Digital Assignment 4

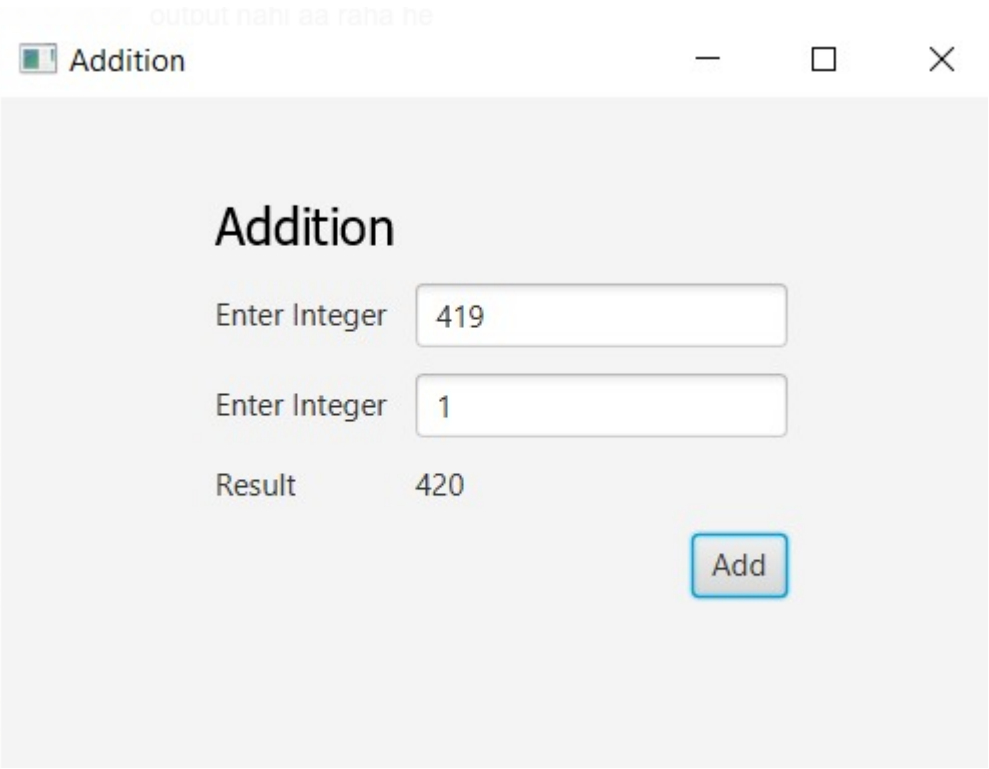
- Submitted by: Yash Kumar Verma
- Registration Number: 19BCE2669
- Code Available on : yashkumarverma-bot/semester3
- Report prepared on : Markdown

Question 1

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class Addition extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Addition");
        GridPane grid = new GridPane();
        grid.setAlignment(Pos.CENTER);
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(25, 25, 25, 25));

        Text scenetitle = new Text("Addition");
        scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
        grid.add(scenetitle, 0, 0, 2, 1);

        Label int1 = new Label("Enter Integer");
        grid.add(int1, 0, 1);

        TextField intText1 = new TextField();
```

```
        grid.add(intText1, 1, 1);

        Label int2 = new Label("Enter Integer");
        grid.add(int2, 0, 2);

        TextField intText2 = new TextField();
        grid.add(intText2, 1, 2);

        Label result = new Label("Result");
        grid.add(result, 0, 3);
        Label result1 = new Label("0");
        grid.add(result1, 1, 3);

        Button btn = new Button("Add");
        HBox hbBtn = new HBox(10);
        hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
        hbBtn.getChildren().add(btn);
        grid.add(hbBtn, 1, 4);

        final Text actiontarget = new Text();
        grid.add(actiontarget, 1, 6);

        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent e) {
                result1.setText("" + (Integer.parseInt(intText1.getText())
+ Integer.parseInt(intText2.getText())));
            }
        });

        Scene scene = new Scene(grid, 400, 275);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Output



Question 2

A cook in VIT canteen prepares parotta and stacks it up in a container, and the server takes parotta from the container and serves to his customer. The max capacity of the container is 15. If parotta in the container is empty, server waits for the cook to prepare new parotta. Write a Java program to illustrate the given scenario using multithreading.

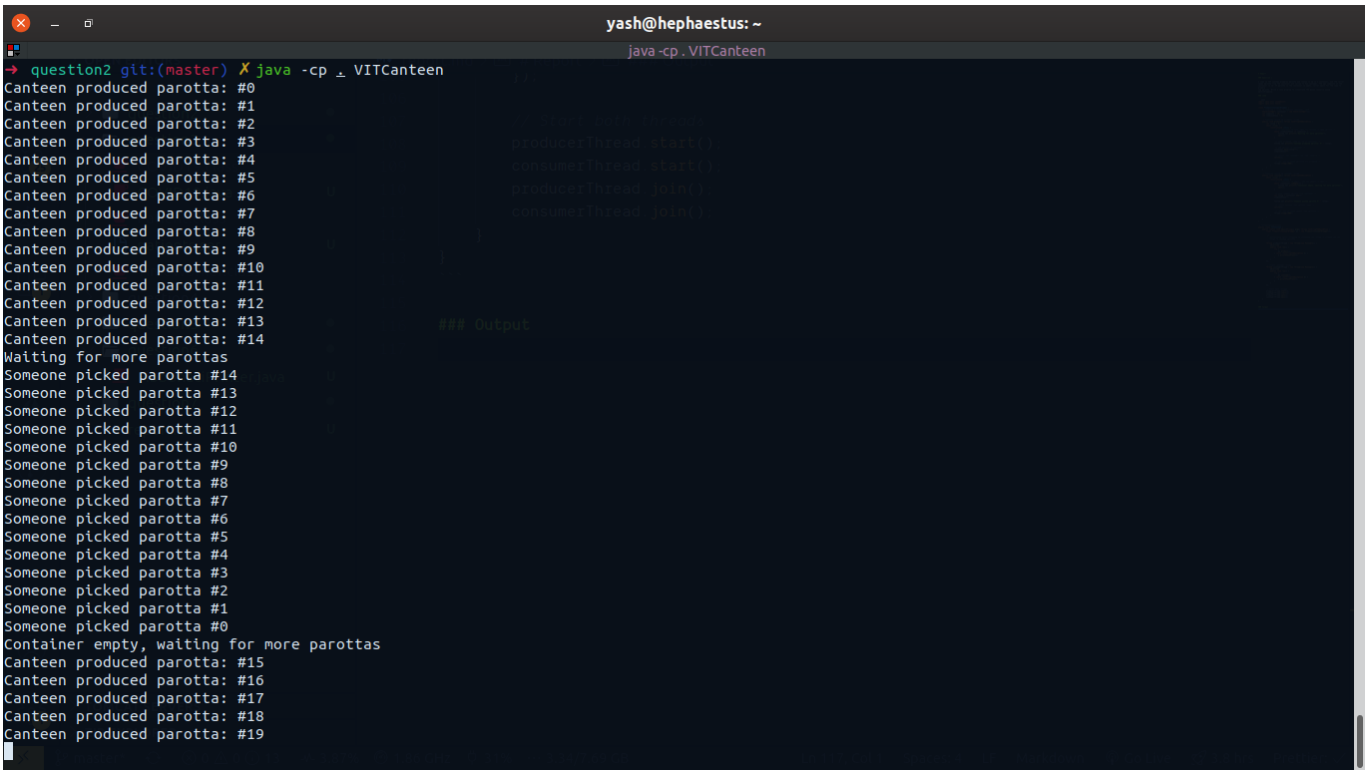
Code

```
import java.util.LinkedList;
import java.util.Stack;

class ProducerConsumerWrapper {
    // creating a stack to simulate container, LIFO
    Stack<Integer> container = new Stack<Integer>();
    int capacity = 15;
    int itemsInStack = 0;

    // Function called by producer thread
    public void produce() throws InterruptedException {
        int value = 0;
        while (true) {
            synchronized (this) {
                /** if container is full, wait for kitchen to make parotta
*/
                while (itemsInStack == capacity) {
                    System.out.println("Waiting for more parottas");
                    wait();
                }

                /** when produced, keep at top of container */
```

```
                System.out.println("Canteen produced parotta: #" + value);

                /** add parotta to container */
                container.push(value++);
                itemsInStack++;

                /** notify that consumers can take resource */
                notify();

                /** set rate at which parottas are produced */
                Thread.sleep(1000);
            }
        }
    }

    // Function called by consumer thread
    public void consume() throws InterruptedException {
        while (true) {
            synchronized (this) {
                /** if container is empty, then ask people to wait */
                while (container.isEmpty()) {
                    System.out.println("Container empty, waiting for more
parottas");
                    wait();
                }

                /** pick topmost parotta from container */
                int value = container.pop();
                itemsInStack--;

                System.out.println("Someone picked parotta #" + value);

                /** notify kitchen that parotta taken */
                notify();

                /** set rate at which people take parottas */
                Thread.sleep(1000);
            }
        }
    }
}

// VITCanteenOperations
public class VITCanteen {
    public static void main(String[] args) throws InterruptedException {
        final ProducerConsumerWrapper pc = new ProducerConsumerWrapper();

        /**
         * initializing multi-threading by attaching one thread to produce
and one
         * threat to consume parottas
         */
        Thread producerThread = new Thread(new Runnable() {
            @Override
```

```
                public void run() {
                    try {
                        pc.produce();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });

            // Create consumer thread
            Thread consumerThread = new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
                        pc.consume();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });

            // Start both threads
            producerThread.start();
            consumerThread.start();
            producerThread.join();
            consumerThread.join();
        }
    }
```

Output



Question3

Write a Java program to define a class 'Covid19' to store the below mentioned details of a Covid patients for CMC hospital. Name, age, address, mobile number, blood group, date of Covid checking. symptoms. Create 'n' objects of this class for all the Covid patients at Vellore. Write these objects to a file. Read these objects from the file and display only those Covid patient details whose symptoms is 'fever' and seven days completed from the date of Covid checking.

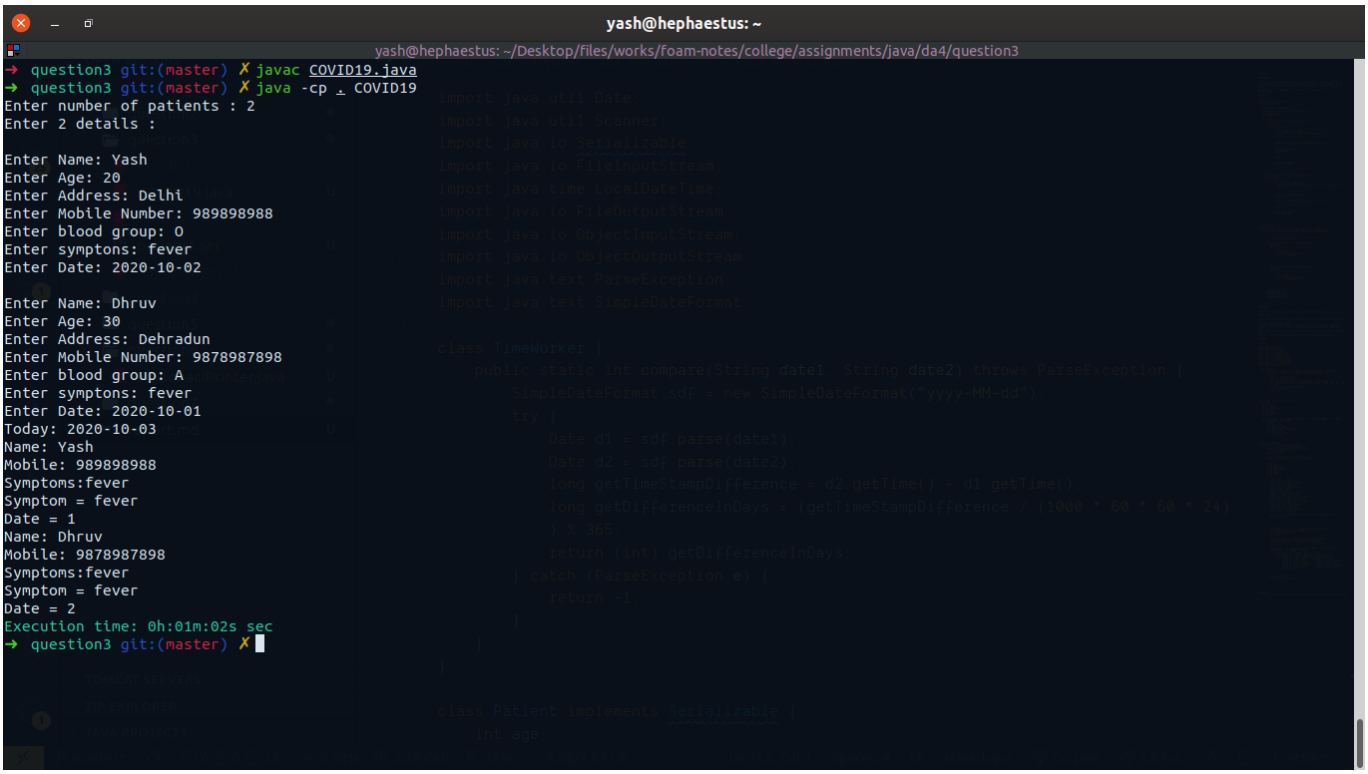
Code

```
import java.util.Date;
import java.util.Scanner;
import java.io.Serializable;
import java.io.FileInputStream;
import java.time.LocalDateTime;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.text.ParseException;
import java.text.SimpleDateFormat;

class TimeWorker {
    public static int compare(String date1, String date2) throws
ParseException {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date d1 = sdf.parse(date1);
            Date d2 = sdf.parse(date2);
            long getTimeStampDifference = d2.getTime() - d1.getTime();
            long getDifferenceInDays = (getTimeStampDifference / (1000 * 60
* 60 * 24)) % 365;
            return (int) getDifferenceInDays;
        } catch (ParseException e) {
            return -1;
        }
    }
}

class Patient implements Serializable {
    int age;
    String name;
    String address;
    String mobileNumber;
    String blood;
    String symptons;
    String date;

    Patient(int age, String name, String address, String mobileNumber,
String blood, String symptons, String date) {
        this.age = age;
        this.name = name;
```

```
        this.address = address;
        this.mobileNumber = mobileNumber;
        this.blood = blood;
        this.symptons = symptons;
        this.date = date;
    }
}

public class COVID19 {
    public static void main(String args[]) {
        Scanner handler = new Scanner(System.in);
        System.out.print("Enter number of patients : ");
        int items = handler.nextInt();
        handler.nextLine();

        Patient patients[] = new Patient[items];
        System.out.println("Enter " + items + " details : ");

        int age;
        String name;
        String address;
        String mobileNumber;
        String blood;
        String symptons;
        String date;

        for (int i = 0; i < items; i++) {
            System.out.println();
            System.out.print("Enter Name: ");
            name = handler.nextLine();
            System.out.print("Enter Age: ");
            age = handler.nextInt();
            handler.nextLine();
            System.out.print("Enter Address: ");
            address = handler.nextLine();
            System.out.print("Enter Mobile Number: ");
            mobileNumber = handler.nextLine();
            System.out.print("Enter blood group: ");
            blood = handler.nextLine();
            System.out.print("Enter symptons: ");
            symptons = handler.nextLine();
            System.out.print("Enter Date: ");
            date = handler.nextLine();

            patients[i] = new Patient(age, name, address, mobileNumber,
blood, symptons, date);
        }

        try {

            /** writing */
            FileOutputStream fos = new FileOutputStream("patients.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(patients);
```

```
            oos.close();

            FileInputStream fis = new FileInputStream("patients.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            Patient[] savedPatient = (Patient[]) ois.readObject();
            ois.close();

            String today = new SimpleDateFormat("yyyy-MM-dd").format(new
Date());
            System.out.println("Today: " + today);
            for (int i = 0; i < savedPatient.length; i += 1) {
                if (savedPatient[i].symptons.equals(new String("fever"))) {
                    if (TimeWorker.compare(savedPatient[i].date, today) <=
7) {
                        System.out.println("Name: " +
savedPatient[i].name);
                        System.out.println("Mobile: " +
savedPatient[i].mobileNumber);
                        System.out.println("Symptoms:" +
savedPatient[i].symptons);
                        System.out.println("Symptom = " +
savedPatient[i].symptons);
                        System.out.println("Date = " +
TimeWorker.compare(savedPatient[i].date, today));
                    }
                }
            }
        } catch (Exception e) {
            System.out.print("Error");
        }
    }
}
```

Output



Question 4

Write a Java program to create a package named banking which has a class named Account and include account details. Perform series of transactions in the main method using the package.

Code

File **Account.java**

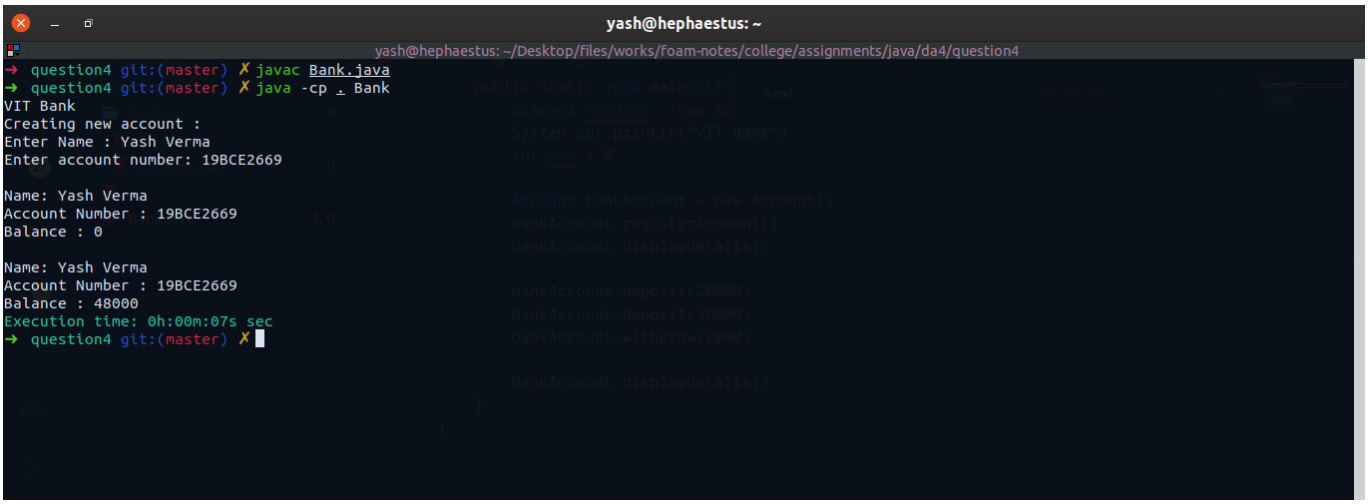
```
package banking;

import java.util.Scanner;

public class Account {
    public Account() {
        System.out.println("Creating new account : ");
    }

    private String name;
    private String accountNumber;
    private int balance;

    public void registerAccount() {
        Scanner handler = new Scanner(System.in);

        System.out.print("Enter Name : ");
        String name = handler.nextLine();

        System.out.print("Enter account number: ");
        String accountNumber = handler.nextLine();
```

```
        this.name = name;
        this.accountNumber = accountNumber;
        this.balance = 0;
        handler.close();
    }

    public void displayDetails() {
        System.out.println();
        System.out.println("Name: " + this.name);
        System.out.println("Account Number : " + this.accountNumber);
        System.out.println("Balance : " + this.balance);
    }

    public void deposit(int amt) {
        this.balance += amt;
    }

    public void withDraw(int amt) {
        this.balance -= amt;
    }
}
```

File **Bank.java**

```
import java.util.Scanner;
import banking.Account;

public class Bank {
    public static void main(String args[]) {
        Scanner handler = new Scanner(System.in);
        System.out.println("VIT Bank");
        int amt = 0;

        Account bankAccount = new Account();
        bankAccount.registerAccount();
        bankAccount.displayDetails();

        bankAccount.deposit(20000);
        bankAccount.deposit(30000);
        bankAccount.withDraw(2000);

        bankAccount.displayDetails();
    }
}
```

Output



Quesion 5

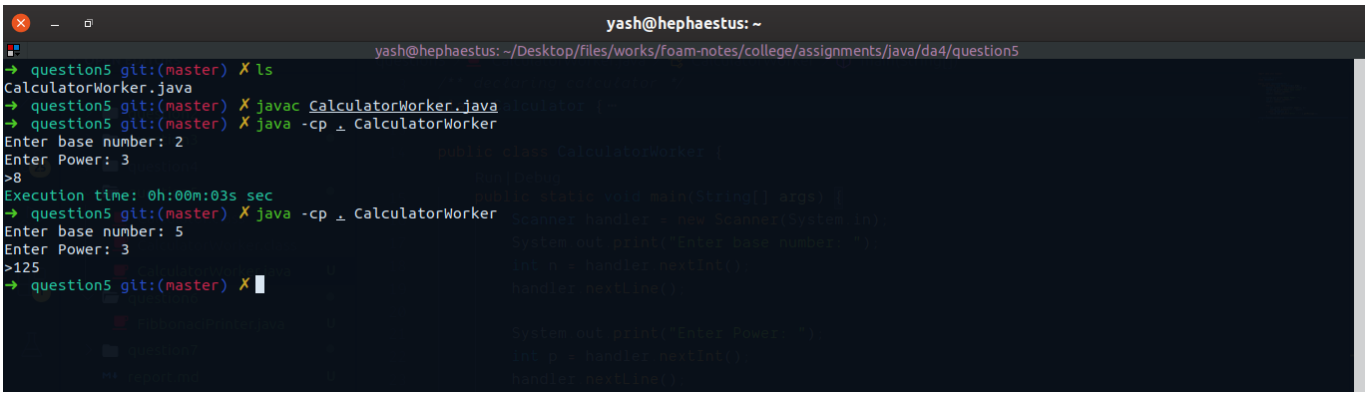
Write a Calculator class with a single method: int power(int n,int p). The power method takes two integers, n and p, as parameters and returns the integer result of np . If either n or p is negative, then the method must throw an exception with the message: n and p should be nonnegative.

Code

```
import java.util.Scanner;

/** declaring calculator */
class Calculator {
    public static int power(int base, int power) throws Exception {
        if (base >= 0 && power >= 0) {
            return (int) Math.pow(base, power);
        } else {
            throw new Exception("base and power should should be 0 or
positive");
        }
    }
}

public class CalculatorWorker {
    public static void main(String[] args) {
        Scanner handler = new Scanner(System.in);
        System.out.print("Enter base number: ");
        int n = handler.nextInt();
        handler.nextLine();

        System.out.print("Enter Power: ");
        int p = handler.nextInt();
        handler.nextLine();

        try {
            int result = Calculator.power(n, p);
            System.out.println(">" + result);
        } catch (Exception e) {
            System.out.println("Error : " + e.getMessage());
```

```
            }
            handler.close();
        }
    }
```

Output



Question 6

Write a Java program using threads to compute the first 25 prime numbers, and to compute the first 50 Fibonacci numbers. Set the priority of thread that computes Fibonacci number to 8 and the other to 5. After calculating 30 Fibonacci numbers, make that thread to sleep and take up prime number computation. After computing the 25 prime numbers continue the Fibonacci number computing.

Code

```
class Prime implements Runnable {
    long j, counter;

    Prime() {
        super();
        counter = 0;
    }

    public void run() {
        for (long i = 0; counter <= 25; i++) {
            for (j = 2; j <= i; j++) {
                if (i % j == 0)
                    break;
            }
            if (j == i) {
                counter++;
                System.out.println("Print number #" + counter + " is " +
i);
            }
        }
    }
}
```

```
class Fibbonaci implements Runnable {
    long a, b, c, n;

    Fibbonaci() {
        a = c = n = 0;
        b = 1;
    }

    public void run() {
        while (n++ < 50) {
            System.out.println(n + " th " + " Fibbonaci = " + a);
            c = a + b;
            a = b;
            b = c;
            try {
                if (n == 30) {
                    System.out.println("Wait thread.");
                    Thread.sleep(500);
                }

            } catch (InterruptedException e) {
                System.out.println("Error : " + e);
            }
        }
    }
}

public class FibbonaciPrinter {

    public static void main(String[] args) {
        Thread currentThread = Thread.currentThread();
        System.out.println("Main thread name : " +
currentThread.getName());

        Prime prime = new Prime();
        Fibbonaci fibbonaci = new Fibbonaci();
        Thread fibThread = new Thread(fibbonaci, "fibonacci");
        Thread primeThread = new Thread(prime, "prime");
        fibThread.start();
        System.out.println("Thread " + fibThread.getName() + " started.");
        primeThread.start();
        System.out.println("Thread " + primeThread.getName() + "
started.");
    }
}
```

Output

```
yash@hephaestus: ~
yash@hephaestus: ~/Desktop/files/works/foam-notes/college/assignments/java/da4/question6
→ question6 git:(master) ✗ javac FibbonaciPrinter.java
→ question6 git:(master) ✗ java FibbonaciPrinter
Main thread name : main
Thread fibonacci started.
Thread prime started.
1 th  Fibbonaci = 0
Print number #1 is 2
Print number #2 is 3
Print number #3 is 5
2 th  Fibbonaci = 1
3 th  Fibbonaci = 1
4 th  Fibbonaci = 2
5 th  Fibbonaci = 3
6 th  Fibbonaci = 5
7 th  Fibbonaci = 8
Print number #4 is 7
8 th  Fibbonaci = 13
Print number #5 is 11
9 th  Fibbonaci = 21
Print number #6 is 13
10 th  Fibbonaci = 34
Print number #7 is 17
11 th  Fibbonaci = 55
Print number #8 is 19
12 th  Fibbonaci = 89
Print number #9 is 23
13 th  Fibbonaci = 144
Print number #10 is 29
14 th  Fibbonaci = 233
Print number #11 is 31
15 th  Fibbonaci = 377
Print number #12 is 37
16 th  Fibbonaci = 610
Print number #13 is 41
17 th  Fibbonaci = 987
Print number #14 is 43
18 th  Fibbonaci = 1597
Print number #15 is 47
19 th  Fibbonaci = 2584
```

```
26 th  Fibbonaci = 75025
Print number #23 is 83
27 th  Fibbonaci = 121393
Print number #24 is 89
28 th  Fibbonaci = 196418
Print number #25 is 97
29 th  Fibbonaci = 317811
Print number #26 is 101
30 th  Fibbonaci = 514229
Wait thread.
31 th  Fibbonaci = 832040
32 th  Fibbonaci = 1346269
33 th  Fibbonaci = 2178309
34 th  Fibbonaci = 3524578
35 th  Fibbonaci = 5702887
36 th  Fibbonaci = 9227465
37 th  Fibbonaci = 14930352
38 th  Fibbonaci = 24157817
39 th  Fibbonaci = 39088169
40 th  Fibbonaci = 63245986
41 th  Fibbonaci = 102334155
42 th  Fibbonaci = 165580141
43 th  Fibbonaci = 267914296
44 th  Fibbonaci = 433494437
45 th  Fibbonaci = 701408733
46 th  Fibbonaci = 1134903170
47 th  Fibbonaci = 1836311903
48 th  Fibbonaci = 2971215073
49 th  Fibbonaci = 4807526976
50 th  Fibbonaci = 7778742049
→ question6 git:(master) ✗
```

Question 7

7.Write a Java program to create a class Person that implements serialization concept with name,age and anuual income of a person as its data members. Store the state of objects of this class in a file. Write another class that reads the objects of the Person class from the file. For each object of the class stored in the file, check the age of the person.

- If the age of a person exceeds 65, then categorize the person as very old.
- If the age of a person exceeds 45, then categorize the person as old.
- If the age of a person exceeds 25 but is less than 45, then categorize the person as very old.
- If the age of a person exceeds 65, then categorize the person as young.
- If the age of a person is less than 25, then categorize the person as very young.

Code

File **PersonReader.java**

```
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import Packages.Person;

/** person class implementing Serializable */

public class PersonReader {
    public static void main(String args[]) {
        System.out.println("Starting de-serialization: ");
        try {
            System.out.println("Reading objects from disk: ");

            FileInputStream fileInputStream = new
FileInputStream("object.ser");
            ObjectInputStream objectInputStream = new
ObjectInputStream(fileInputStream);
            Person[] persons = (Person[]) objectInputStream.readObject();

            // looping over all persons
            for (Person person : persons) {
                processAge(person);
            }

        } catch (FileNotFoundException e) {
            System.out.println("File Not found!");
        } catch (IOException e) {
            System.out.println("IO Exception!");
        } catch (ClassNotFoundException e) {
            System.out.println("Undefined Class called");
```

```
        }

    }

    static void processAge(Person p) {
        if (p.age > 65) {
            System.out.println(p.name + " is very old");
        } else if (p.age > 45) {
            System.out.println(p.name + " is old");
        } else if (p.age > 25) {
            System.out.println(p.name + " not young");
        } else {
            System.out.println(p.name + " is very young");
        }
    }
}
```

File **PersonWriter.java**

```
import java.io.Serializable;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.ObjectOutputStream;
import Packages.Person;

public class PersonWriter implements Serializable {
    public static void main(String args[]) {
        System.out.println("Starting Serialization");
        Person yash = new Person("Yash Kumar Verma", 20, 5000);
        Person dhruv = new Person("Dhruv Kumar Verma", 27, 10000);
        Person ravi = new Person("Ravi Kishan", 68, 100000);
        Person shyam = new Person("Shyam Mukherjee", 48, 60000);
        Person[] persons = { yash, dhruv, ravi, shyam };

        try {
            FileOutputStream fileOutputStream = new
FileOutputStream("object.ser");
            ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);

            objectOutputStream.writeObject(persons);
            System.out.println("Objects Written to disk");

            objectOutputStream.close();

        } catch (FileNotFoundException e) {
            System.out.println("File Not found!");
        } catch (IOException e) {
            System.out.println("IO Exception!");
        }
```

```
        }
    }
```

File **Person.java**

```java
package Packages;

import java.io.Serializable;

public class Person implements Serializable {
    public String name;
    public int age;
    public int income;

    public Person(String name, int age, int income) {
        this.age = age;
        this.income = income;
        this.name = name;
    }
}
```

Output