Name:Amit Kumar Parhi

Email:amitkparhi07@gmail.com

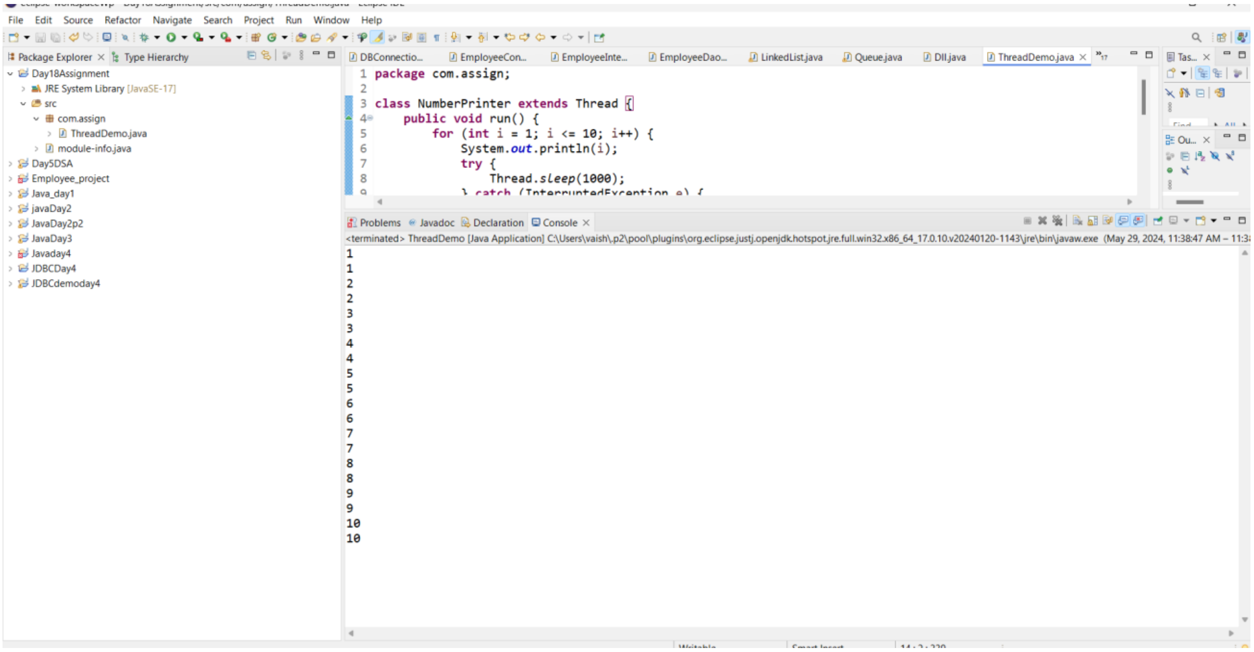
Day 18:

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

Solution:

```java
package com.assign;

class NumberPrinter 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 ThreadDemo {
    public static void main(String[] args) {
        Thread thread1 = new NumberPrinter();
        Thread thread2 = new NumberPrinter();

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

Output:



```java
package com.assign;

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

```
<terminated> ThreadDemo [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (May 29, 2024, 11:38:47 AM – 11:3
1
1
2
2
3
3
4
4
5
5
6
6
7
7
8
8
9
9
10
10
```

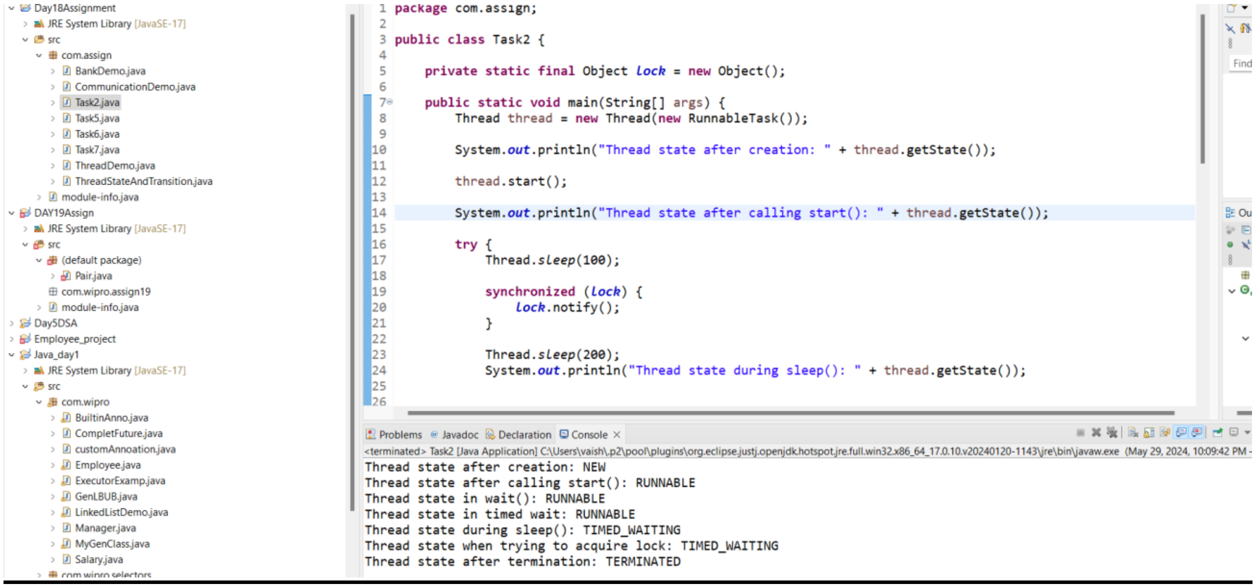
Task 2: States and Transitions Create a Java class that simulates a thread going through different lifecycle states: NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCKED, and TERMINATED. Use methods like sleep(), wait(), notify(), and join() to demonstrate these states..

Solution:

```java
package com.assign;

public class Task2 {

    private static final Object lock = new Object();

    public static void main(String[] args) {
        Thread thread = new Thread(new RunnableTask());

        System.out.println("Thread state after creation: " +
thread.getState());

        thread.start();

        System.out.println("Thread state after calling start(): " +
thread.getState());

        try {
            Thread.sleep(100);

            synchronized (lock) {
                lock.notify();
            }

            Thread.sleep(200);
            System.out.println("Thread state during sleep(): " +
thread.getState());


            synchronized (lock) {
```

```
        System.out.println("Thread state when trying to
acquire lock: " + thread.getState());
      }


      thread.join();
     System.out.println("Thread state after termination: " +
thread.getState());

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

  private static class RunnableTask implements Runnable {

    @Override
    public void run() {
      synchronized (lock) {
        try {

          lock.wait();
          System.out.println("Thread state in wait(): " +
Thread.currentThread().getState());


          Thread.sleep(100);
          System.out.println("Thread state in timed wait: "
+ Thread.currentThread().getState());
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }

      try {
        Thread.sleep(100);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Output:



```java
package com.assign;

public class Task2 {

    private static final Object lock = new Object();

    public static void main(String[] args) {
        Thread thread = new Thread(new RunnableTask());

        System.out.println("Thread state after creation: " + thread.getState());

        thread.start();

        System.out.println("Thread state after calling start(): " + thread.getState());

        try {
            Thread.sleep(100);

            synchronized (lock) {
                lock.notify();
            }

            Thread.sleep(200);
            System.out.println("Thread state during sleep(): " + thread.getState());

```

```
<terminated> Task2 [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe  (May 29, 2024, 10:09:42 PM -
Thread state after creation: NEW
Thread state after calling start(): RUNNABLE
Thread state in wait(): RUNNABLE
Thread state in timed wait: RUNNABLE
Thread state during sleep(): TIMED_WAITING
Thread state when trying to acquire lock: TIMED_WAITING
Thread state after termination: TERMINATED
```

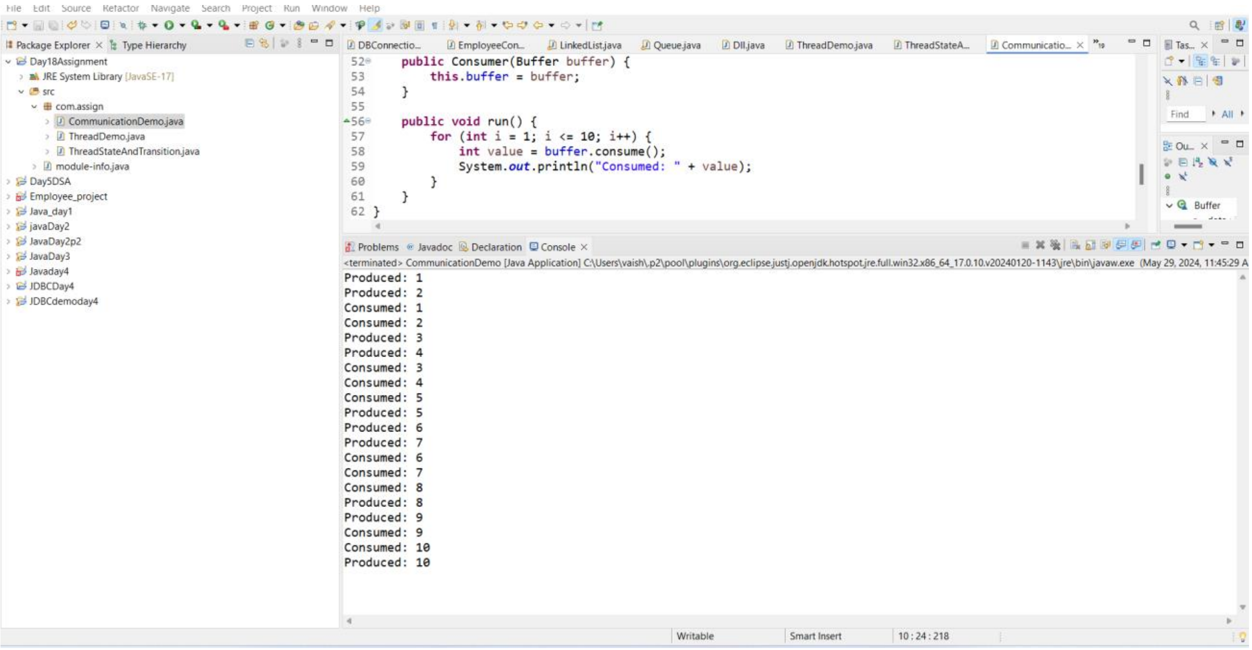
Task 3: Synchronization and Inter-thread Communication Implement a producer-consumer problem using wait() and notify() methods to handle the correct processing sequence between threads.

Solution:

```java
package com.assign;

class Buffer {
    private int data;
    private boolean empty = true;

    public synchronized void produce(int value) {
        while (!empty) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        data = value;
        empty = false;
        notify();
    }

    public synchronized int consume() {
        while (empty) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        empty = true;
        notify();
        return data;
    }
}
```

```
class Producer extends Thread {
    private Buffer buffer;

    public Producer(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            buffer.produce(i);
            System.out.println("Produced: " + i);
        }
    }
}

class Consumer extends Thread {
    private Buffer buffer;

    public Consumer(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            int value = buffer.consume();
            System.out.println("Consumed: " + value);
        }
    }
}

public class CommunicationDemo {
    public static void main(String[] args) {
        Buffer buffer = new Buffer();
        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);

        producer.start();
        consumer.start();
    }
}
```

Output:



```java
    public Consumer(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            int value = buffer.consume();
            System.out.println("Consumed: " + value);
        }
    }
}
```

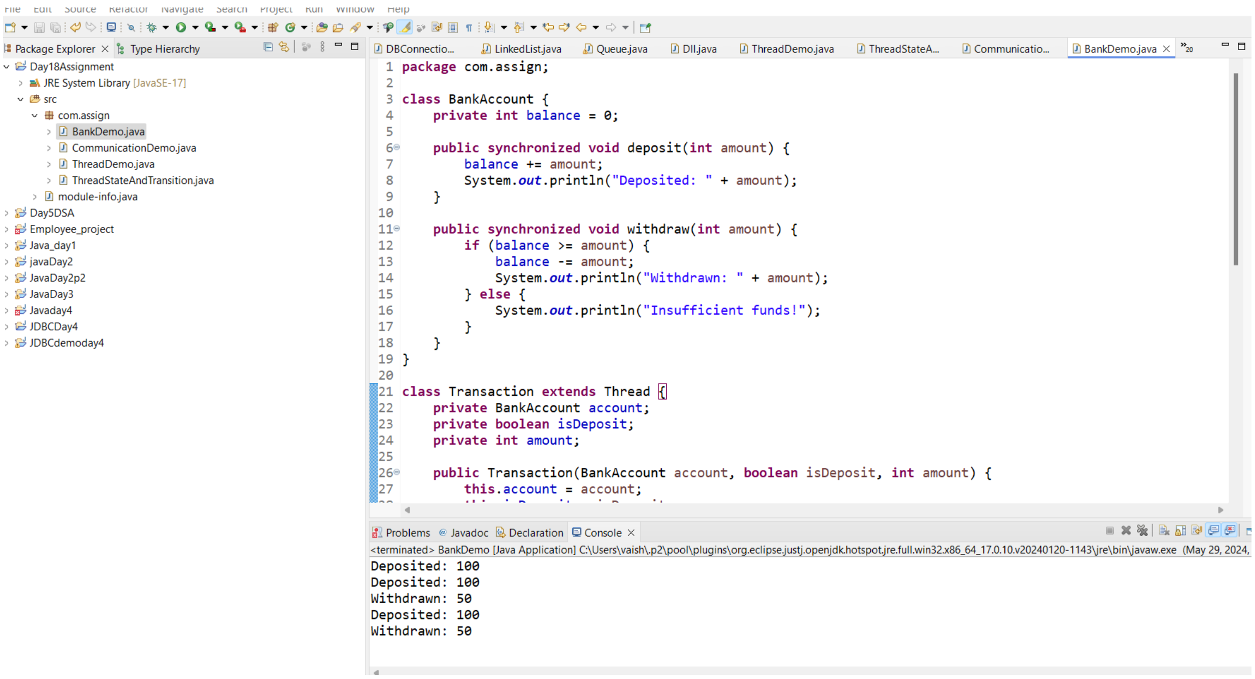
```
<terminated> CommunicationDemo [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (May 29, 2024, 11:45:29 A
Produced: 1
Produced: 2
Consumed: 1
Consumed: 2
Produced: 3
Produced: 4
Consumed: 3
Consumed: 4
Consumed: 5
Produced: 5
Produced: 6
Produced: 7
Consumed: 6
Consumed: 7
Consumed: 8
Produced: 8
Produced: 9
Consumed: 9
Consumed: 10
Produced: 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.

Solution:

```
package com.assign;

class BankAccount {
    private int balance = 0;

    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println("Deposited: " + amount);
    }

    public synchronized void withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println("Insufficient funds!");
        }
    }
}

class Transaction extends Thread {
    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 {
            account.withdraw(amount);
        }
    }
```

```java
}

public class BankDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();
        Transaction[] transactions = new Transaction[5];

        for (int i = 0; i < transactions.length; i++) {
            if (i % 2 == 0) {
                transactions[i] = new Transaction(account, true, 100);
            } else {
                transactions[i] = new Transaction(account, false, 50);
            }
        }

        for (Transaction transaction : transactions) {
            transaction.start();
        }
    }
}
```



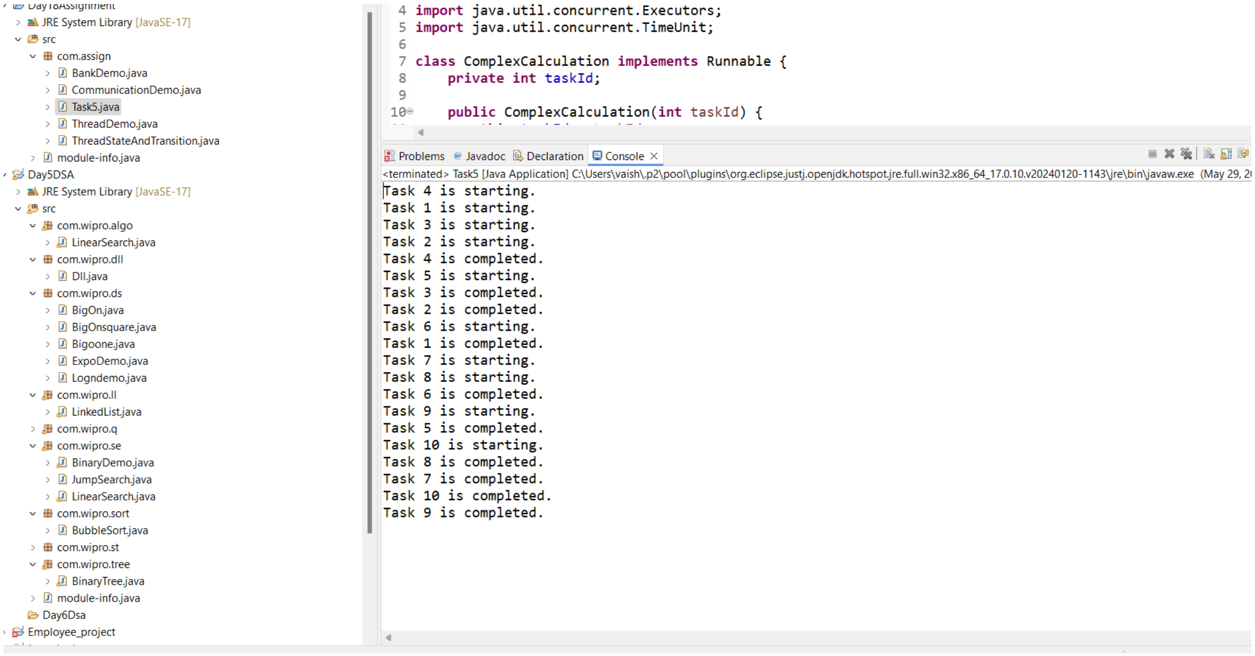
```java
package com.assign;

class BankAccount {
    private int balance = 0;

    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println("Deposited: " + amount);
    }

    public synchronized void withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println("Insufficient funds!");
        }
    }
}

class Transaction extends Thread {
    private BankAccount account;
    private boolean isDeposit;
    private int amount;

    public Transaction(BankAccount account, boolean isDeposit, int amount) {
        this.account = account;
```

```
<terminated> BankDemo [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (May 29, 2024,
Deposited: 100
Deposited: 100
Withdrawn: 50
Deposited: 100
Withdrawn: 50
```

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.

Solution:

```java
package com.assign;

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

class ComplexCalculation implements Runnable {
    private int taskId;

    public ComplexCalculation(int taskId) {
        this.taskId = taskId;
    }

    @Override
    public void run() {
        System.out.println("Task " + taskId + " is starting.");

        try {
            TimeUnit.SECONDS.sleep(2);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        System.out.println("Task " + taskId + " is completed.");
    }
}

public class Task5 {
    public static void main(String[] args) {
        ExecutorService executorService =
Executors.newFixedThreadPool(4);

        for (int i = 1; i <= 10; i++) {
            executorService.submit(new ComplexCalculation(i));
        }

        executorService.shutdown();
        try {
            if (!executorService.awaitTermination(60,
TimeUnit.SECONDS)) {
                executorService.shutdownNow();
            }
        } catch (InterruptedException e) {
            executorService.shutdownNow();
        }
    }
}
```



```
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

class ComplexCalculation implements Runnable {
    private int taskId;

    public ComplexCalculation(int taskId) {
```

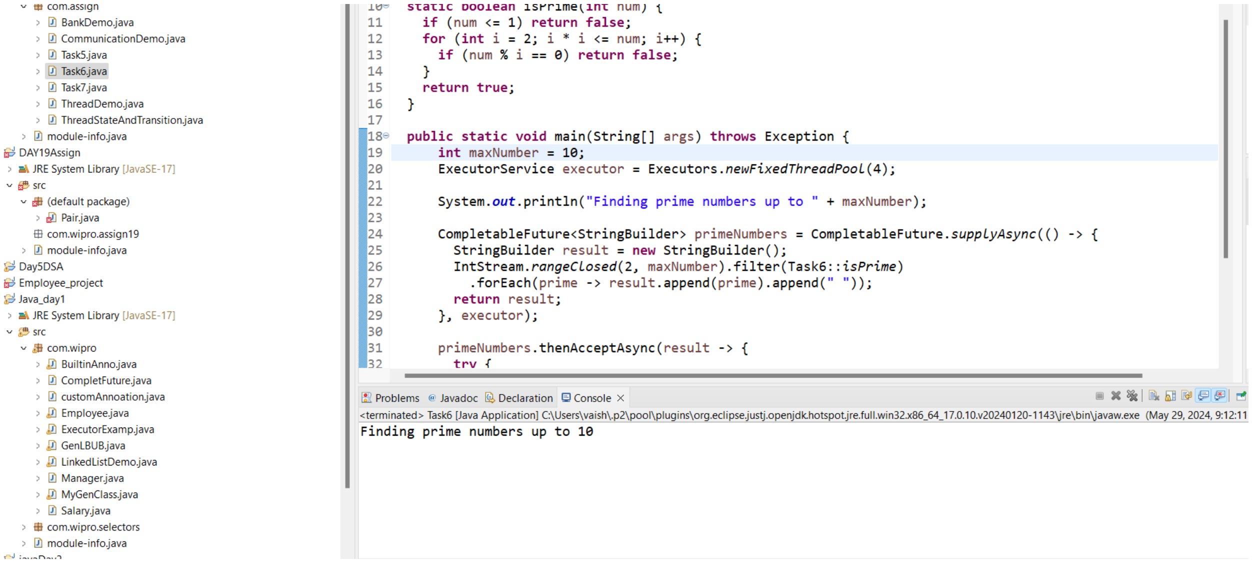
```
<terminated> Task5 [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe  (May 29, 2
Task 4 is starting.
Task 1 is starting.
Task 3 is starting.
Task 2 is starting.
Task 4 is completed.
Task 5 is starting.
Task 3 is completed.
Task 2 is completed.
Task 6 is starting.
Task 1 is completed.
Task 7 is starting.
Task 8 is starting.
Task 6 is completed.
Task 9 is starting.
Task 5 is completed.
Task 10 is starting.
Task 8 is completed.
Task 7 is completed.
Task 10 is completed.
Task 9 is 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.

Solution:

```
package com.assign;

import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.stream.IntStream;

public class Task6 {

  static boolean isPrime(int num) {
    if (num <= 1) return false;
    for (int i = 2; i * i <= num; i++) {
      if (num % i == 0) return false;
    }
    return true;
  }

  public static void main(String[] args) throws Exception {
       int maxNumber = 10;
       ExecutorService executor = Executors.newFixedThreadPool(4);

       System.out.println("Finding prime numbers up to " + maxNumber);

       CompletableFuture<StringBuilder> primeNumbers =
CompletableFuture.supplyAsync(() -> {
         StringBuilder result = new StringBuilder();
         IntStream.rangeClosed(2, maxNumber).filter(Task6::isPrime)
           .forEach(prime -> result.append(prime).append(" "));
         return result;
       }, executor);

       primeNumbers.thenAcceptAsync(result -> {
         try {

java.nio.file.Files.writeString(java.nio.file.Paths.get("d:/data/prime
_numbers.txt"), result.toString());
           System.out.println("Prime numbers written to file:
prime_numbers.txt");
```

```
        } catch (Exception e) {
          e.printStackTrace();
        }
      }, executor);

      executor.shutdown();
    }
}
```



```
  static boolean isPrime(int num) {
    if (num <= 1) return false;
    for (int i = 2; i * i <= num; i++) {
      if (num % i == 0) return false;
    }
    return true;
  }

  public static void main(String[] args) throws Exception {
      int maxNumber = 10;
      ExecutorService executor = Executors.newFixedThreadPool(4);

      System.out.println("Finding prime numbers up to " + maxNumber);

      CompletableFuture<StringBuilder> primeNumbers = CompletableFuture.supplyAsync(() -> {
        StringBuilder result = new StringBuilder();
        IntStream.rangeClosed(2, maxNumber).filter(Task6::isPrime)
          .forEach(prime -> result.append(prime).append(" "));
        return result;
      }, executor);

      primeNumbers.thenAcceptAsync(result -> {
        try {
```

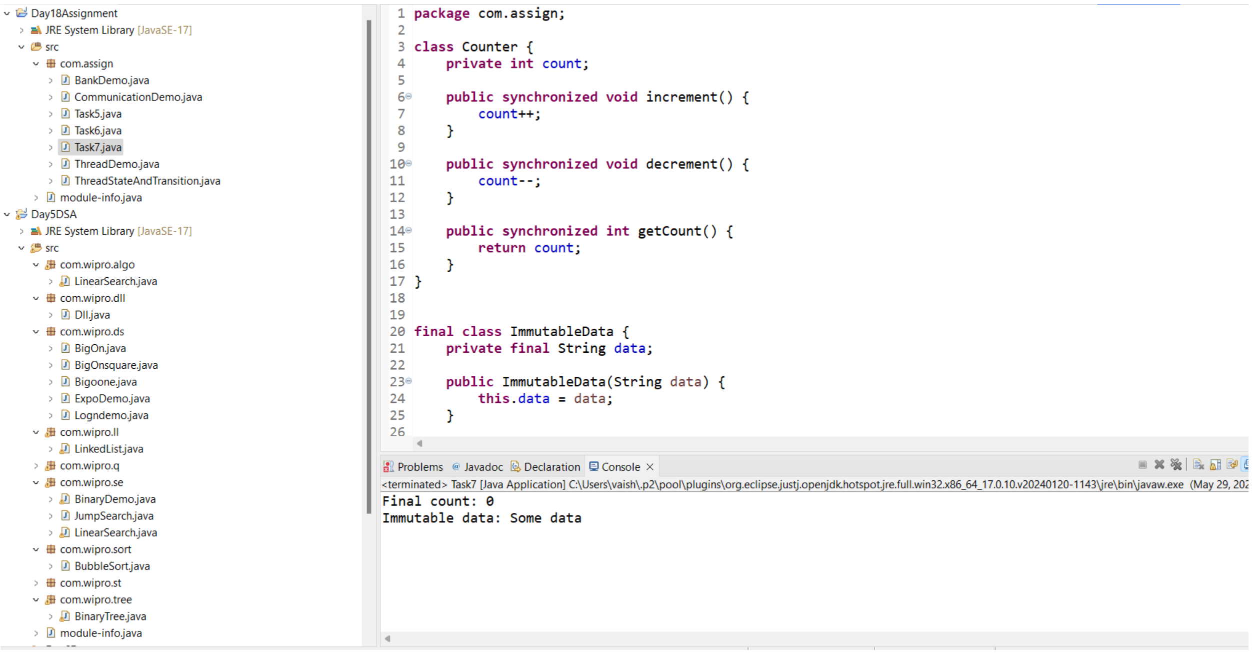
```
<terminated> Task6 [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe  (May 29, 2024, 9:12:11
Finding prime numbers up to 10
```

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.

Solution:

```
package com.assign;

class Counter {
    private int count;

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

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

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


final class ImmutableData {
    private final String data;

    public ImmutableData(String data) {
        this.data = data;
    }

    public String getData() {
        return data;
    }
}

public class Task7 {
    public static void main(String[] args) {
        Counter counter = new Counter();

        Runnable incrementTask = () -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        };
```

```java
        Runnable decrementTask = () -> {
            for (int i = 0; i < 1000; i++) {
                counter.decrement();
            }
        };

        Thread thread1 = new Thread(incrementTask);
        Thread thread2 = new Thread(decrementTask);

        thread1.start();
        thread2.start();

        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        System.out.println("Final count: " + counter.getCount());

        // Immutable object usage
        ImmutableData immutableData = new ImmutableData("Some data");
        System.out.println("Immutable data: " +
immutableData.getData());
    }
}
```



```java
package com.assign;

class Counter {
    private int count;

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

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

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


final class ImmutableData {
    private final String data;

    public ImmutableData(String data) {
        this.data = data;
    }

```

```
<terminated> Task7 [Java Application] C:\Users\vaish\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe  (May 29, 202
Final count: 0
Immutable data: Some data
```