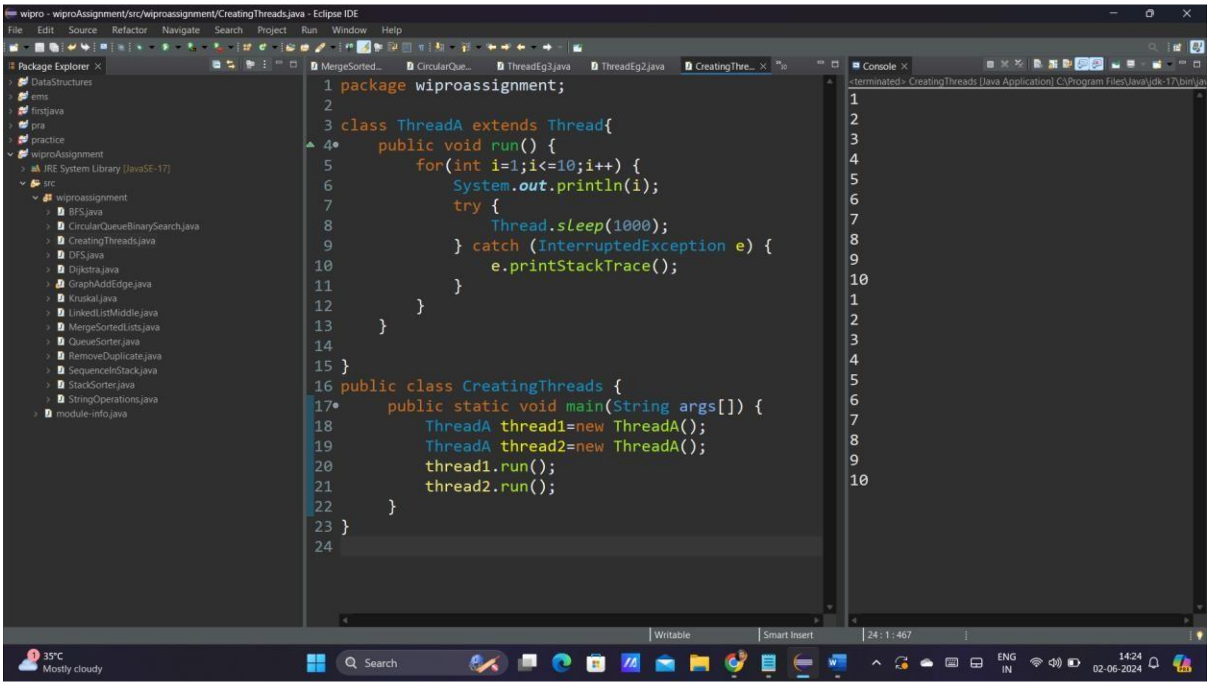
Assignments

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.

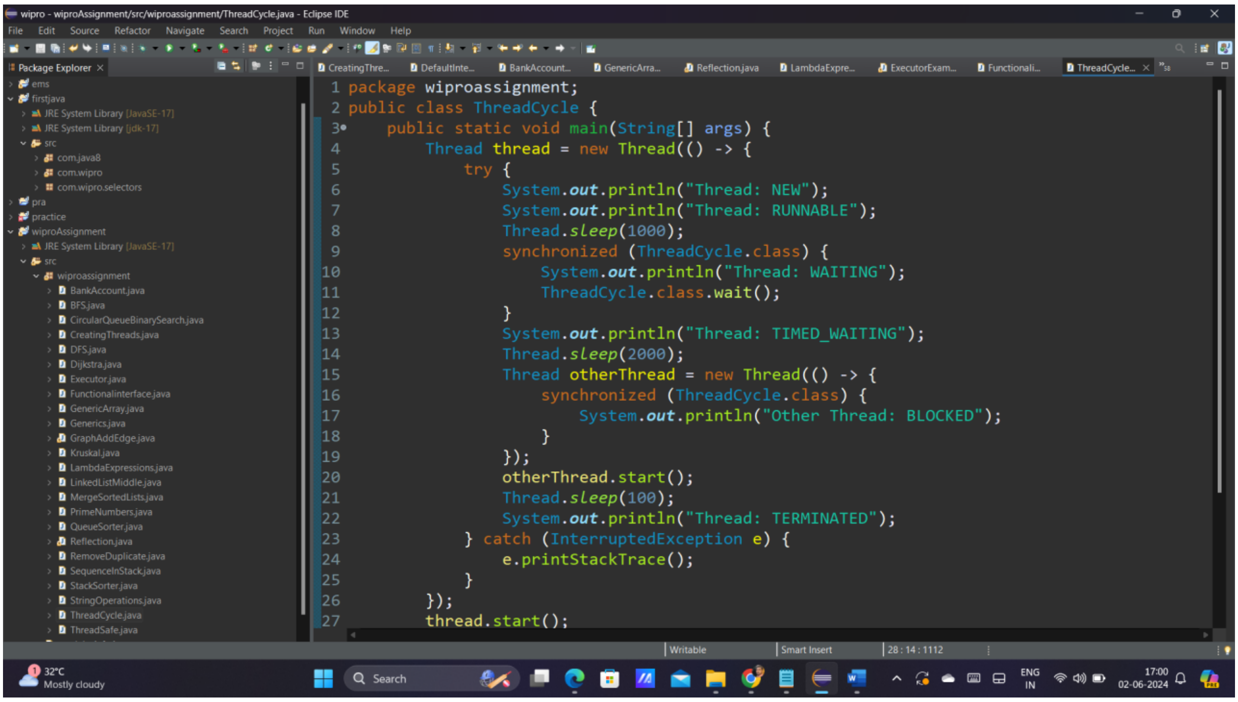


```java
package wiproassignment;

class ThreadA extends Thread{
    public void run() {
        for(int i=1;i<=10;i++) {
            System.out.println(i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

}
public class CreatingThreads {
     public static void main(String args[]) {
        ThreadA thread1=new ThreadA();
        ThreadA thread2=new ThreadA();
        thread1.run();
        thread2.run();
     }
}
```

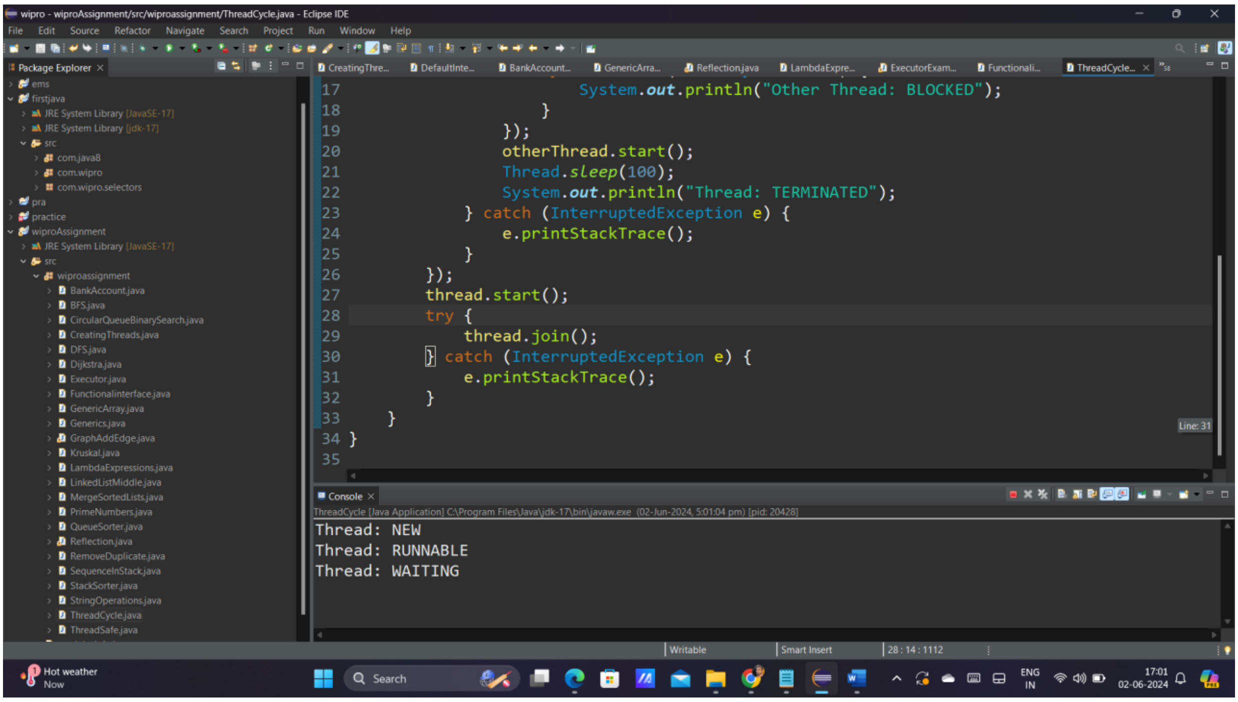
```
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
```

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..


wipro - wiproAssignment/src/wiproassignment/ThreadCycle.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
ThreadCycle.java
Writable
Smart Insert
28 : 14 : 1112

```java
package wiproassignment;
public class ThreadCycle {
    public static void main(String[] args) {
        Thread thread = new Thread(() -> {
            try {
                System.out.println("Thread: NEW");
                System.out.println("Thread: RUNNABLE");
                Thread.sleep(1000);
                synchronized (ThreadCycle.class) {
                    System.out.println("Thread: WAITING");
                    ThreadCycle.class.wait();
                }
                System.out.println("Thread: TIMED_WAITING");
                Thread.sleep(2000);
                Thread otherThread = new Thread(() -> {
                    synchronized (ThreadCycle.class) {
                        System.out.println("Other Thread: BLOCKED");
                    }
                });
                otherThread.start();
                Thread.sleep(100);
                System.out.println("Thread: TERMINATED");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        thread.start();
```

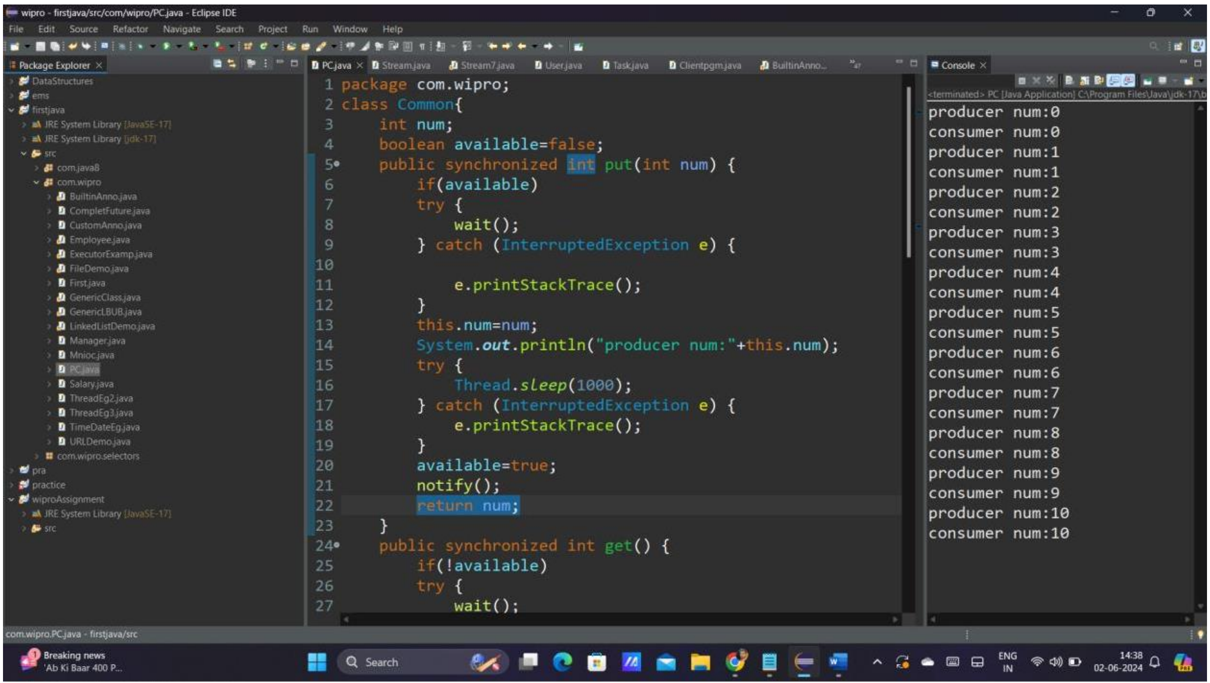

wipro - wiproAssignment/src/wiproassignment/ThreadCycle.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
ThreadCycle.java
Console
Writable
Smart Insert
28 : 14 : 1112

```java
                        System.out.println("Other Thread: BLOCKED");
                    }
                });
                otherThread.start();
                Thread.sleep(100);
                System.out.println("Thread: TERMINATED");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        thread.start();
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
ThreadCycle [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-Jun-2024, 5:01:04 pm) [pid: 20428]
Thread: NEW
Thread: RUNNABLE
Thread: WAITING
```

Task 3: Synchronization and Inter-thread Communication

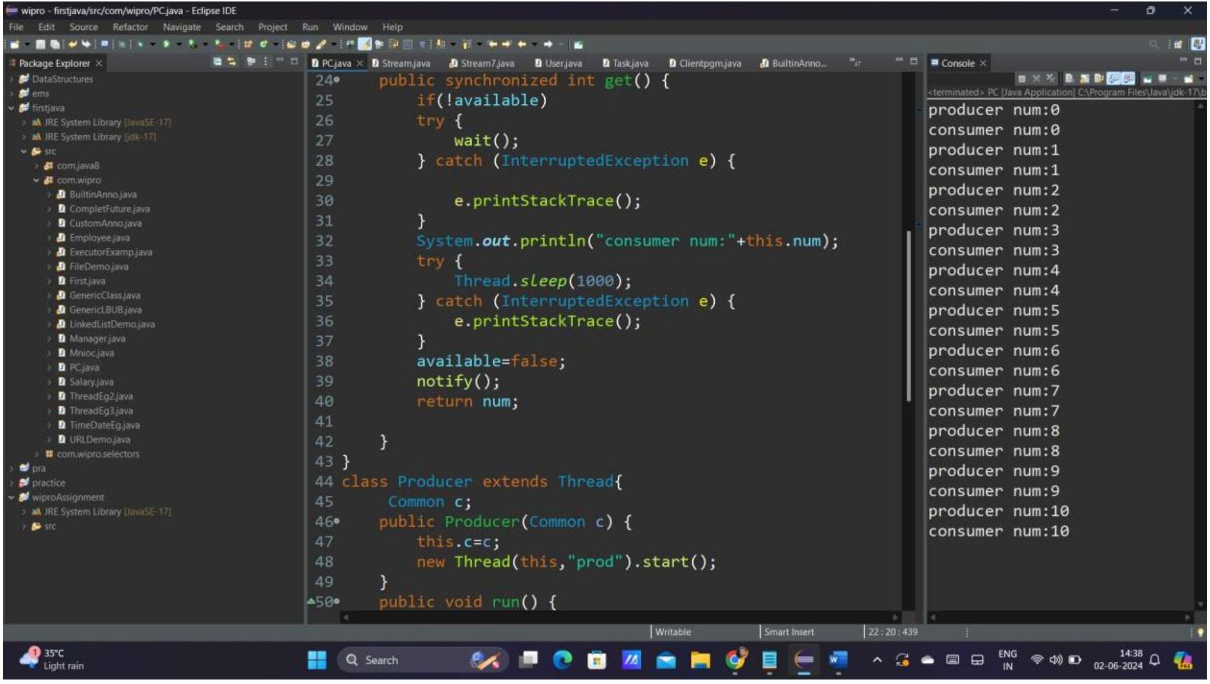
Implement a producer-consumer problem using wait() and notify() methods to handle the correct processing sequence between threads.



wipro - firstjava/src/com/wipro/PC.java - Eclipse IDE

```java
package com.wipro;
class Common{
    int num;
    boolean available=false;
    public synchronized int put(int num) {
        if(available)
        try {
            wait();
        } catch (InterruptedException e) {

            e.printStackTrace();
        }
        this.num=num;
        System.out.println("producer num:"+this.num);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        available=true;
        notify();
        return num;
    }
    public synchronized int get() {
        if(!available)
        try {
            wait();
```

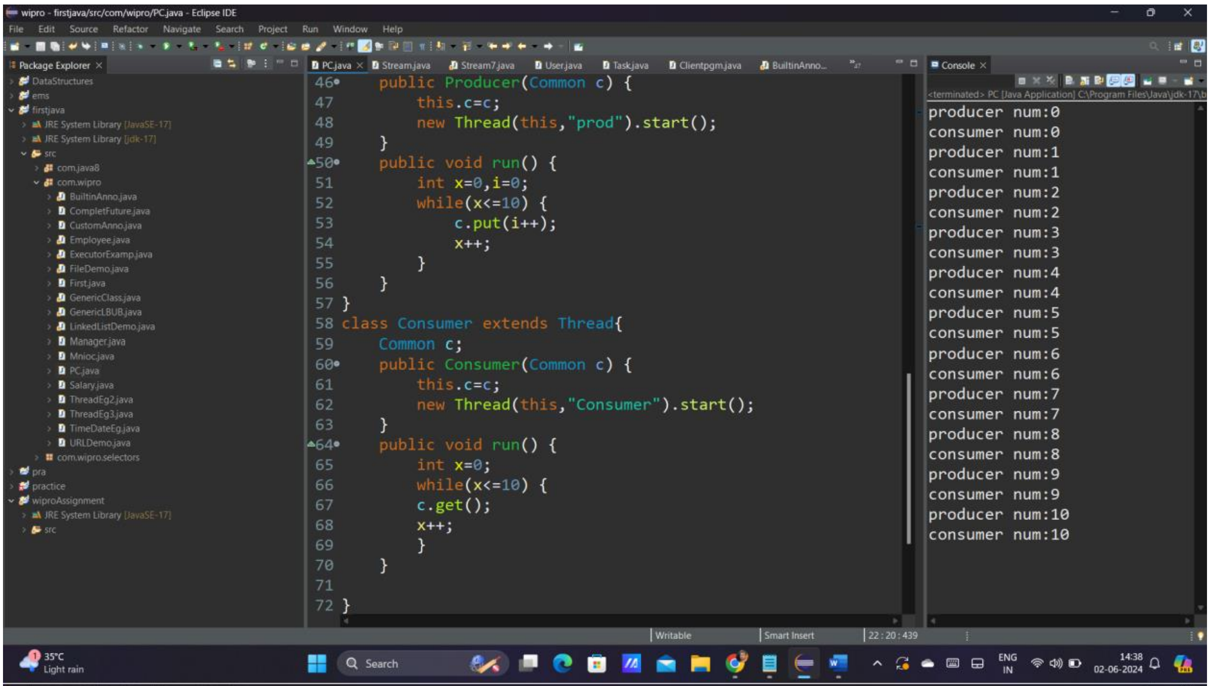
```
producer num:0
consumer num:0
producer num:1
consumer num:1
producer num:2
consumer num:2
producer num:3
consumer num:3
producer num:4
consumer num:4
producer num:5
consumer num:5
producer num:6
consumer num:6
producer num:7
consumer num:7
producer num:8
consumer num:8
producer num:9
consumer num:9
producer num:10
consumer num:10
```



wipro - firstjava/src/com/wipro/PC.java - Eclipse IDE

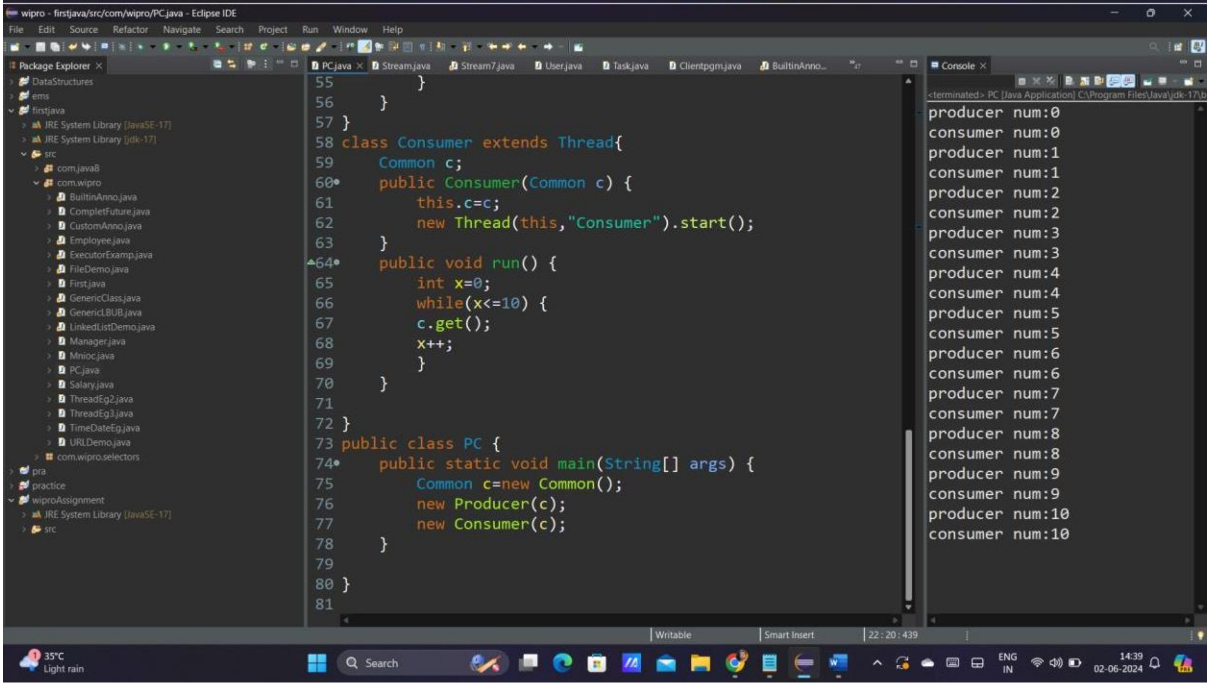
```java
    public synchronized int get() {
        if(!available)
        try {
            wait();
        } catch (InterruptedException e) {

            e.printStackTrace();
        }
        System.out.println("consumer num:"+this.num);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        available=false;
        notify();
        return num;

    }
}
class Producer extends Thread{
     Common c;
    public Producer(Common c) {
        this.c=c;
        new Thread(this,"prod").start();
    }
    public void run() {
```

```
producer num:0
consumer num:0
producer num:1
consumer num:1
producer num:2
consumer num:2
producer num:3
consumer num:3
producer num:4
consumer num:4
producer num:5
consumer num:5
producer num:6
consumer num:6
producer num:7
consumer num:7
producer num:8
consumer num:8
producer num:9
consumer num:9
producer num:10
consumer num:10
```



```java
    public Producer(Common c) {
        this.c=c;
        new Thread(this,"prod").start();
    }
    public void run() {
        int x=0,i=0;
        while(x<=10) {
            c.put(i++);
            x++;
        }
    }
}
class Consumer extends Thread{
    Common c;
    public Consumer(Common c) {
        this.c=c;
        new Thread(this,"Consumer").start();
    }
    public void run() {
        int x=0;
        while(x<=10) {
        c.get();
        x++;
        }
    }

}
```

```
<terminated> PC [Java Application] C:\Program Files\Java\jdk-17\b
producer num:0
consumer num:0
producer num:1
consumer num:1
producer num:2
consumer num:2
producer num:3
consumer num:3
producer num:4
consumer num:4
producer num:5
consumer num:5
producer num:6
consumer num:6
producer num:7
consumer num:7
producer num:8
consumer num:8
producer num:9
consumer num:9
producer num:10
consumer num:10
```

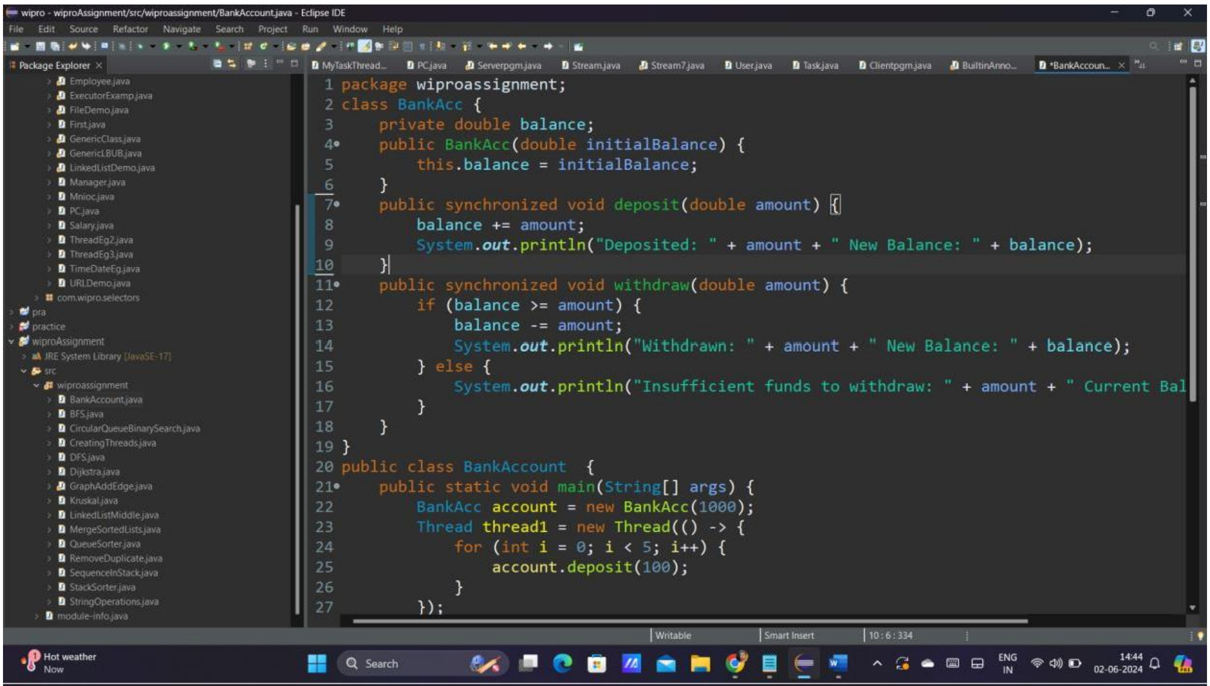


```java
        }
    }
}
class Consumer extends Thread{
    Common c;
    public Consumer(Common c) {
        this.c=c;
        new Thread(this,"Consumer").start();
    }
    public void run() {
        int x=0;
        while(x<=10) {
        c.get();
        x++;
        }
    }

}
public class PC {
    public static void main(String[] args) {
        Common c=new Common();
        new Producer(c);
        new Consumer(c);
    }

}

```

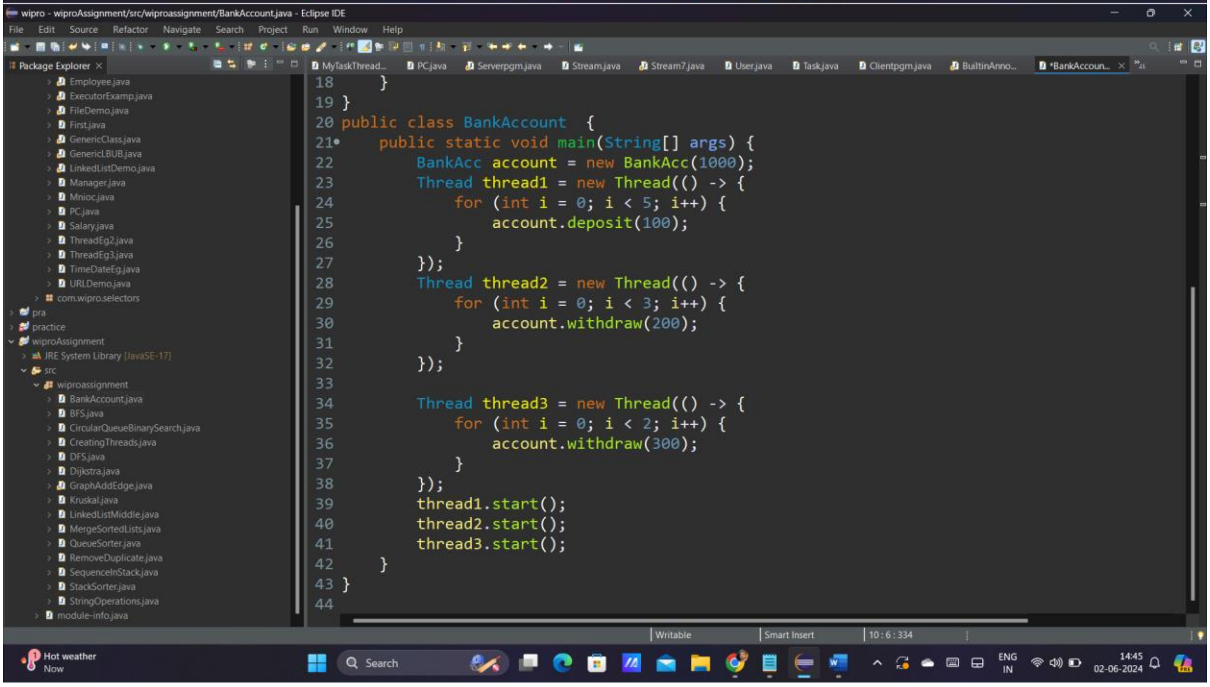
```
<terminated> PC [Java Application] C:\Program Files\Java\jdk-17\b
producer num:0
consumer num:0
producer num:1
consumer num:1
producer num:2
consumer num:2
producer num:3
consumer num:3
producer num:4
consumer num:4
producer num:5
consumer num:5
producer num:6
consumer num:6
producer num:7
consumer num:7
producer num:8
consumer num:8
producer num:9
consumer num:9
producer num:10
consumer num:10
```

Task 4: Synchronized Blocks and Methods

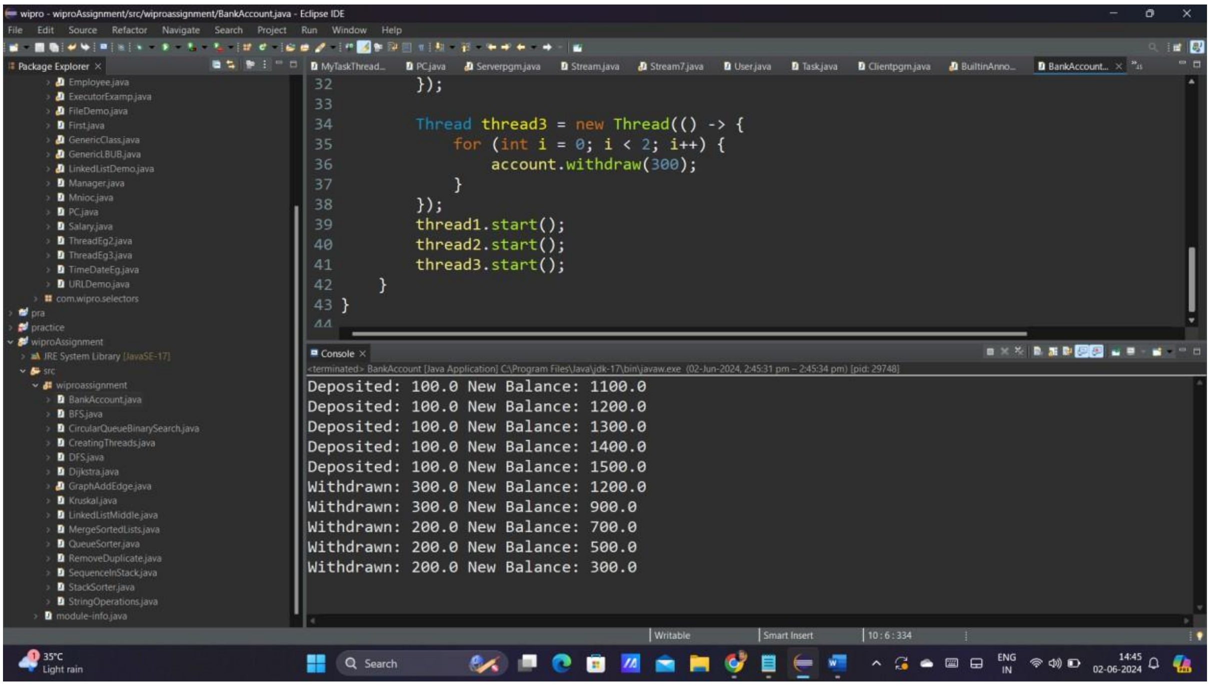
Write a program that simulates a bank account being accessed by multiple threads to perform deposits and withdrawals using synchronized methods to prevent race conditions.


wipro - wiproAssignment/src/wiproassignment/BankAccount.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
MyTaskThread... PCjava Serverpgm.java Stream.java Stream7.java User.java Task.java Clientpgm.java BuiltinAnno... *BankAccoun...
Writable
Smart Insert
10 : 6 : 334

```
package wiproassignment;
class BankAcc {
    private double balance;
    public BankAcc(double initialBalance) {
        this.balance = initialBalance;
    }
    public synchronized void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: " + amount + " New Balance: " + balance);
    }
    public synchronized void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount + " New Balance: " + balance);
        } else {
            System.out.println("Insufficient funds to withdraw: " + amount + " Current Bal
        }
    }
}
public class BankAccount  {
    public static void main(String[] args) {
        BankAcc account = new BankAcc(1000);
        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                account.deposit(100);
            }
        });
```


wipro - wiproAssignment/src/wiproassignment/BankAccount.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Writable
Smart Insert
10 : 6 : 334

```
    }
}
public class BankAccount  {
    public static void main(String[] args) {
        BankAcc account = new BankAcc(1000);
        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                account.deposit(100);
            }
        });
        Thread thread2 = new Thread(() -> {
            for (int i = 0; i < 3; i++) {
                account.withdraw(200);
            }
        });

        Thread thread3 = new Thread(() -> {
            for (int i = 0; i < 2; i++) {
                account.withdraw(300);
            }
        });
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

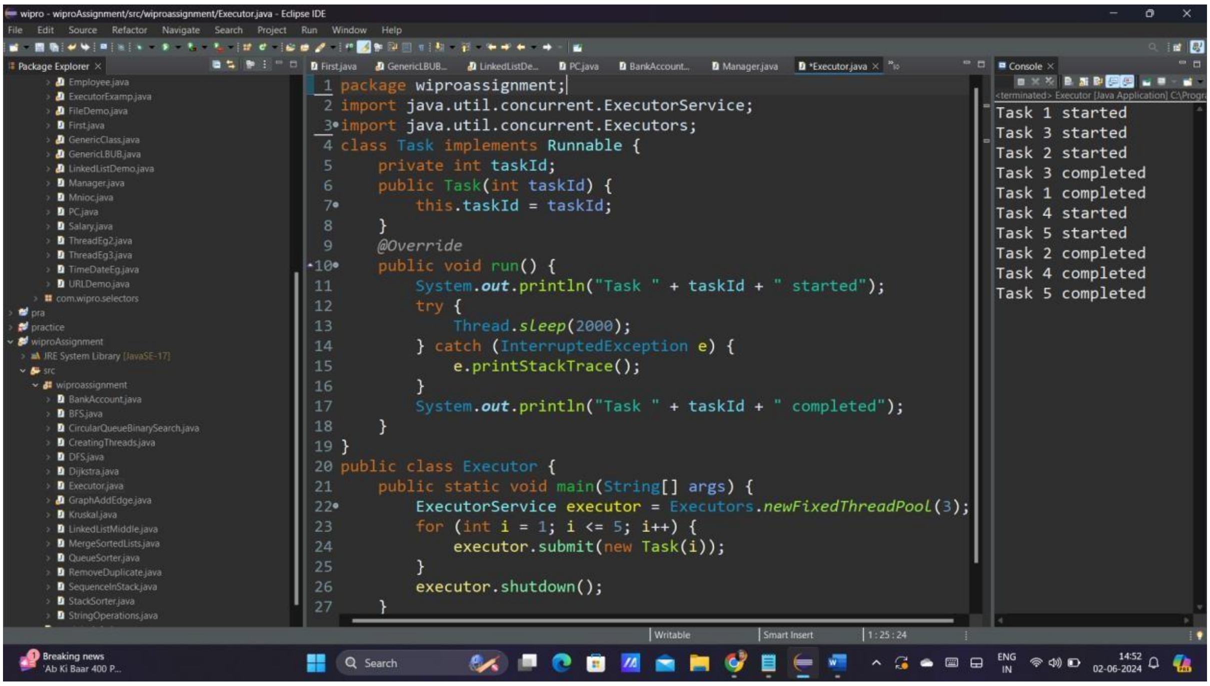


```
        });

        Thread thread3 = new Thread(() -> {
            for (int i = 0; i < 2; i++) {
                account.withdraw(300);
            }
        });
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

```
<terminated> BankAccount [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-Jun-2024, 2:45:31 pm – 2:45:34 pm) [pid: 29748]
Deposited: 100.0 New Balance: 1100.0
Deposited: 100.0 New Balance: 1200.0
Deposited: 100.0 New Balance: 1300.0
Deposited: 100.0 New Balance: 1400.0
Deposited: 100.0 New Balance: 1500.0
Withdrawn: 300.0 New Balance: 1200.0
Withdrawn: 300.0 New Balance: 900.0
Withdrawn: 200.0 New Balance: 700.0
Withdrawn: 200.0 New Balance: 500.0
Withdrawn: 200.0 New Balance: 300.0
```

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.

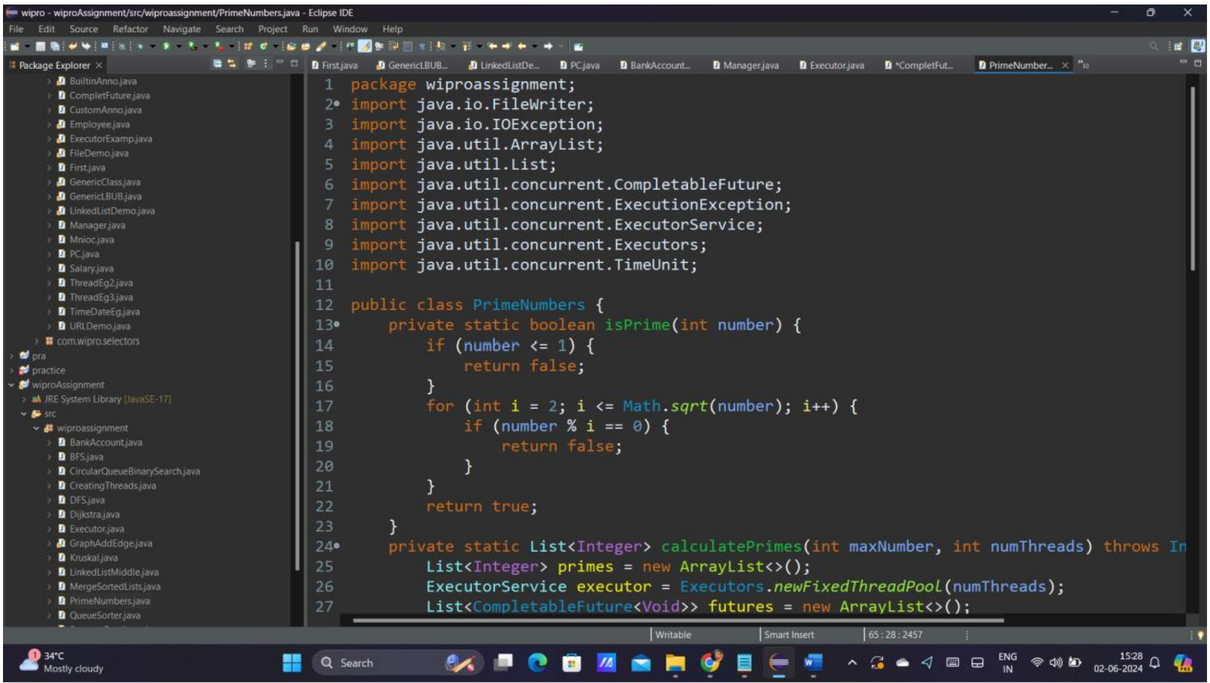


```
package wiproassignment;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class Task implements Runnable {
    private int taskId;
    public Task(int taskId) {
        this.taskId = taskId;
    }
    @Override
    public void run() {
        System.out.println("Task " + taskId + " started");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Task " + taskId + " completed");
    }
}
public class Executor {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(3);
        for (int i = 1; i <= 5; i++) {
            executor.submit(new Task(i));
        }
        executor.shutdown();
    }
```

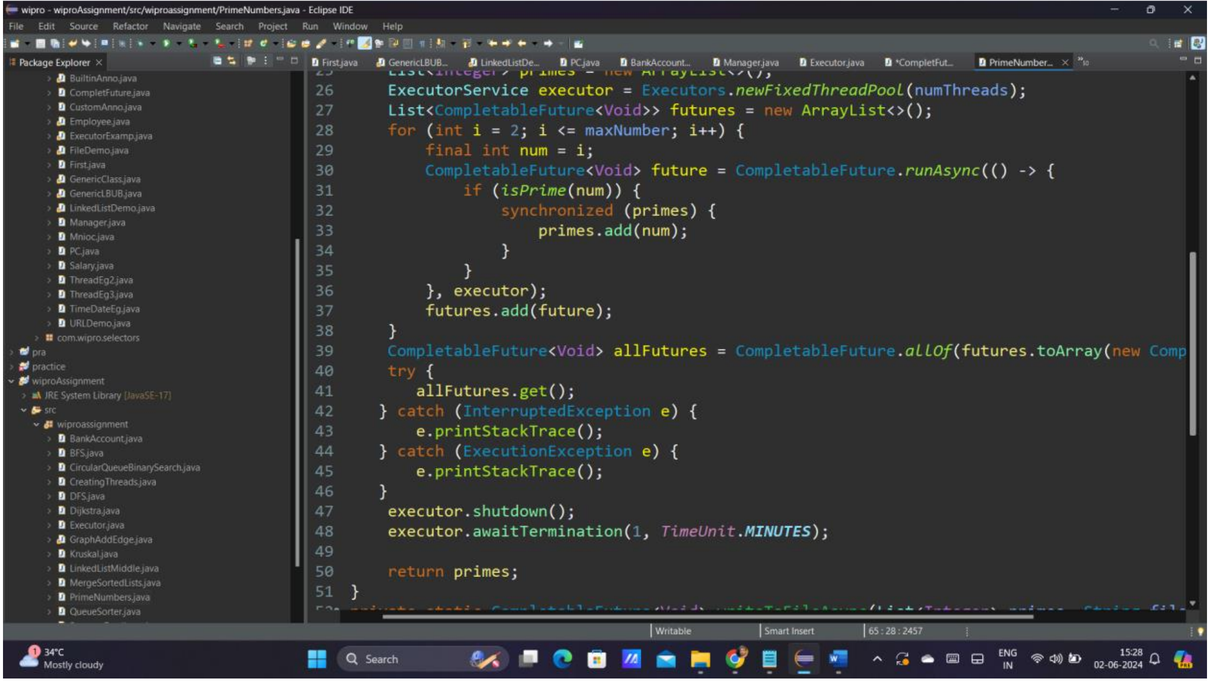
```
Task 1 started
Task 3 started
Task 2 started
Task 3 completed
Task 1 completed
Task 4 started
Task 5 started
Task 2 completed
Task 4 completed
Task 5 completed
```

Task 6: Executors, Concurrent Collections, CompletableFuture

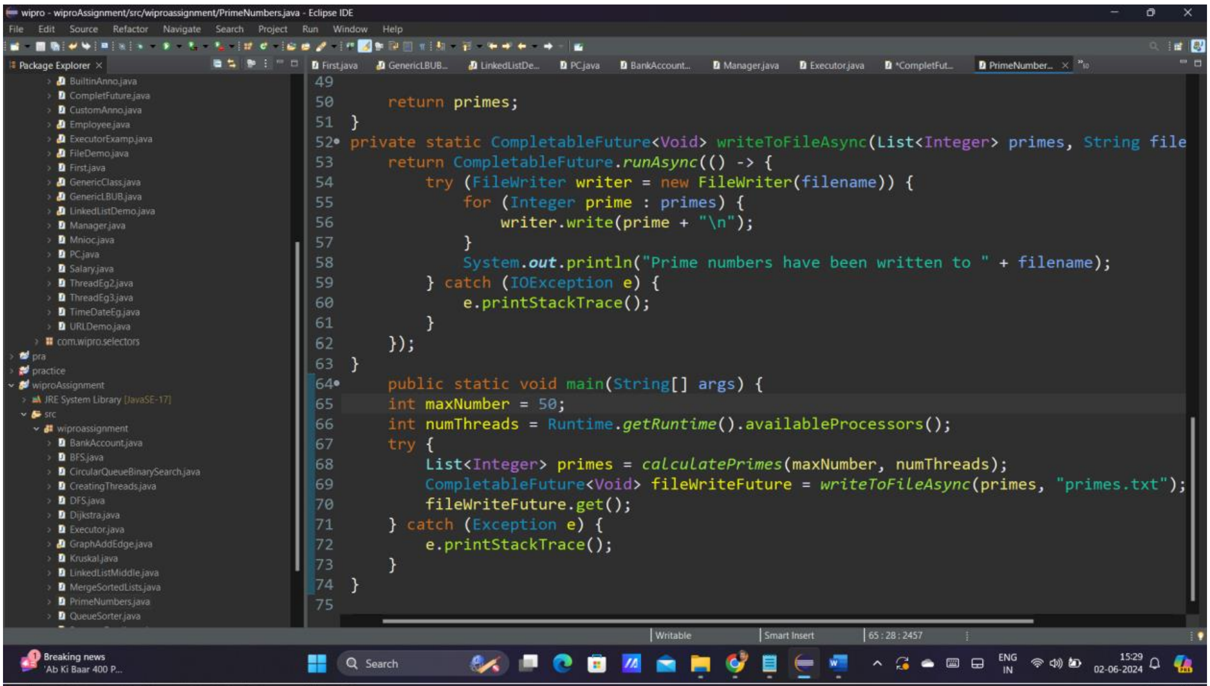
Use an ExecutorService to parallelize a task that calculates prime numbers up to a given number and then use CompletableFuture to write the results to a file asynchronously.


wipro - wiproAssignment/src/wiproassignment/PrimeNumbers.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Writable
Smart Insert
65 : 28 : 2457

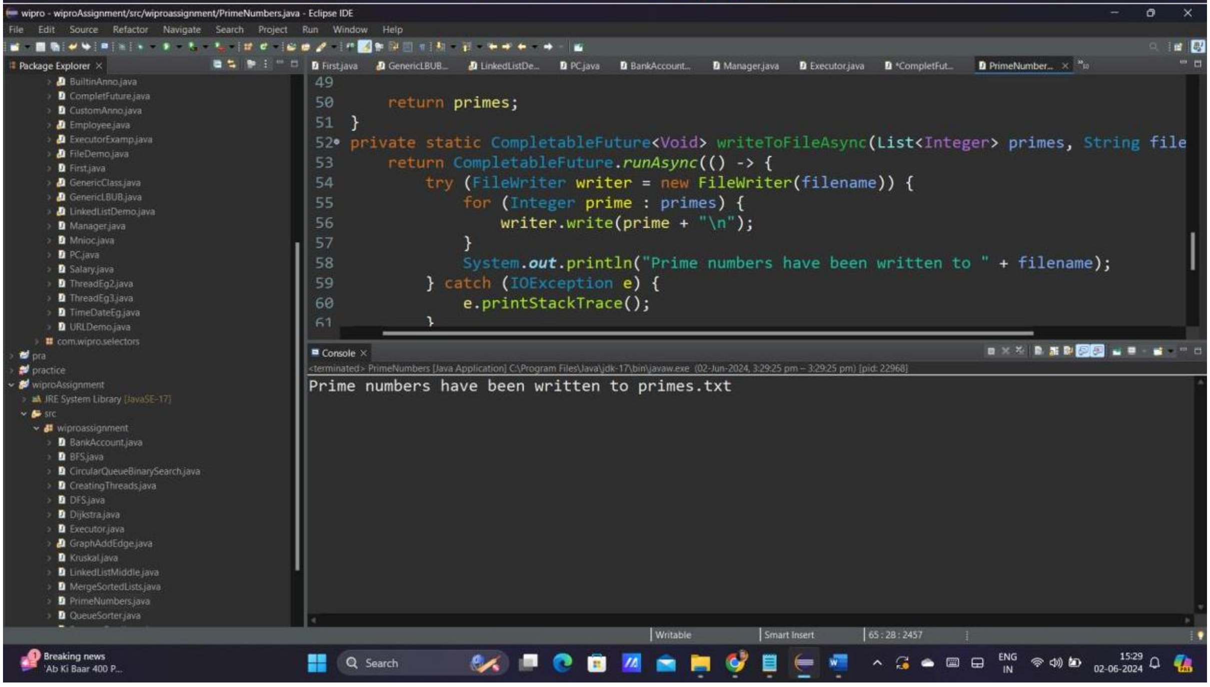
```java
package wiproassignment;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class PrimeNumbers {
    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;
    }
    private static List<Integer> calculatePrimes(int maxNumber, int numThreads) throws In
        List<Integer> primes = new ArrayList<>();
        ExecutorService executor = Executors.newFixedThreadPool(numThreads);
        List<CompletableFuture<Void>> futures = new ArrayList<>();
```


wipro - wiproAssignment/src/wiproassignment/PrimeNumbers.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Writable
Smart Insert
65 : 28 : 2457

```java
    ExecutorService executor = Executors.newFixedThreadPool(numThreads);
    List<CompletableFuture<Void>> futures = new ArrayList<>();
    for (int i = 2; i <= maxNumber; i++) {
        final int num = i;
        CompletableFuture<Void> future = CompletableFuture.runAsync(() -> {
            if (isPrime(num)) {
                synchronized (primes) {
                    primes.add(num);
                }
            }
        }, executor);
        futures.add(future);
    }
    CompletableFuture<Void> allFutures = CompletableFuture.allOf(futures.toArray(new Comp
    try {
       allFutures.get();
   } catch (InterruptedException e) {
       e.printStackTrace();
   } catch (ExecutionException e) {
       e.printStackTrace();
   }
    executor.shutdown();
    executor.awaitTermination(1, TimeUnit.MINUTES);

    return primes;
}
```

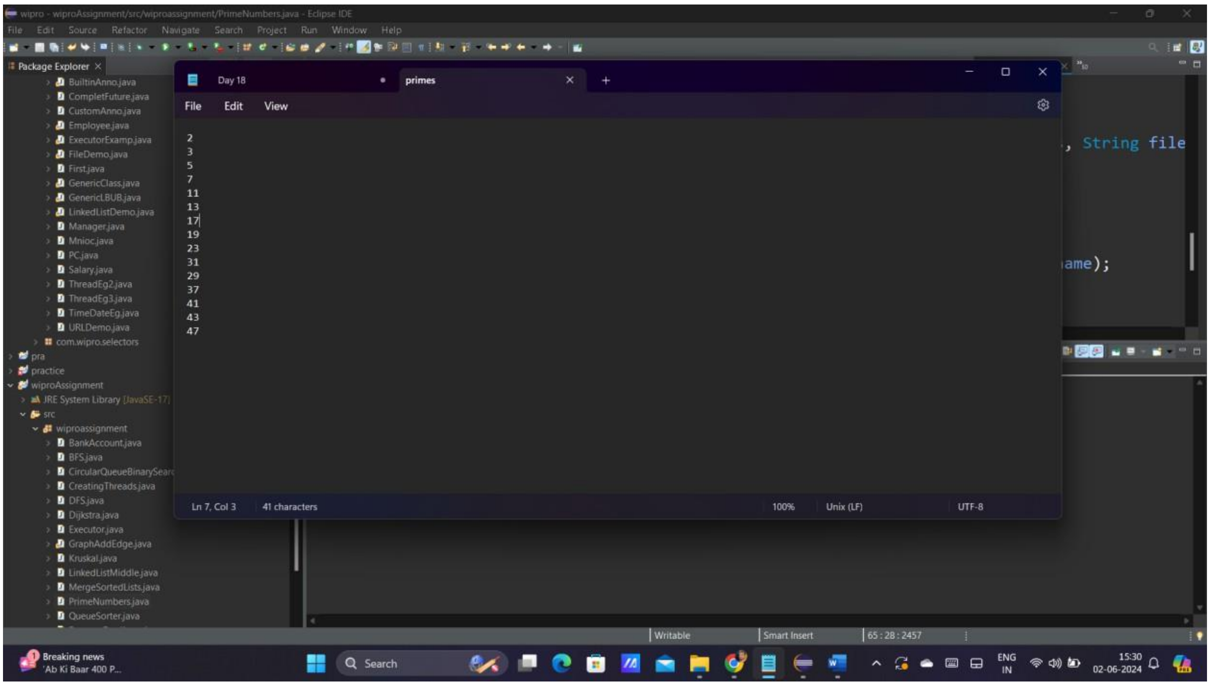


```java

        return primes;
    }
    private static CompletableFuture<Void> writeToFileAsync(List<Integer> primes, String file
        return CompletableFuture.runAsync(() -> {
            try (FileWriter writer = new FileWriter(filename)) {
                for (Integer prime : primes) {
                    writer.write(prime + "\n");
                }
                System.out.println("Prime numbers have been written to " + filename);
            } catch (IOException e) {
                e.printStackTrace();
            }
        });
    }
        public static void main(String[] args) {
        int maxNumber = 50;
        int numThreads = Runtime.getRuntime().availableProcessors();
        try {
            List<Integer> primes = calculatePrimes(maxNumber, numThreads);
            CompletableFuture<Void> fileWriteFuture = writeToFileAsync(primes, "primes.txt");
            fileWriteFuture.get();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```


wipro - wiproAssignment/src/wiproassignment/PrimeNumbers.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
First.java GenericLBUB... LinkedListDe... PC.java BankAccount... Manager.java Executor.java *CompletFut... PrimeNumber...
Console
Writable
Smart Insert
65 : 28 : 2457
Breaking news
'Ab Ki Baar 400 P...
Search
ENG
IN
15:29
02-06-2024

```java

        return primes;
    }
    private static CompletableFuture<Void> writeToFileAsync(List<Integer> primes, String file
        return CompletableFuture.runAsync(() -> {
            try (FileWriter writer = new FileWriter(filename)) {
                for (Integer prime : primes) {
                    writer.write(prime + "\n");
                }
                System.out.println("Prime numbers have been written to " + filename);
            } catch (IOException e) {
                e.printStackTrace();
```

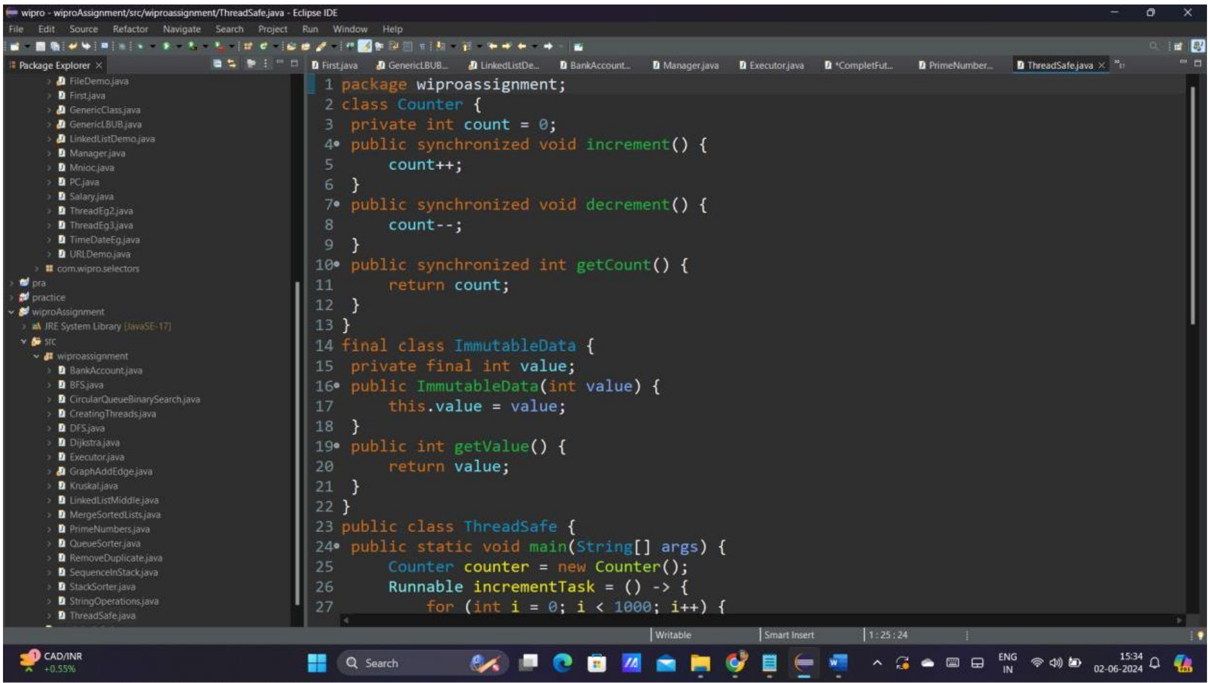
```
<terminated> PrimeNumbers [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-Jun-2024, 3:29:25 pm – 3:29:25 pm) [pid: 22968]
Prime numbers have been written to primes.txt
```


wipro - wiproAssignment/src/wiproassignment/PrimeNumbers.java - Eclipse IDE
Day 18
primes
File
Edit
View
Ln 7, Col 3
41 characters
100%
Unix (LF)
UTF-8

```
2
3
5
7
11
13
17
19
23
31
29
37
41
43
47
```

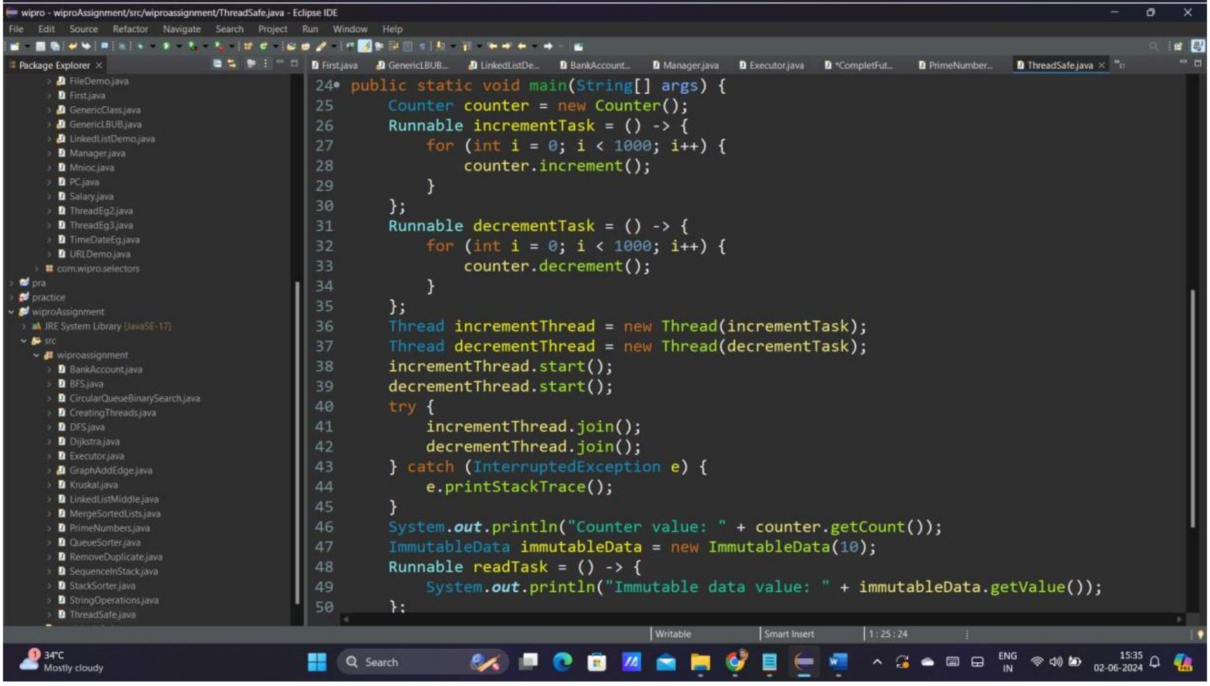
Task 7: Writing Thread-Safe Code, Immutable Objects

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.



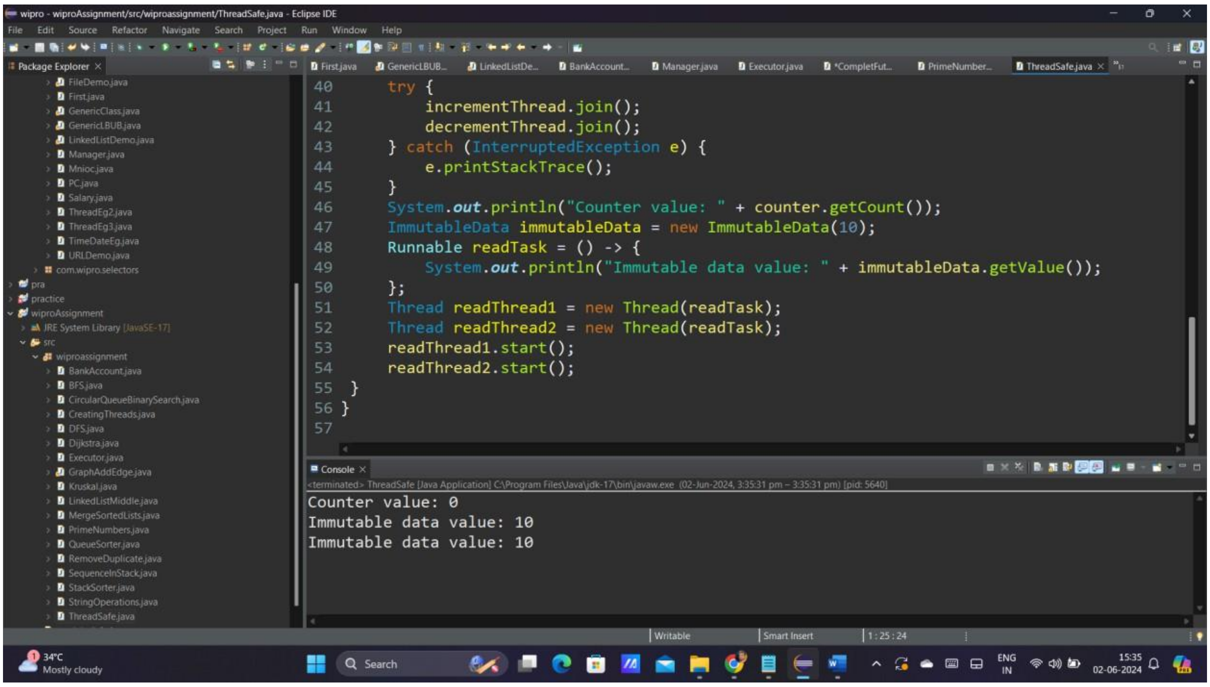
wipro - wiproAssignment/src/wiproassignment/ThreadSafe.java - Eclipse IDE
Package Explorer
ThreadSafe.java
Writable
Smart Insert
1 : 25 : 24

```java
package wiproassignment;
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;
 }
}
public class ThreadSafe {
 public static void main(String[] args) {
     Counter counter = new Counter();
     Runnable incrementTask = () -> {
         for (int i = 0; i < 1000; i++) {
```



wipro - wiproAssignment/src/wiproassignment/ThreadSafe.java - Eclipse IDE
Package Explorer
ThreadSafe.java
Writable
Smart Insert
1 : 25 : 24

```java
 public static void main(String[] args) {
     Counter counter = new Counter();
     Runnable incrementTask = () -> {
         for (int i = 0; i < 1000; i++) {
             counter.increment();
         }
     };
     Runnable decrementTask = () -> {
         for (int i = 0; i < 1000; i++) {
             counter.decrement();
         }
     };
     Thread incrementThread = new Thread(incrementTask);
     Thread decrementThread = new Thread(decrementTask);
     incrementThread.start();
     decrementThread.start();
     try {
         incrementThread.join();
         decrementThread.join();
     } catch (InterruptedException e) {
         e.printStackTrace();
     }
     System.out.println("Counter value: " + counter.getCount());
     ImmutableData immutableData = new ImmutableData(10);
     Runnable readTask = () -> {
         System.out.println("Immutable data value: " + immutableData.getValue());
     };
```



```java
        try {
            incrementThread.join();
            decrementThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Counter value: " + counter.getCount());
        ImmutableData immutableData = new ImmutableData(10);
        Runnable readTask = () -> {
            System.out.println("Immutable data value: " + immutableData.getValue());
        };
        Thread readThread1 = new Thread(readTask);
        Thread readThread2 = new Thread(readTask);
        readThread1.start();
        readThread2.start();
    }
}
```

<terminated> ThreadSafe [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-Jun-2024, 3:35:31 pm – 3:35:31 pm) [pid: 5640]

```
Counter value: 0
Immutable data value: 10
Immutable data value: 10
```