|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  | ECS2301 Software Engineering and Project  Mid term project (100 marks) | |  |

## Instructions:

* 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

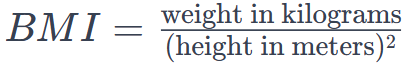
|  |
| --- |
| Important notice: Please note that you need to write minimal code (stub functions). This is to prove your logic and navigation is working. You do NOT need to complete the calculations. That will be completed during the final project. |

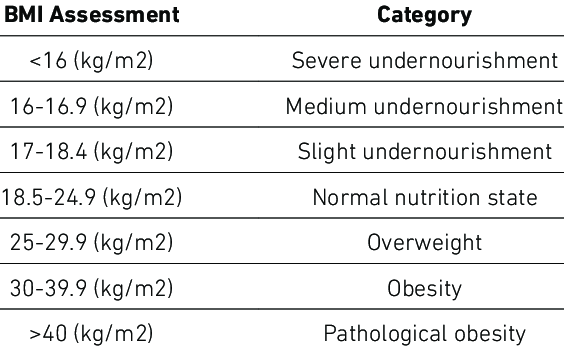
|  |  |
| --- | --- |
| Full name: AF.MohamedImzaan |  |
| Student index:22ug2-0072 |  |
| Date of submission:2024/01/14 |  |

## 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:

Here's a breakdown of the BMI categories:

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 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.

1. In your main class create a displayMenu() method and show the following choices. The program must continuously run until ‘exit’ is selected (10 marks) :
   1. Create a record. (Ask for user id, then ask data for name, year of birth, height & weight)
   2. Show BMI data for all users.
   3. Show BMI data for a selected user.
   4. Delete all.
   5. Exit application.
2. Write methods for all actions above, inside main class (10 marks for each)
   1. index() : to show all records. Call Bmi.display() here.
   2. view(int id) : to show one record for the given id. Call Bmi.display() here.
   3. create() : Call Bmi constructor here.
   4. delete() : delete all records.
   5. exit() : exit to system
3. A section on the lessons learnt during this exercise (20 marks)
4. Generate JavaDoc files (10 marks)
5. Upload your complete, commented, tested code as a Git Hub repository to /midterm branch. Include the link in the document (10 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.

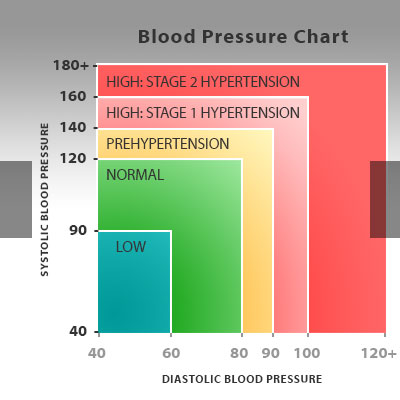
#### 4. Hypertension Stage 2

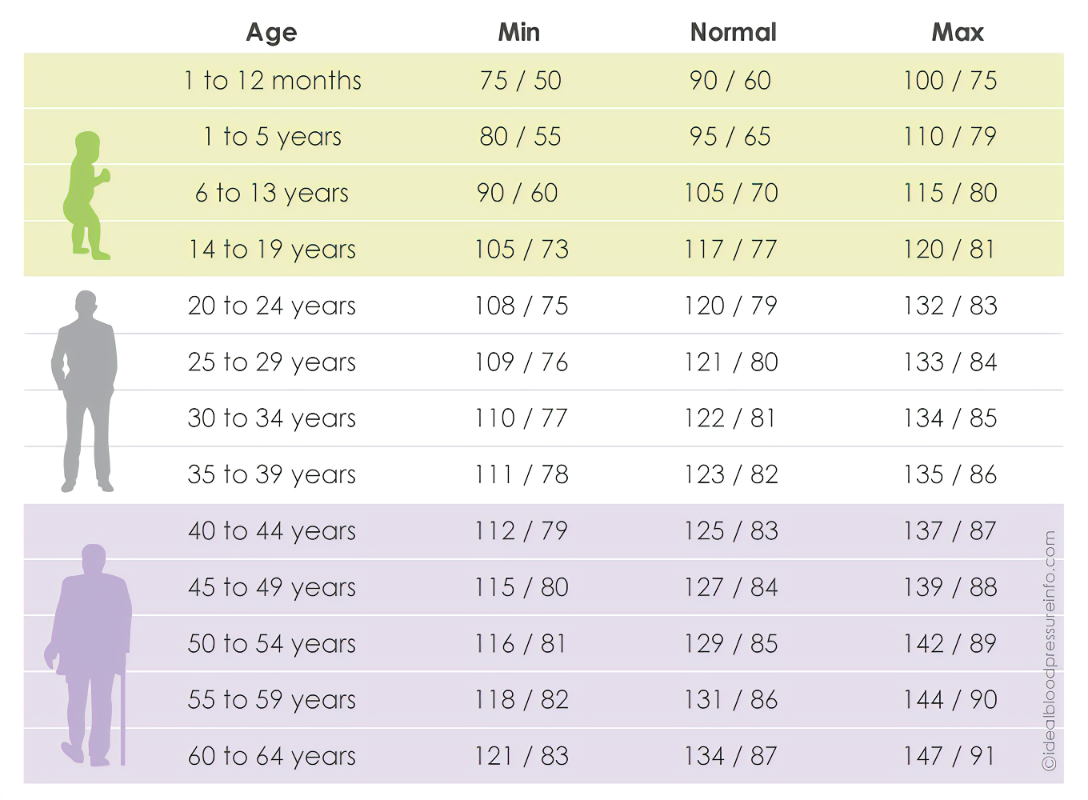
Hypertension Stage 2 is when blood pressure consistently is 140/90 mm Hg or higher.

#### 5. Hypertensive crisis

This stage of high blood pressure requires medical attention.

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



Blood pressure shown as systolic/diastolic

### Requirements:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Tester | | -id (int)  -name (String)  -yob (year of birth)  -systolic (int)  -diastolic (int) | | +main()  +displayMenu()  +index()  +view(int id)  +create()  +delete()  +exit() |   Tester class diagram | |  | | --- | | BloodPressure | | -id (int)  -name (String)  -yob (year of birth)  -systolic (int)  -diastolic (int) | | +bloodPressure() (constructor)  +setters/getters for properties  +calculate()  +display() |   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.

1. In your main class create a displayMenu() method and show the following choices. The program must continuously run until ‘exit’ is selected (10 marks) :
   1. Create a record. (Ask for user id, then ask data for name, year of birth, systolic & diastolic)
   2. Show blood pressure data for all users
   3. Show blood pressure data for a selected user.
   4. Delete all
   5. Exit application
2. Write methods for all actions above, inside main class (10 marks for each)
   1. index() : to show all records. Call BloodPressure.display() here.
   2. view(int id) : to show one record for the given id. Call BloodPressure.display() here.
   3. create() : create a new record. Call BloodPressure constructor here.
   4. delete() : delete all records
   5. exit() : exit to system
3. A section on the lessons learnt during this exercise (20 marks)
4. Generate JavaDoc files (10 marks)
5. Upload your complete, commented, tested code as a Git Hub repository to /midterm branch. Include the link in the document (10 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.

1. In your main class create a displayMenu() method and show the following choices. The program must continuously run until ‘exit’ is selected (10 marks) :
   1. Create a record. (Ask for user id, then ask data for name, year of birth and blood sugar level)
   2. Show blood sugar data for all users
   3. Show blood sugar data for a selected user.
   4. Delete all
   5. Exit application
2. Write methods for all actions above, inside main class (10 marks for each)
   1. index() : to show all records. Call BloodSugar.display() here.
   2. view(int id) : to show one record for the given id. Call BloodSugar.display() here.
   3. create() : create a new record. Call BloodSugar constructor here.
   4. delete() : delete all records
   5. exit() : exit to system
3. A section on the lessons learnt during this exercise (20 marks)
4. Generate JavaDoc files (10 marks)
5. Upload your complete, commented, tested code as a Git Hub repository to /midterm branch. Include the link in the document (10 marks)

# Paste the code here

GitHub repo url:

import java.util.ArrayList;

import java.util.Scanner;

/\*

\* 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

\*/

/\*\*

\*

\* @author Mohamed imzaan

\*/

public class Tester {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

ArrayList<Bmi> bmiRecords = new ArrayList<>();

Scanner scanner = new Scanner(System.in);

int choice;

do {

displayMenu();

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1:

for(int i = 0; i < 5 ; i++){

bmiRecords.add(createBmiRecord(scanner));

System.out.println("-------------------------");

}

break;

case 2:

for (Bmi bmi : bmiRecords) {

bmi.display();

System.out.println("----------------------");

}

break;

case 3:

int id = getId(scanner);

Bmi bmiToDisplay = getBmiById(bmiRecords, id);

if (bmiToDisplay != null) {

bmiToDisplay.display();

} else {

System.out.println("No BMI record found for the given ID.");

}

break;

case 4:

bmiRecords.clear();

System.out.println("All BMI records deleted successfully.");

break;

case 5:

System.out.println("Exiting the application.");

break;

default:

System.out.println("Invalid choice. Please choose again.");

}

} while (choice != 5);

}

private static Bmi createBmiRecord(Scanner scanner) {

System.out.print("Enter ID: ");

int id = scanner.nextInt();

System.out.print("Enter Name: ");

String name = scanner.next();

System.out.print("Enter Year of Birth: ");

int yob = scanner.nextInt();

System.out.print("Enter Height: ");

int height = scanner.nextInt();

System.out.print("Enter Weight: ");

int weight = scanner.nextInt();

return new Bmi(id, name, yob, height, weight);

}

private static int getId(Scanner scanner) {

System.out.print("Enter ID: ");

return scanner.nextInt();

}

private static Bmi getBmiById(ArrayList<Bmi> bmiRecords, int id) {

for (Bmi bmi : bmiRecords) {

if (bmi.getId() == id) {

return bmi;

}

}

return null;

}

private static void displayMenu() {

System.out.println("\nBMI Calculator Menu:");

System.out.println("1. Create a record.");

System.out.println("2. Show BMI data for all users.");

System.out.println("3. Show BMI data for a selected user.");

System.out.println("4. Delete all.");

System.out.println("5. Exit application.");

}

}

class Bmi {

private int id;

private String name;

private int yob;

private int height;

private int weight;

public Bmi(int id, String name, int yob, int height, int weight) {

this.id = id;

this.name = name;

this.yob = yob;

this.height = height;

this.weight = weight;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

public int getYob() {

return yob;

}

public int getHeight() {

return height;

}

public int getWeight() {

return weight;

}

public double calculateBmi() {

return weight / (height \* height);

}

public void display() {

System.out.println("\nID: " + id);

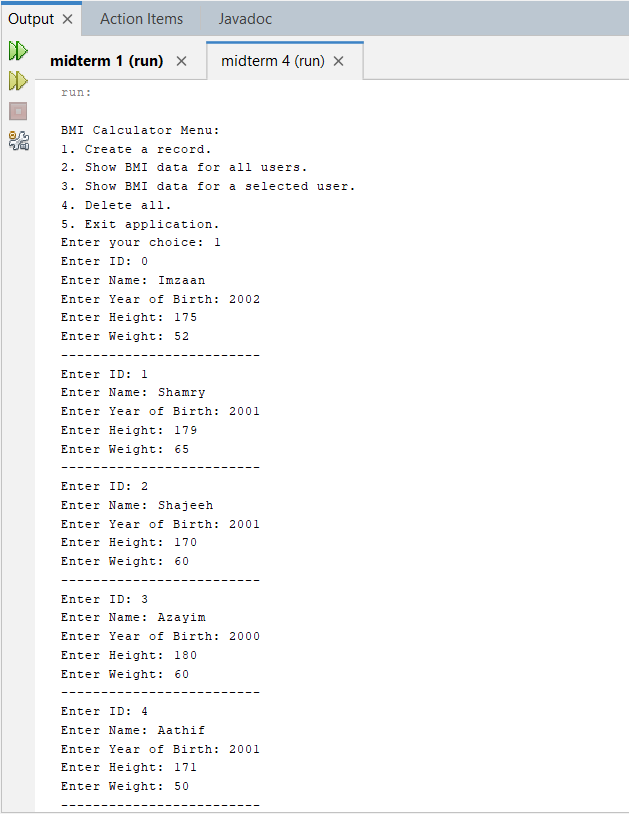
System.out.println("Name: " + name);

}

}

# Paste the output screens here

Crop them appropriately.



# 

# Lessons learnt

In your words, what did you learnt during this exercise?

**Programming that is object-oriented (OOP)**

* Designing and implementing the Bmi class required an understanding of OOP concepts. The BMI records were modeled using the concepts of encapsulation, abstraction, and classes and objects.

**Validation of Input:**

* The Scanner class is used by the application to do basic input validation, making ensuring that the user inputs are of the appropriate data types (int, double, String). In order to avoid mistakes and unexpected behavior, this is essential.

**Error Resolution:**

* In order to accept incorrect choices gently, appropriate error handling methods were put in place, particularly in the menu. This guarantees an intuitive user interface and guards against unexpected input crashing the program.

**Modularity and Organization of Code**:

The code is arranged into understandable methods (such create, index, and view), each of which handles a certain feature. This modular strategy improves maintainability, readability, and troubleshooting ease.

**Data Organization (ArrayList):**

The ArrayList data structure is used to hold BMI records, which highlights how crucial it is to select the right data structures depending on the specifications. Here, variable-sized records may be stored dynamically using an ArrayList.

**Constructor Understanding:**

* To generate instances of the Bmi class, the Java constructor idea was utilized. When an object is created, constructors allow its properties to be initialized.

**Code Readability and Remarks:**

* There are comments in the code that describe each technique and important portions. This is necessary to help people (or your future self) comprehend the code and to ensure the maintainability of the code.

**Handling User Input:**

* Handling user input carefully is necessary, particularly when transferring between multiple data types (int, double, and String). To handle input problems, clear instructions and the appropriate use of nextLine() after numerical inputs were employed.