ECS2301 Software Engineering and Project

Final project (100 marks)

Instructions:

- This is a continuation of your mid-term exam project. You will complete the logic, input validation and output formatting here.
- If your SLTC student number ends with 0,1,2,3 then you must attempt question 1.
- If your SLTC student number ends with 4,5,6 then you must attempt question 2.
- If your SLTC student number ends with 7,8,9 then you must attempt question 3.
- Submit your answers as a single file (**.ZIP**) on or before the deadline provided in the LMS.
- Submission must include this document explaining your code, output screens and lessons learnt
- Late submission will not be considered for the marking.
- Make sure to include this document in your submission

Full name: MSM. Mohammed Musharraff

Student index: 22UG2-0064

Date of submission: 3/30/2024

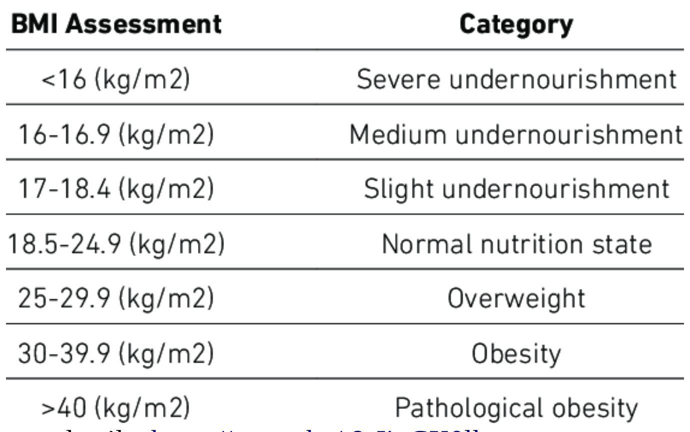
Q1. BMI calculator

BMI, or Body Mass Index, is a numerical value of a person's weight in relation to their height. It is a commonly used screening tool to categorize individuals into different weight status categories, such as underweight, normal weight, overweight, and obesity.

BMI is calculated using the following formula:

$$BMI = \frac{\text{weight in kilograms}}{(\text{height in meters})^2}$$

Here's a breakdown of the BMI categories:



BMI Assessment	**Category**
<16 (kg/m2)	Severe undernourishment
16-16.9 (kg/m2)	Medium undernourishment
17-18.4 (kg/m2)	Slight undernourishment
18.5-24.9 (kg/m2)	Normal nutrition state
25-29.9 (kg/m2)	Overweight
30-39.9 (kg/m2)	Obesity
>40 (kg/m2)	Pathological obesity

Watch this video for more details: https://youtu.be/t8sIioCX0lk

Requirements:

Tester
-id (int) -name (String) -yob (year of birth) -height (int) -weight (int)

+main()
+displayMenu()
+index()
+view(int id)
+create()
+delete()
+exit()

Tester class diagram

Bmi
-id (int) -name (String) -yob (year of birth) -height (int) -weight (int)

+bmi() (constructor)
+setters/getters for properties
+calculate()
+display()

Bmi class diagram

Write an application to store BMI of 5 users. It should be a menu driven application. You need to show the evidence of using classes, objects, methods, properties, setters/getters, constructors, abstraction, inheritance, polymorphism and java collections.

a) In your main class create a displayMenu() method and show the following choices. The program must continuously run until 'exit' is selected (10 marks) :

   i. Create a record. (Ask for user id, then ask data for name, year of birth, height & weight)

   ii. Show BMI data for all users.

   iii. Show BMI data for a selected user.

   iv. Delete all.

   v. Exit application.

b) Write methods for all actions above, inside main class (10 marks for each)

   i. index() : to show all records. Call Bmi.display() here.

   ii. view(int id) : to show one record for the given id. Include BMI category and recommendations if there are any. Call Bmi.display() here.

   iii. create() : create a new record with appropriate input validation. Call Bmi constructor here.

   iv. delete() : delete all records.

   v. exit() : exit to system

c) Use a suitable Java collection to store 5 users (10 marks)

d) Calculate BMI accurately, taking into consideration height in cm (5 marks)

e) Calculate the age with year of birth and current year (5 marks)

f) A section on the lessons learnt during this exercise (10 marks)

g) Generate JavaDoc files (5 marks)

h) Upload your complete, commented, tested code as a Git Hub repository to `/final` branch. Include the link in the document (5 marks)

Q2. Blood pressure monitor

Your blood pressure is recorded as two numbers:

- Systolic blood pressure (the first number) – indicates how much pressure your blood is exerting against your artery walls when the heart contracts.
- Diastolic blood pressure (the second number) – indicates how much pressure your blood is exerting against your artery walls while the heart muscle is resting between contractions.

Blood pressure categories

The five blood pressure ranges as recognized by the American Heart Association are:

1. Normal

Blood pressure numbers of less than 120/80 mm Hg (millimeters of mercury) are considered within the normal range.

2. Elevated

Elevated blood pressure is when readings consistently range from 120-129 systolic and less than 80 mm Hg diastolic.

3. Hypertension Stage 1

Hypertension Stage 1 is when blood pressure consistently ranges from 130 to 139 systolic or 80 to 89 mm Hg diastolic.

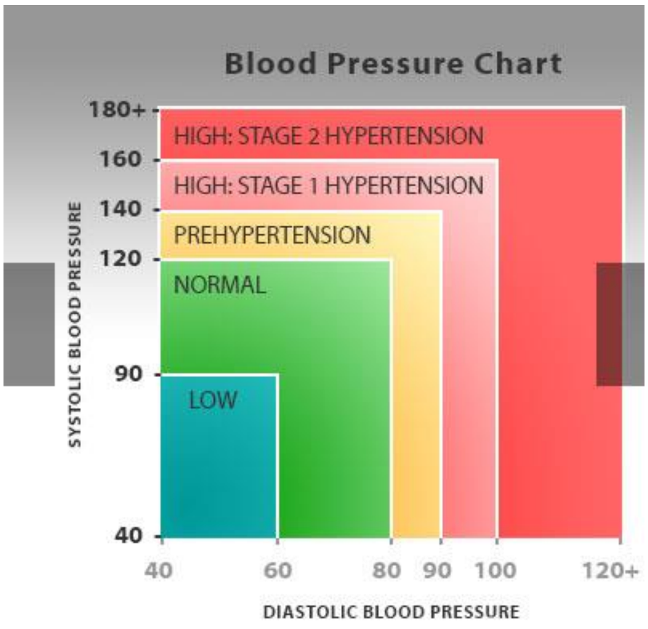
4. Hypertension Stage 2

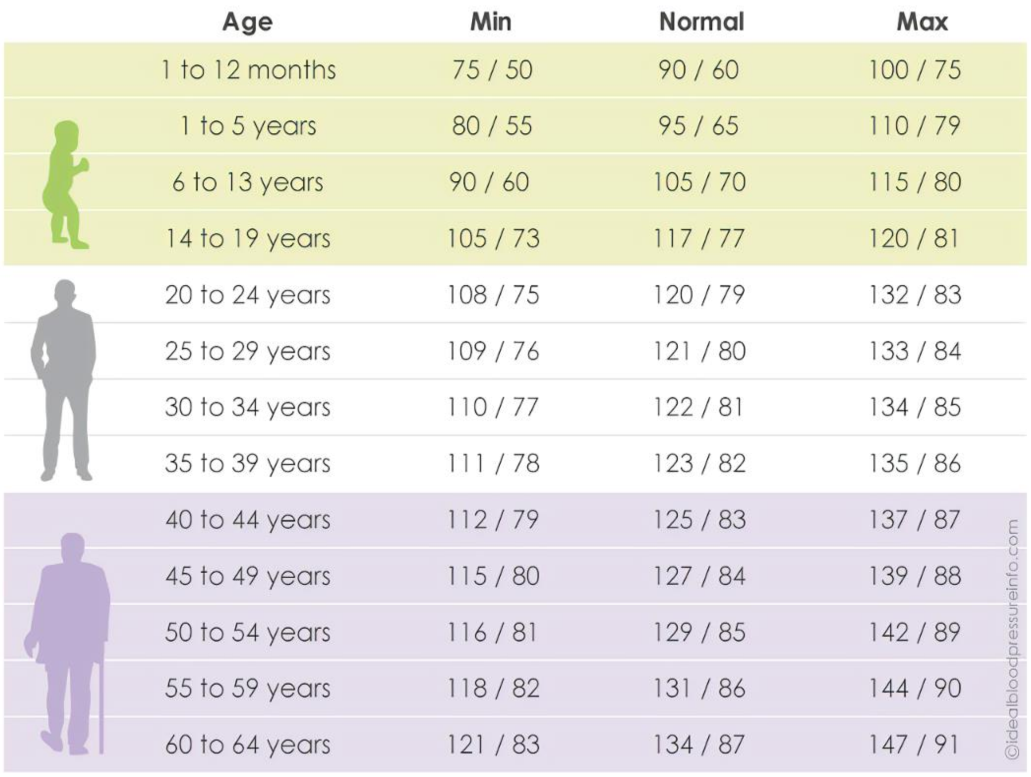
Hypertension Stage 2 is when blood pressure consistently is 140/90 mm Hg or higher.

5. Hypertensive crisis

This stage of high blood pressure requires medical attention.

Watch this video for more details: https://youtu.be/o8DX89jm710

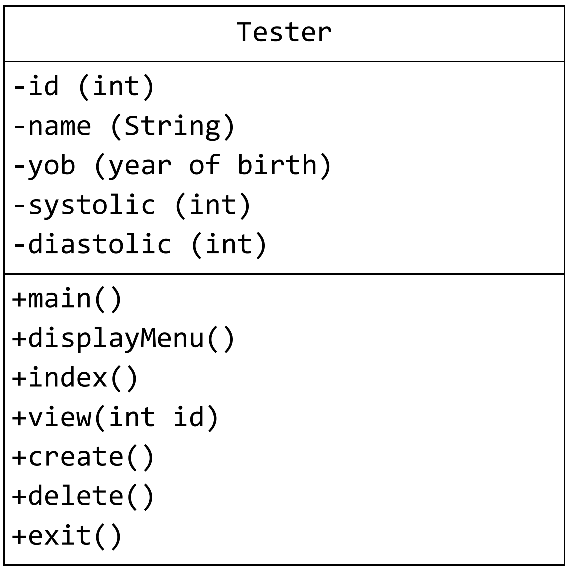




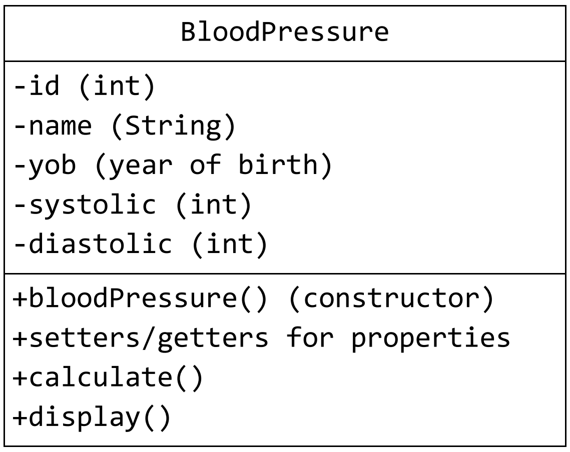
Age	Min	Normal	Max
1 to 12 months	75 / 50	90 / 60	100 / 75
1 to 5 years	80 / 55	95 / 65	110 / 79
6 to 13 years	90 / 60	105 / 70	115 / 80
14 to 19 years	105 / 73	117 / 77	120 / 81
20 to 24 years	108 / 75	120 / 79	132 / 83
25 to 29 years	109 / 76	121 / 80	133 / 84
30 to 34 years	110 / 77	122 / 81	134 / 85
35 to 39 years	111 / 78	123 / 82	135 / 86
40 to 44 years	112 / 79	125 / 83	137 / 87
45 to 49 years	115 / 80	127 / 84	139 / 88
50 to 54 years	116 / 81	129 / 85	142 / 89
55 to 59 years	118 / 82	131 / 86	144 / 90
60 to 64 years	121 / 83	134 / 87	147 / 91

Blood pressure shown as systolic/diastolic

Requirements:



Tester class diagram



Blood pressure class diagram

Write an application to store blood pressure of 5 users. It should be a menu driven application. You need to show the evidence of using classes, objects, methods, properties, setters/getters, constructors, abstraction, inheritance, polymorphism and java collections.

a) In your main class create a displayMenu() method and show the following choices. The program must continuously run until 'exit' is selected (10 marks) :

   i. Create a record. (Ask for user id, then ask data for name, year of birth, systolic & diastolic data.)

   ii. Show blood pressure data for all users

   iii. Show blood pressure data for a selected user.

   iv. Delete all

   v. Exit application

b) Write methods for all actions above, inside main class (10 marks for each)

   i. index() : to show all records. Call BloodPressure.display() here.

   ii. view(int id) : to show one record for the given id. Include blood pressure category and recommendations if there are any for their age. Call BloodPressure.display() here.

   iii. create() : create a new record with appropriate input validation. Call BloodPressure constructor here.

   iv. delete() : delete all records

   v. exit() : exit to system

c) Use a suitable Java collection to store 5 users (10 marks)

d) Calculate the age with year of birth and current year (10 marks)

e) A section on the lessons learnt during this exercise (10 marks)

f) Generate JavaDoc files (5 marks)

g) Upload your complete, commented, tested code as a Git Hub repository to `/final` branch. Include the link in the document (5 marks)

Q3. Blood sugar monitor

Glucose comes from the food we eat and its sugar content. When a person consumes a food with high sugar content, that is turned into glucose. The glucose is then absorbed into the bloodstream with the support of insulin. This is then distributed between the body's cells and used as energy.

Diabetes is a chronic health condition that occurs when the body is unable to properly regulate blood sugar (glucose) levels. According to the International Diabetes Federation (IDF) and the World Health Organization (WHO), diabetes was estimated to be responsible for around 4 million deaths globally in the year 2019.

This chart details goals for specific groups of people with diabetes.

	Before meals (fasting)	**After meals (post-prandial)**	**Other**
Adults with type 1 diabetes	80–130 mg/dL	< 180 mg/dL (1 or 2 hours after)	
Adults with type 2 diabetes	80–130mg/dL	< 180 mg/dL and (1 or 2 hours after)	
Children with type 1 diabetes	90-130 mg/dL		90–150 mg/dL at bedtime/overnight
Pregnant people (T1D, gestational diabetes)	< 95 mg/dL	140 mg/dL (1 hour after)	120 mg/dL (2 hours after)
65 or older	80–180 mg/dL		80–200 mg/dL for those in poorer health, assisted living, end of life
Without diabetes	99 mg/dL or below	140 mg/dL or below	

Watch this video for more details: https://youtu.be/O7l3qg0Z4GE

Requirements:

Tester
-id (int) -name (String) -yob (year of birth) -sugar_level (int)

+main()
+displayMenu()
+index()
+view(int id)
+create()
+delete()
+exit()

Tester class diagram

BloodSugar
-id (int) -name (String) -yob (year of birth) -sugar_level (int)

+bloodSugar() (constructor)
+setters/getters for properties
+calculate()
+display()

Blood sugar class diagram

Write an application to store blood sugar of 5 users. It should be a menu driven application. You need to show the evidence of using classes, objects, methods, properties, setters/getters, constructors, abstraction, inheritance, polymorphism and java collections.

a) In your main class create a displayMenu() method and show the following choices. The program must continuously run until 'exit' is selected (10 marks) :

   i. Create a record. (Ask for user id, then ask data for name, year of birth and blood sugar level)

   ii. Show blood sugar data for all users

   iii. Show blood sugar data for a selected user.

   iv. Delete all

   v. Exit application

b) Write methods for all actions above, inside main class (10 marks for each)

   i. index() : to show all records. Call BloodSugar.display() here.

   ii. view(int id) : to show one record for the given id. Include blood sugar category and recommendations if there are any. Call BloodSugar.display() here.

   iii. create() : create a new record with appropriate input validation. Call BloodSugar constructor here.

   iv. delete() : delete all records

   v. exit() : exit to system

c) Use a suitable Java collection to store 5 users (10 marks)

d) Calculate the age with year of birth and current year (10 marks)

e) A section on the lessons learnt during this exercise (10 marks)

f) Generate JavaDoc files (5 marks)

g) Upload your complete, commented, tested code as a Git Hub repository to `/final` branch. Include the link in the document (5 marks)

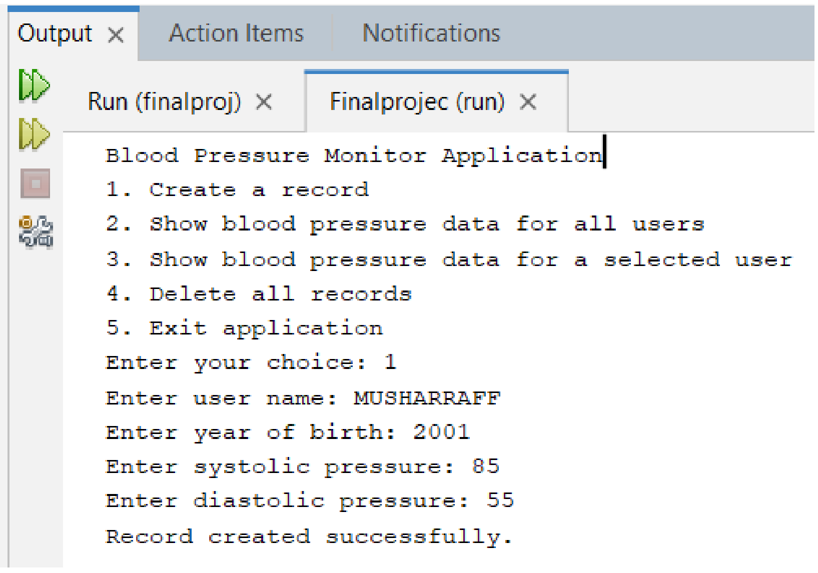
Paste the code here

GitHub repo url: https://github.com/Musharraff5/Final-project.git

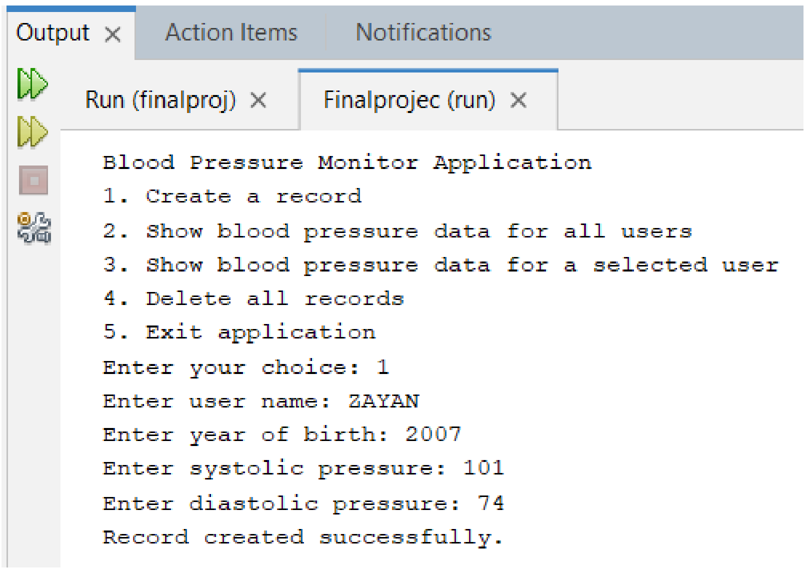
```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package finalprojec;

/**
 *
 * @author 22ug2-0064 Musharraff
 */
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Scanner;


class BloodPressure {
    private int id;
    private String name;
    private int yearOfBirth;
    private int systolic;
    private int diastolic;


    public BloodPressure(int id, String name, int yearOfBirth, int systolic, int diastolic) {
        this.id = id;
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.systolic = systolic;
        this.diastolic = diastolic;
    }


    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getYearOfBirth() {
    return yearOfBirth;
}

public void setYearOfBirth(int yearOfBirth) {
    this.yearOfBirth = yearOfBirth;
}

public int getSystolic() {
    return systolic;
}

public void setSystolic(int systolic) {
    this.systolic = systolic;
}

public int getDiastolic() {
    return diastolic;
}

public void setDiastolic(int diastolic) {
    this.diastolic = diastolic;
}


public int calculateAge() {
    int currentYear = Calendar.getInstance().get(Calendar.YEAR);
    return currentYear - yearOfBirth;
}


public String calculateCategory() {
    if (systolic < 90 && diastolic < 60) {
        return "LOW";
    } else if (systolic >= 90 && systolic < 120 && diastolic <= 80 && diastolic > 60) {
        return "NORMAL";
    } else if (systolic >= 120 && systolic <= 139 && diastolic <= 90 && diastolic > 80) {
            return "PREHYPERTENSION";
        } else if ((systolic >= 140 && systolic <= 159) || (diastolic >= 90 && diastolic <= 99)) {
            return "STAGE 1 HYPERTENSION";
        } else if (systolic >= 160 || diastolic >= 100) {
            return "Hypertension Stage 2";
        } else {
            return "Hypertensive crisis";
        }
    }


    public void display() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Year of Birth: " + yearOfBirth);
        System.out.println("Systolic: " + systolic);
        System.out.println("Diastolic: " + diastolic);
        System.out.println("Age: " + calculateAge());
        System.out.println("Blood Pressure Category: " + calculateCategory());
        System.out.println("----------------------");
    }
}


public class Tester {
    private ArrayList<BloodPressure> users = new ArrayList<>();
    private int currentId = 1;
    private Scanner scanner = new Scanner(System.in);


    public void displayMenu() {
        System.out.println("\nBlood Pressure Monitor Application");
        System.out.println("1. Create a record");
        System.out.println("2. Show blood pressure data for all users");
        System.out.println("3. Show blood pressure data for a selected user");
        System.out.println("4. Delete all records");
        System.out.println("5. Exit application");
        System.out.print("Enter your choice: ");
    }


    public static void main(String[] args) {
        Tester tester = new Tester();
        tester.run();
    }
    public void run() {
        String choice;
        do {
            displayMenu();
            choice = scanner.nextLine();
            switch (choice) {
                case "1":
                    create();
                    break;
                case "2":
                    index();
                    break;
                case "3":
                    System.out.print("Enter user ID: ");
                    int id = Integer.parseInt(scanner.nextLine());
                    view(id);
                    break;
                case "4":
                    delete();
                    break;
                case "5":
                    exit();
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a number between 1 and 5.");
            }
        } while (!choice.equals("5"));
    }


    public void create() {
        System.out.print("Enter user name: ");
        String name = scanner.nextLine();
        System.out.print("Enter year of birth: ");
        int yearOfBirth = Integer.parseInt(scanner.nextLine());
        System.out.print("Enter systolic pressure: ");
        int systolic = Integer.parseInt(scanner.nextLine());
        System.out.print("Enter diastolic pressure: ");
        int diastolic = Integer.parseInt(scanner.nextLine());

        BloodPressure newRecord = new BloodPressure(currentId, name, yearOfBirth, systolic,
diastolic);
        users.add(newRecord);
        currentId++;
        System.out.println("Record created successfully.");
    }


    public void index() {
        if (users.isEmpty()) {
            System.out.println("No records found.");
        } else {
            for (BloodPressure user : users) {
                user.display();
            }
        }
    }


    public void view(int id) {
        boolean found = false;
        for (BloodPressure user : users) {
            if (user.getId() == id) {
                user.display();
                found = true;
                break;
            }
        }
        if (!found) {
            System.out.println("User not found with ID: " + id);
        }
    }


    public void delete() {
        users.clear();
        System.out.println("All records deleted.");
    }


    public void exit() {
        System.out.println("Thank you for using the Blood Pressure Monitor Application.");
        System.exit(0);
    }
}
```

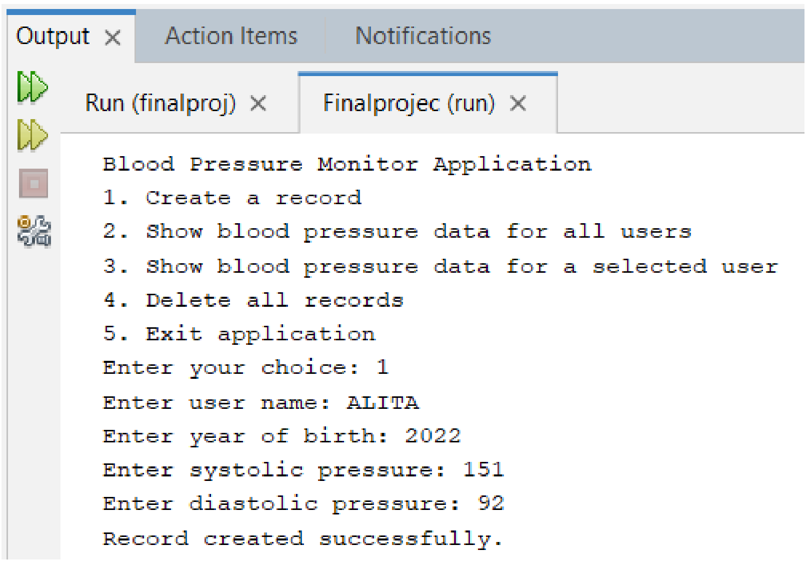
Paste the output screens here



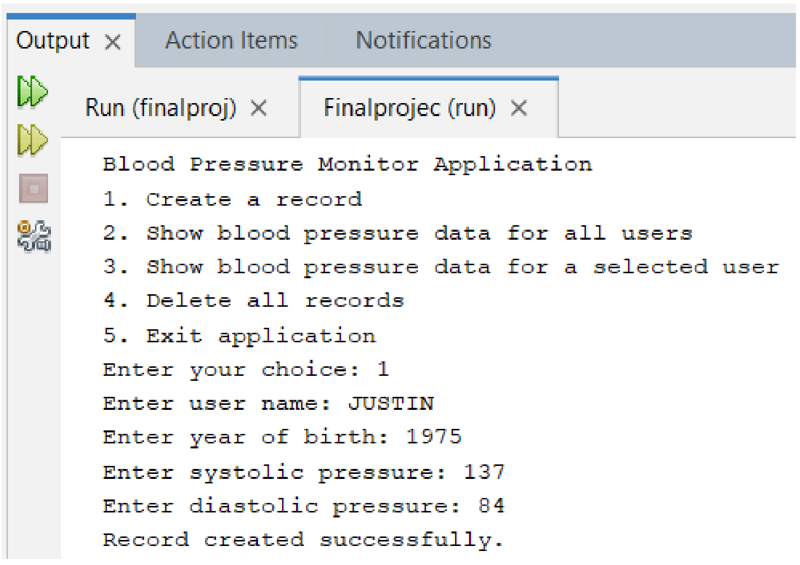
```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 1
Enter user name: MUSHARRAFF
Enter year of birth: 2001
Enter systolic pressure: 85
Enter diastolic pressure: 55
Record created successfully.
```



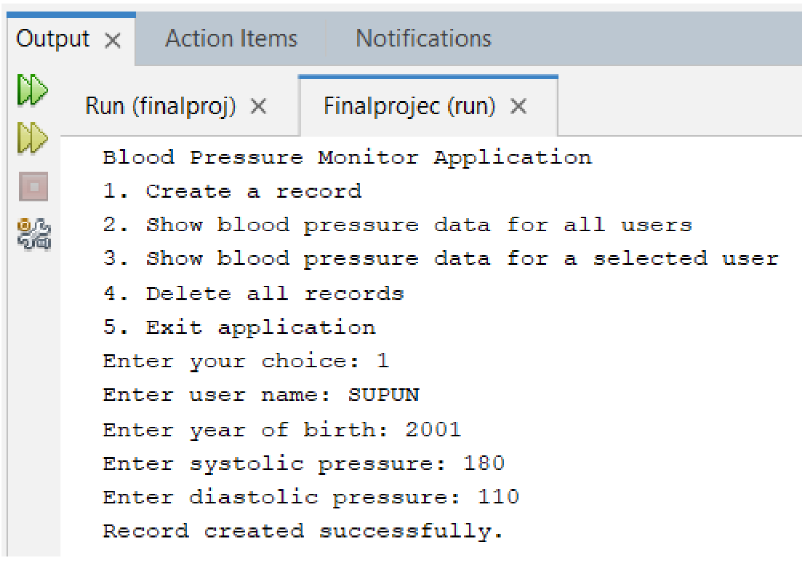
```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 1
Enter user name: ZAYAN
Enter year of birth: 2007
Enter systolic pressure: 101
Enter diastolic pressure: 74
Record created successfully.
```



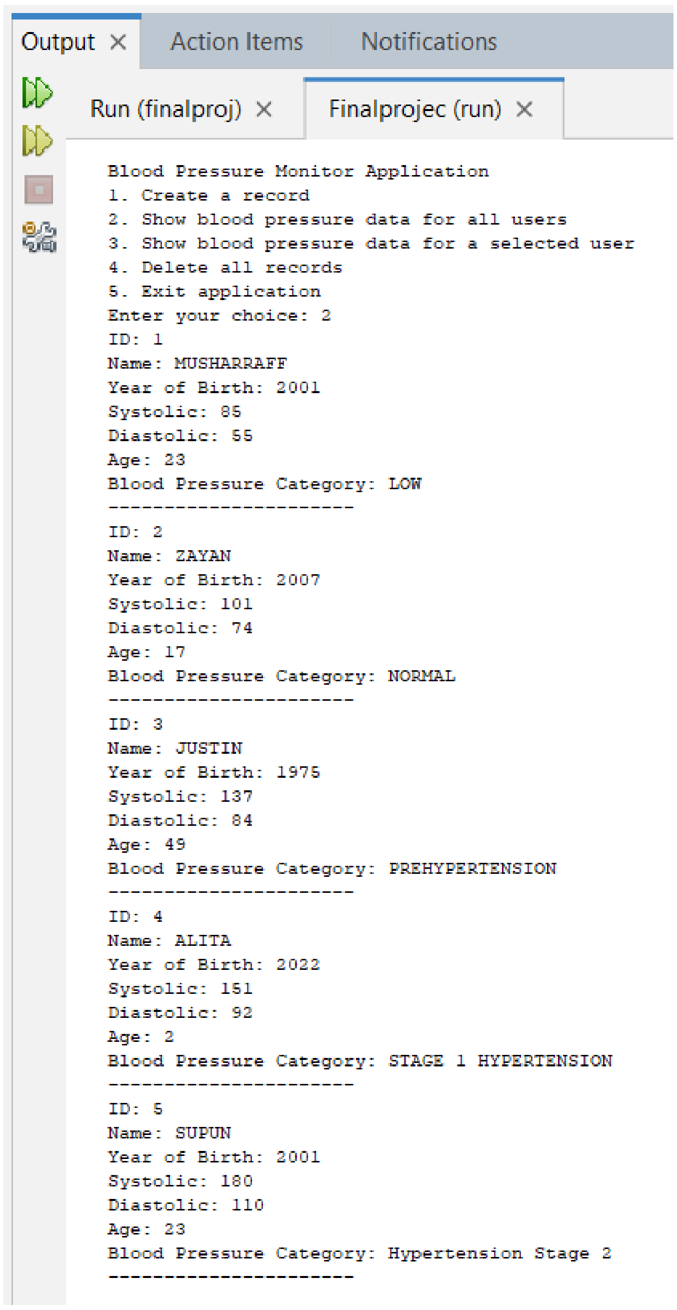
```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 1
Enter user name: ALITA
Enter year of birth: 2022
Enter systolic pressure: 151
Enter diastolic pressure: 92
Record created successfully.
```



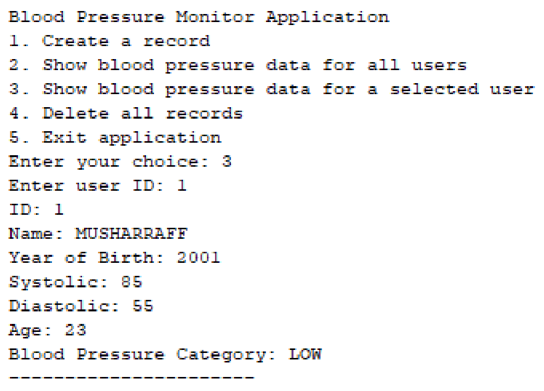
```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 1
Enter user name: JUSTIN
Enter year of birth: 1975
Enter systolic pressure: 137
Enter diastolic pressure: 84
Record created successfully.
```



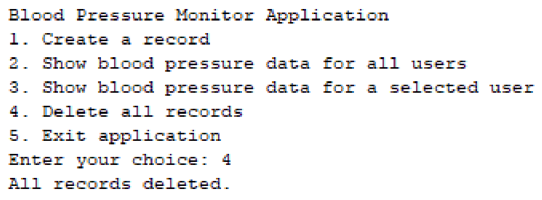
```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 1
Enter user name: SUPUN
Enter year of birth: 2001
Enter systolic pressure: 180
Enter diastolic pressure: 110
Record created successfully.
```



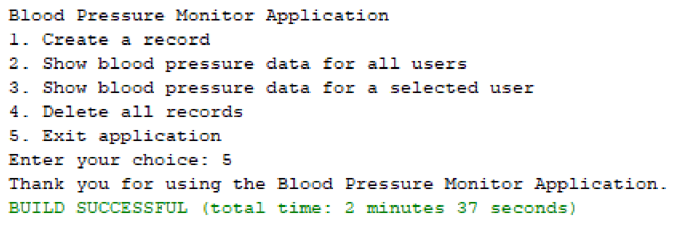
```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 2
ID: 1
Name: MUSHARRAFF
Year of Birth: 2001
Systolic: 85
Diastolic: 55
Age: 23
Blood Pressure Category: LOW
----------------------
ID: 2
Name: ZAYAN
Year of Birth: 2007
Systolic: 101
Diastolic: 74
Age: 17
Blood Pressure Category: NORMAL
----------------------
ID: 3
Name: JUSTIN
Year of Birth: 1975
Systolic: 137
Diastolic: 84
Age: 49
Blood Pressure Category: PREHYPERTENSION
----------------------
ID: 4
Name: ALITA
Year of Birth: 2022
Systolic: 151
Diastolic: 92
Age: 2
Blood Pressure Category: STAGE 1 HYPERTENSION
----------------------
ID: 5
Name: SUPUN
Year of Birth: 2001
Systolic: 180
Diastolic: 110
Age: 23
Blood Pressure Category: Hypertension Stage 2
----------------------
```



```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 3
Enter user ID: 1
ID: 1
Name: MUSHARRAFF
Year of Birth: 2001
Systolic: 85
Diastolic: 55
Age: 23
Blood Pressure Category: LOW
----------------------
```



```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 4
All records deleted.
```

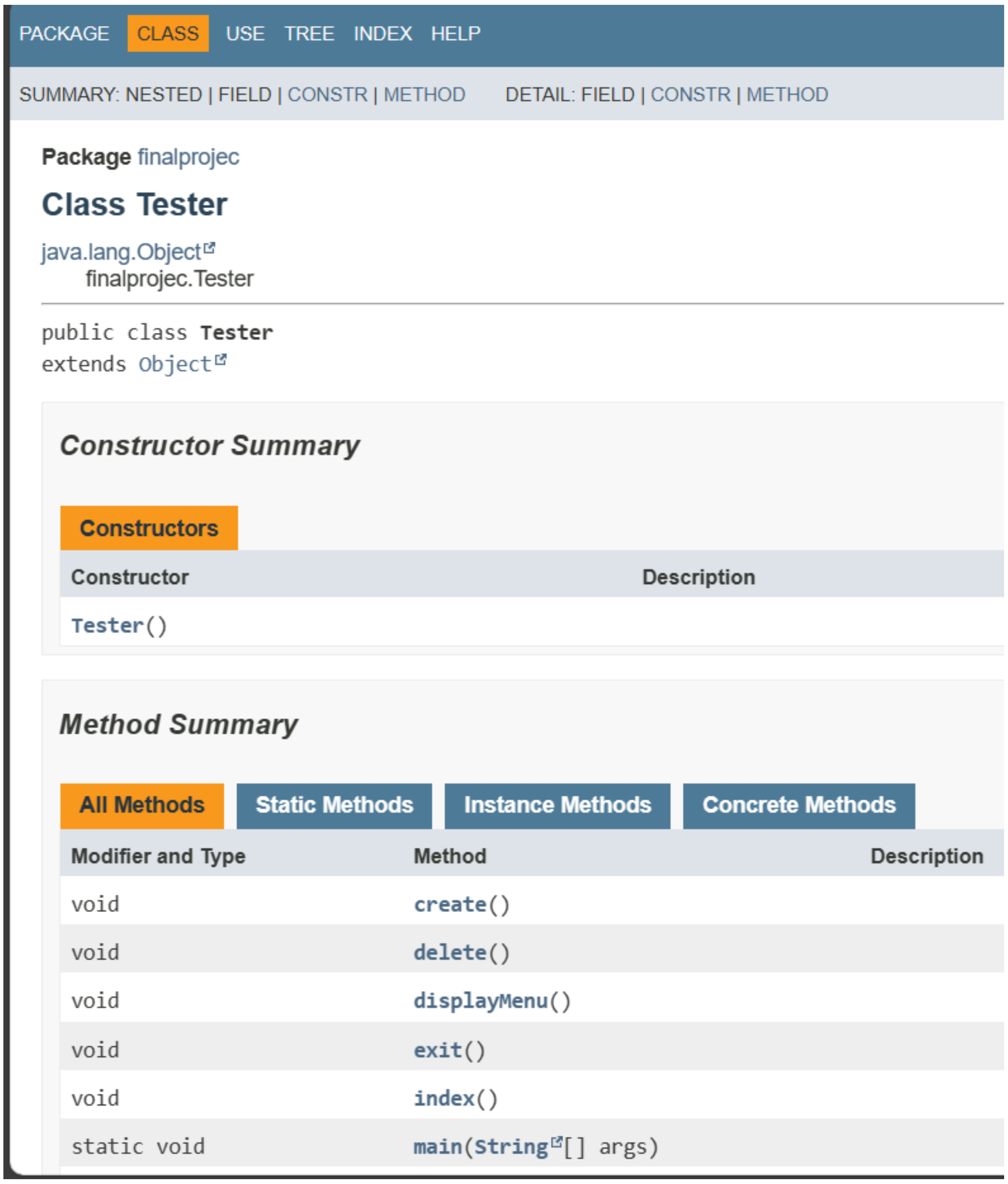


```
Blood Pressure Monitor Application
1. Create a record
2. Show blood pressure data for all users
3. Show blood pressure data for a selected user
4. Delete all records
5. Exit application
Enter your choice: 5
Thank you for using the Blood Pressure Monitor Application.
BUILD SUCCESSFUL (total time: 2 minutes 37 seconds)
```

Lessons learnt

During this exercise, lessons were learned such as understanding the requirements thoroughly before starting the development process, utilizing object-oriented design principles like encapsulation, inheritance, and polymorphism for creating well-structured and maintainable code, handling user input properly to ensure application stability and usability, using appropriate data structures and collections for efficiently storing and manipulating data, implementing robust error handling mechanisms to gracefully handle unexpected situations and prevent application crashes, conducting thorough testing and debugging to identify and rectify any issues or bugs in the code, and emphasizing the importance of documentation, including code comments and user manuals, for understanding the functionality and usage of the application. These lessons provided valuable insights into software development practices, enabling the creation of robust and user-friendly software solutions.

Javadoc's screenshot (Also included in the zip file)


PACKAGE CLASS USE TREE INDEX HELP
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package finalprojec

Class Tester

java.lang.Object
    finalprojec.Tester

```
public class Tester
extends Object
```

Constructor Summary

Constructors

Constructor	Description
`Tester()`	

Method Summary

All Methods
Static Methods
Instance Methods
Concrete Methods

Modifier and Type	Method	Description
`void`	`create()`	
`void`	`delete()`	
`void`	`displayMenu()`	
`void`	`exit()`	
`void`	`index()`	
`static void`	`main(String[] args)`	