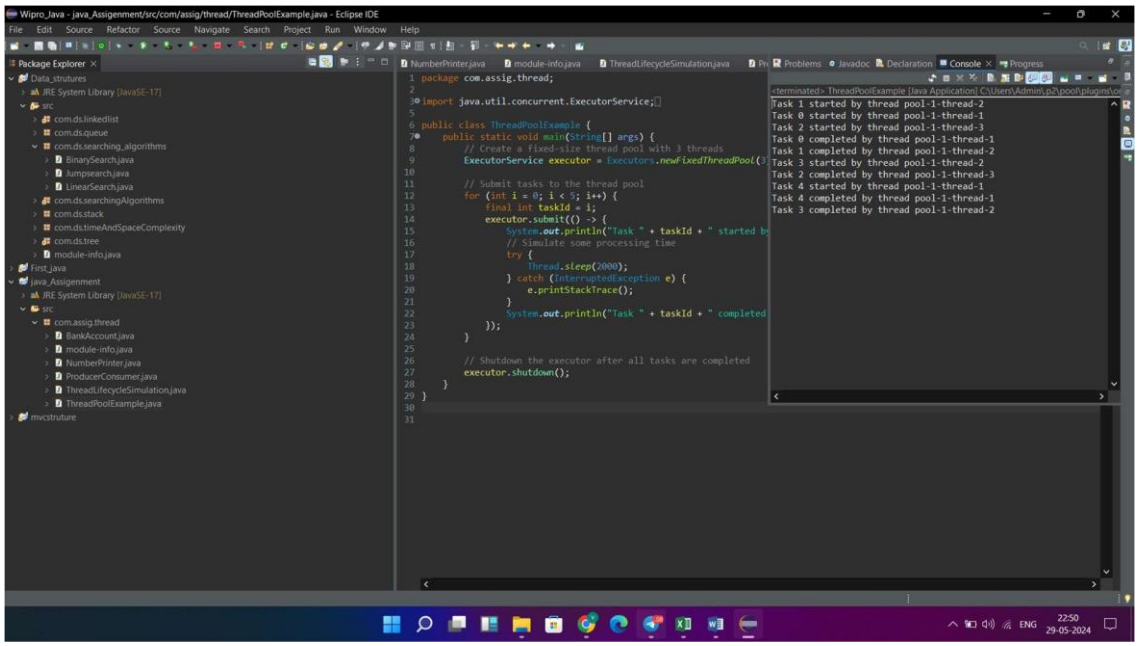

```
+ amount + " but insufficient funds.");
        }
    }
}
```

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();
  }
}
```



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();
    }

    // Method to calculate prime numbers within a range
    private static List<Integer> calculatePrimes(int start, int end) {
        List<Integer> primes = new ArrayList<>();
        for (int number = start; number <= end; number++) {
            if (isPrime(number)) {
                primes.add(number);
            }
        }
        return primes;
    }

    // Method to check if a number is prime
    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;
    }
}
```

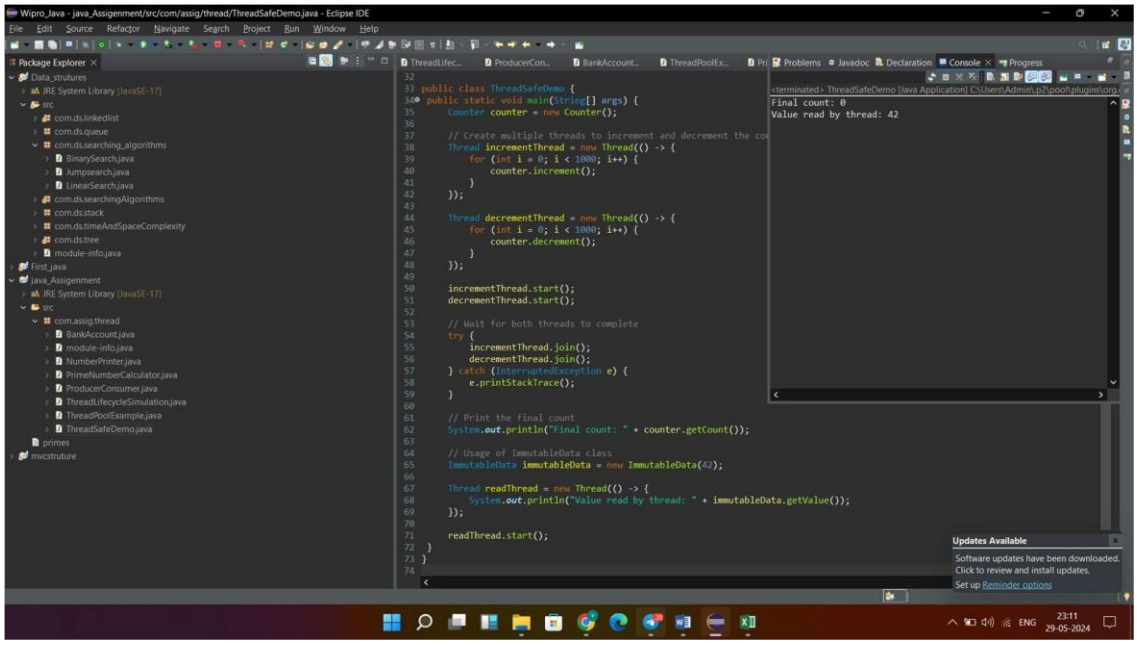
Task 7: Writing Thread-Safe Code, Immutable Objects

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();
    }
}
```


Wipro_Java - java_Assigenment/src/com/assig/thread/ThreadSafeDemo.java - Eclipse IDE
File Edit Source Refactor 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_Assigenment
JRE System Library [JavaSE-17]
src
com.assig.thread
BankAccount.java
module-info.java
NumberPrinter.java
PrimeNumberCalculator.java
ProducerConsumer.java
ThreadLifecycleSimulation.java
ThreadPoolExample.java
ThreadSafeDemo.java
primes
mvcstruture
ThreadLifec...
ProducerCon...
BankAccount...
ThreadPoolEx...
Problems
Javadoc
Declaration
Console
Progress
<terminated> ThreadSafeDemo [Java Application] C:\Users\Admin\.p2\pool\plugins\org.
Final count: 0
Value read by thread: 42
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();
}
}
Updates Available
Software updates have been downloaded.
Click to review and install updates.
Set up Reminder options
ENG
23:11
29-05-2024