NEW HORIZON COLLEGE OF ENGINEERING

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

A PROJECT REPORT

for

Mini Project using Java (21CSE56)

on

SMART MACRO TRACKER

Submitted by

CHRIS JORDAN J

USN: 1NH21CS062, Sem- Sec: 5 A

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Academic Year: 2023-24

DrillBit

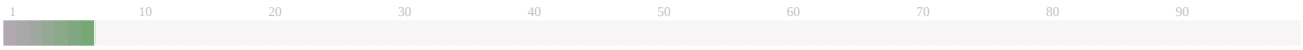
The Report is Generated by DrillBit Plagiarism Detection Software

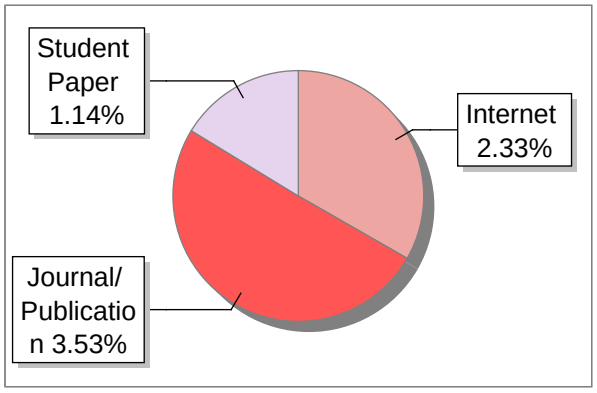
Submission Information

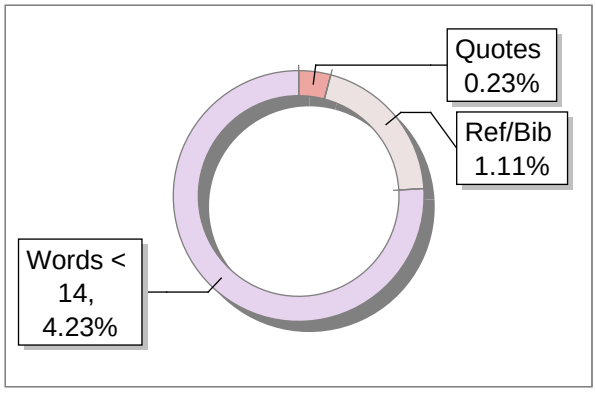
Author Name	1NH21CS062_6721062
Title	SmartMacroTracker_
Paper/Submission ID	1446014
Submitted by	anumm.nhce@newhorizonindia.edu
Submission Date	2024-02-19 11:56:16
Total Pages	50
Document type	Project Work

Result Information

Similarity **7 %**







Exclude Information

Quotes	Not Excluded
References/Bibliography	Not Excluded
Sources: Less than 14 Words %	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

7	31	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	Submitted to Visvesvaraya Technological University, Belagavi	1	Student Paper
2	artsdocbox.com	<1	Internet Data
3	Thesis submitted to dspace.mit.edu	<1	Publication
4	repository-tnmgrmu.ac.in	<1	Publication
5	A transactional model for automatic exception handling by Brun-2011	<1	Publication
6	www.freepatentsonline.com	<1	Internet Data
7	www.freepatentsonline.com	<1	Internet Data
8	revistas.unab.edu.co	<1	Publication
9	Preserving key in XML data transformation by Md-2009	<1	Publication
10	www.geeksforgeeks.org	<1	Internet Data
11	Consumer-driven product development and improvement combined with sens by D-2015	<1	Publication
12	scholarworks.waldenu.edu	<1	Publication
13	Advances in Metered Dose Inhaler Technology Hardware Development by Stein-2014	<1	Publication

14	www.geeksforgeeks.org	<1	Internet Data
15	dochero.tips	<1	Internet Data
16	Recommendation Models for Open Authorization by Shehab-2012	<1	Publication
17	REPOSITORY - Submitted to Siddaganga Institute of Technology Tumkuru on 2023-12-26 10-37	<1	Student Paper
18	towardsdatascience.com	<1	Internet Data
19	ACM Press the 5th ACM Conference- Newport Beach, California (2014., by Gnimpieba, Etienne - 2014	<1	Publication
20	Adaptive radio resource management for interactive user-centric IPTV s by Bor-Jiun-2011	<1	Publication
21	dochero.tips	<1	Internet Data
22	dokumen.pub	<1	Internet Data
23	eprints.umm.ac.id	<1	Internet Data
24	etheses.whiterose.ac.uk	<1	Publication
25	Fabrication and electrical characterization of pyrroleaniline copoly, by Snmezolu, S Durm- 2011	<1	Publication
26	index-of.es	<1	Publication
27	philarchive.org	<1	Publication
28	quizlet.com	<1	Internet Data
29	repositorio.uam.es	<1	Publication
30	Submitted to Visvesvaraya Technological University, Belagavi	<1	Student Paper
31	summerofhpc.prace-ri.eu	<1	Internet Data

ABSTRACT

An inventive Java-MySQL tool called Smart Macro Tracker was created to easily track and analyse daily nutritional consumption. This application was created in NetBeans with Java Swing and uses a JDBC connection to create a strong connection with MySQL for effective data management. The UI is easy to use and makes it simple to enter protein and calorie consumption for breakfast, lunch, snacks, and dinner. This increases user engagement. The Smart Macro Tracker has sophisticated features including customised food programmes and water consumption monitoring in addition to standard tracking. Users are empowered by the programme to make knowledgeable choices regarding their eating habits. Adding eye-catching tables to data visualisation enhances the user experience overall by giving result analysis a compelling new dimension.Additionally, the Smart Macro Tracker incorporates features that go beyond tracking nutrition.

Moreover, the Smart Macro Tracker offers a comprehensive approach to health management by integrating tools for analysing Body Mass Index (BMI), going beyond nutrition tracking. Not only does the visually appealing user interface guarantee simplicity of use, but it also enhances the application's overall appeal. Java, MySQL, and careful UI design are combined in Smart Macro Tracker, a powerful yet approachable programme that offers a smooth platform for nutritional tracking and analysis. It's a useful tool for people looking for a holistic approach to health and wellness because of its many features.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman, New Horizon Educational Institutions, for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha,** Principal, New Horizon College of Engineering, for his constant support and encouragement.

I would like to thank **Dr. Anu**, Professor and Dean-Academics, NHCE, for her valuable guidance.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and HOD, Department of Computer Science and Engineering, for her constant support.

I also express my gratitude to **Dr. Devi Naveen**, Sr. Assistant Professor, Department of Computer Science and Engineering, my mini project reviewer, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

Name: CHRIS JORDAN J

USN: 1NH21CS062

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1. INTRODUCTION

Dietary awareness is essential for promoting a balanced lifestyle in terms of health and fitness. Understanding the need of careful dietary monitoring, the Java-MySQL project Smart Macro Tracker is born, enabling users to track, evaluate, and maximise their daily intake of protein and calories. This paper explores the details of the Smart Macro Tracker, explaining its features, design, and potential overall effects on personal health. Constructed in the NetBeans environment using Java Swing, the Smart Macro Tracker uses a JDBC connection to interface with MySQL easily, guaranteeing reliable and secure data administration. A crucial component of every software programme, the user interface has been painstakingly designed to be both aesthetically beautiful and functional.

By providing sophisticated capabilities, the Smart Macro Tracker goes above and beyond standard applications for monitoring protein and calorie intake. The Smart Macro Tracker's integration of Body Mass Index (BMI) analysis is a crucial feature. This feature highlights the application's dedication to all-encompassing wellness management by giving users a holistic view of their health. The results are easier to read when visually appealing tables are used to portray the data, which promotes a better knowledge of an individual's eating habits. The results are easier to read when visually appealing tables are used to portray the data, which promotes a better knowledge of an individual's eating habits. This research provides a comprehensive analysis of the Smart Macro Tracker, illuminating its design, features, and potential to revolutionise how people think about their nutritional health. This project seeks to revolutionise nutritional monitoring by combining the power of MySQL and Java with a well-thought-out user interface. This will encourage not just the input of information but also a comprehensive journey towards healthy living.

1.1. PROBLEM DEFINITION

The disjointed state of the current nutritional tracking software is the main problem that the Smart Macro Tracker seeks to address. Numerous products on the market lack a unified strategy, which leads to fragmented user experiences and constrained functionality. The goal of this project is to provide a unified platform that can handle a wide range of user needs by providing sophisticated capabilities and streamlining data entry. Furthermore, the lack of user-friendly interfaces in the systems now in use prevents their wider adoption. In order to solve this, the Java Swing framework's easy design is given top priority in the Smart Macro Tracker. This guarantees that users of all technical skill levels can easily navigate and enter their nutritional data. This focus on usability is essential to encouraging users to stay with the programme over the long term and to engage consistently.

The project also aims to close the gap in the market for applications that are more advanced than simple nutritional tracking. With capabilities like BMI analysis and customised food regimens, the Smart Macro Tracker seeks to provide customers a more comprehensive picture of their health. For those who want to make well-informed decisions regarding their food and general health, this thorough approach is crucial.

1.2. OBJECTIVES

1. Develop a user-friendly interface that allows individuals to effortlessly input and track their daily caloric and protein intake for various meals.

2. Incorporate advanced features, such as diet plans and BMI analysis, to offer users a more holistic perspective on their nutritional habits and overall health.

3. Leverage the capabilities of Java programming language and MySQL database. management for efficient data handling, ensuring a secure and scalable foundation.

4. Design an aesthetically pleasing UI within the Java Swing framework to enhance user engagement and accessible to individuals with varying levels of technical expertise.

5. Integrate a feature for users to track and analyze their water intake, recognizing the importance of hydration in overall health management.

6. Implement visually appealing tables for data representation, facilitating a more intuitive interpretation of nutritional data and analysis results.

7. Provide motivational features like transformation pictures.

8. Ensure that the application is scalable to accommodate a growing user base and optimize performance for efficient data processing and analysis.

1.3. METHODOLOGY

Project Planning: Determine important components including meal programmes, nutritional tracking, BMI calculations, and water intake monitoring. Plan the E-R Diagram database layout, taking into account tables for user information, validation, and relationship-building.

User Interface Design: Create a visually appealing and user-friendly interface

Integration of Databases: JDBC is used to implement Java code MySQL connection.

User Authentication: Put authorization and authentication systems in place for users. To verify user credentials, use the database's user information table.

Data Visualization: Create routines that allow data to be retrieved and shown in eye-catching tables. Present the findings of nutritional patterns using Java Swing components.

Testing and Debugging: Make that all of the application's functions, such as user authentication, nutritional tracking, and advanced features, are thoroughly tested.

Planning for Scalability: Integrate user feedback into regular maintenance and upgrades to establish a feedback loop that enables continuous improvement.

1.4. EXPECTED OUTCOMES

The project aims to provide a smart and intuitive nutritional tracking interface together with other important results. Java Swing is used to create an aesthetically beautiful and user-friendly interface that makes interacting with the application easy and fun. Advanced features like BMI analysis, water intake tracking, and customised food regimens are being implemented, which promises a comprehensive approach to nutrition and wellness. By using strong user authentication procedures and JDBC connections, the project seeks to create secure database handling that will protect sensitive user data kept in the MySQL database. The application's emphasis on aesthetically pleasing data will help to produce an in-depth and perceptive examination of dietary trends. The Smart Macro Tracker hopes to give consumers a useful tool for making educated dietary selections and attaining general well-being through the effective integration of various components.

1.5. HARDWARE AND SOFTWARE REQUIREMENTS

1.5.1. HARDWARE REQUIREMENTS

1) Personal computer
2) Minimum of 2GB RAM
3) 64-bit operating system
4) Processor - x86 Compatible processor, 1.7 GHz Clock speed

1.5.2. SOFTWARE REQUIREMENTS

1) Operating system - Windows 7 or above
2) Integrated Development Environment (IDE)- Apache Netbeans IDE
3) Database Management System- MySQL Database
4) Connector: JDBC Driver
5) Graphical User Interface: Swing
6) Web server: Xampp Apache

CHAPTER 2

2. FUNDAMENTALS OF JAVA

2.1 INTRO TO JAVA

Java is a strong and flexible programming language that has made a name for itself in the software development industry. Java was first created at Sun Microsystems in the middle of the 1990s by James Gosling and Mike Sheridan. Since then, it has matured into a language that is well-known for its object-oriented design, scalability, and portability. The Java Virtual Machine (JVM) allows Java to adhere to the "Write Once, Run Anywhere" (WORA) principle, which is one of its most notable characteristics. Java code is compiled into bytecode, which may execute on any device that has a JVM that is compatible with it. Because of its cross-platform interoperability, Java has been used extensively in the creation of a wide range of applications, including enterprise-level systems and mobile and online apps. Java's syntax is intended to be understandable, straightforward, and succinct; it was influenced by C and C++. Because of its object-oriented design, it promotes modular and reusable code, which helps to create applications that are scalable and maintainable. Java's emphasis on consistency and simplicity makes it accessible to both inexperienced and seasoned programmers, which is why developers value it.

The language offers developers a full toolbox for a range of tasks with its extensive standard library and robust collection of APIs (Application Programming Interfaces). Java's vast ecosystem includes frameworks such as Apache Struts, Spring, and Hibernate, which enable developers to create complex, enterprise-level applications quickly. Another important factor in Java's popularity is its dedication to security. The language offers a stable and safe environment for executing programmes with features like the Java Security Manager and bytecode verification. Java is now the go-to platform for creating applications in settings where data security is crucial because of its emphasis on security. Java has remained relevant in recent years because to constant upgrades and enhancements. The language has remained up to date and in line with changing development techniques thanks to the addition of features like lambdas in Java 8, modules in Java 9, and improvements in later releases.

2.2 PROS OF JAVA

Platform Independence (Write Once, Run Anywhere): The Java Virtual Machine (JVM) allows Java to be platform independent. Bytecode is created during the compilation process and is interoperable with any device that has a JVM. This feature provides a uniform experience across different operating systems and lessens the difficulties involved in building and maintaining platform-specific code hence being platform independent. **Java's Object-Oriented Programming (OOP):** paradigm allows for modular and reusable code through the use of OOP principles such as polymorphism, inheritance, and encapsulation. OOP encourages a methodical approach to programming, which facilitates the design, implementation, and upkeep of complicated software systems. **Maintainability and Code Reusability**: Java promotes code reusability through its focus on modularity and OOP. Classes and other components can be effectively reused by developers in many areas of the program. Codebases become easier to manage as a result.

Rich Standard Library and APIs: Java comes with a huge range of classes and methods for typical programming tasks in its standard library, known as Java Standard Edition, or SE. Because of the wide range of APIs, development is made easier, enabling programmers to concentrate on application-specific logic.

Java's adaptability and ecosystem: are demonstrated by the range of fields in which it is used, such as web development, Android mobile app development, and enterprise-level systems. Because it offers pre-made solutions for common problems, the vast ecosystem of third-party libraries and frameworks, including Spring and Hibernate, improves Java's capabilities and speeds up development.

Features of Security: Java places a high priority on security with tools like bytecode verification and the Java Security Manager. These safeguards help to provide a secure runtime environment, which qualifies Java for use in applications that depend on system integrity and data security.

2.3 DATA TYPES

Java data types provide the kinds of values that can be stored in a variable. There are two primary classifications: primitive and non-primitive. Primitive: Simple values like integers and characters are represented by primitive types like int, double, float, char, Boolean, byte, short, etc. Non- Primitive: User-defined classes, strings, and arrays are examples of non-primitive types that can handle more complicated data structures. In Java, selecting the right data types is essential to maximizing program functionality and memory efficiency.

Type	Description	Default	Size	Example Literals	Range of values
boolean	true or false	false	1 bit	true, false	true, false
char	Unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\', '\n', 'β'	characters representation of ASCII values 0 to 255
int	twos-complement intger	0	32 bits	-2,-1,0,1,2	-2,147,483,648 to 2,147,483,647
long	twos-complement integer	0	64 bits	-2L,-1L,0L,1L,2L	-9,223,372,036,8 to 9,223,372,036,8
float	IEEE 754 floating point	0.0	32 bits	1.23e100f , -1.23e-100f , .3f ,3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d , -123456e-300d , 1e1d	upto 16 decimal digits

Table: 1. Description of Data Types in Java

2.4 CONTROL FLOW

The sequence in which statements are carried out within a Java programme is known as the Java control flow. It consists of looping constructions (for, while, do-while) for repeating activities, conditional expressions (if, otherwise if, else) for decision-making, and branching statements (break, continue, return) to manage programme flow. Unless these control flow structures change it, the main method is executed first and then proceeds in a sequential manner. Loops repeat code blocks recursively, branching statements manage the flow within loops or procedures, and conditional statements let code to run based on predefined criteria. Java programmes are guaranteed to execute logically thanks to this organised flow.

(i) **IF/ IF- ELSE/ IF- ELSE IF**: The "if" statement is used to execute code conditionally. This is further extended by the "if-else" statement, which offers an additional block to run in the event that the condition is false. “If else ladder” is used is further applications that requires multiple condition processing.

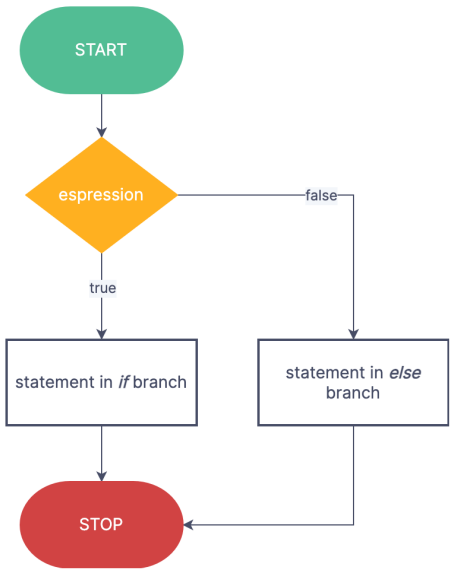


Fig: 1. Overview of Basic Conditional Statements

(ii) **LOOPING CONSTRUCTS**: Loops are control flow structures that let a programming block run repeatedly. When the number of iterations is known ahead of time, the "for" loop is utilised, providing the initialization, condition, and iteration expressions in a condensed format. The "while" loop, on the other hand, checks the condition before each iteration and

keeps running as long as it is true. By automating repeated activities and iterating over data structures or sequences, both loops enable the efficient repetition of code.

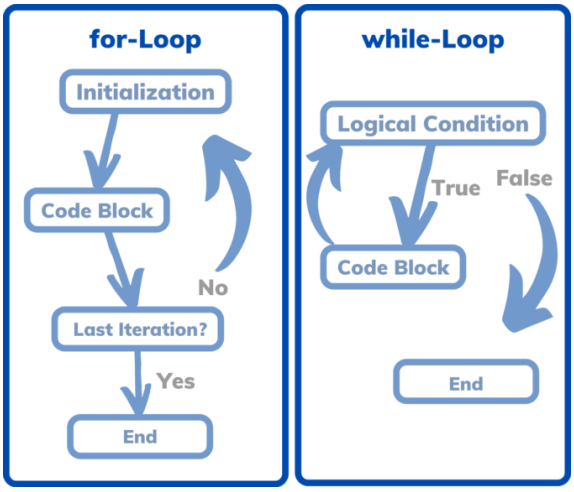


Fig: 2. Flow Chart of Looping Statements in Java

(iii) **BREAK & CONTINUE**: When a loop in Java is about to end prematurely, the "break" statement is used to stop it immediately and go on to the following line outside the loop. Conversely, the "continue" command is used in loops to exclude the remaining code from the current iteration and move straight on to the next one, so avoiding the lines that follow in the loop body.

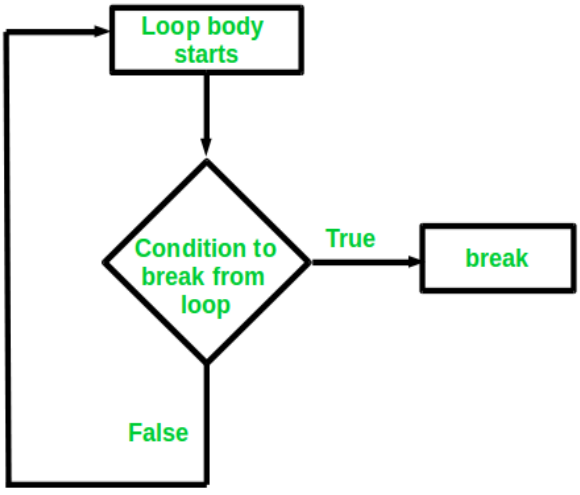


Fig: 3. Working of Break Statement

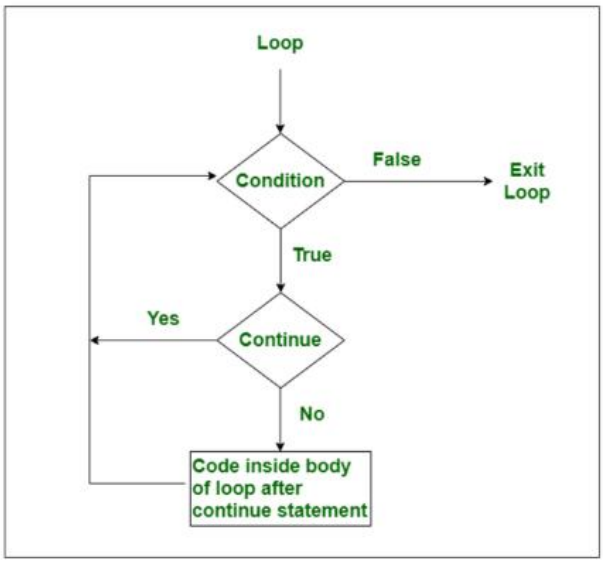


Fig: 4. Working of Continue Statement

2.5 METHODS

Methods are modular chunks of code that are intended to carry out certain tasks; this helps to improve code organisation and encourage reusability. A method is defined by defining its return type, which indicates the data type of the result or "void" if none, its visibility (public, private, or protected), its unique identifier (method name), and any parameters it could accept. The instructions for the given job are contained in the method body, which encapsulates features for effective code organisation. By allowing values to be sent to the procedure, parameters provide flexibility and adaptability. Depending on the return type, the return statement may or may not be required. It returns a value to the caller code. Through the use of methods, programmers may adopt a modular approach to writing code, which helps to create more readable, maintainable, and reusable Java code by breaking complicated tasks down into manageable components.

2.6 OBJECT ORIENTED PROGRAMMING

The programming paradigm known as object-oriented programming, or OOP, organises code around objects, which are just instances of classes. It encourages polymorphism, which allows objects to take on different forms, inheritance, which permits classes to inherit traits and behaviours, and encapsulation, which groups data and methods together. OOP makes software development more efficient and maintainable by facilitating code organisation, reusability, and a more natural representation of real-world things.

(i) OBJECTS: In Java, objects are instances of classes that symbolise actual physical objects. They enable communication within a programme by encapsulating behaviours and data. Objects are the building blocks of object-oriented programming; they are made from class blueprints.

(ii) CLASS: In Java, a class is a blueprint that specifies an object's composition and actions. It acts as a template, including properties and functions, directing the production of object instances.

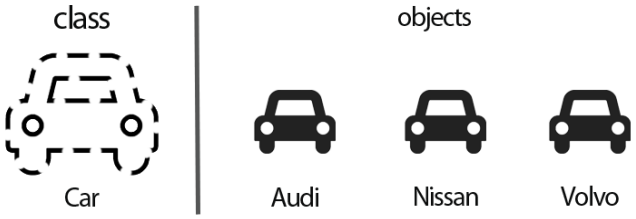


Fig: 5. Pictorial representation of a Class and Objects in Java

(iii) ENCAPSULATION - Java encapsulation includes combining methods and data into a class and limiting access to internal features. It encourages modularity, improves security, and makes efficient code organization easier.

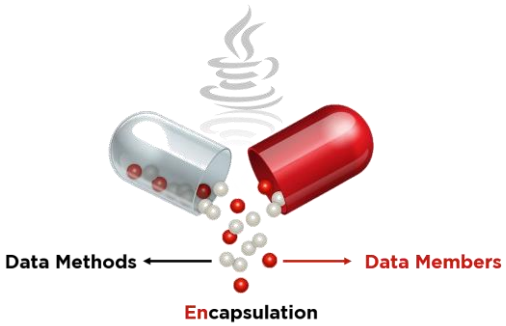


Fig: 6. Encapsulation

(iv) INHERITANCE- A subclass can inherit characteristics and behaviours from a superclass through inheritance, which creates a hierarchical connection between classes. In Java programmes, it encourages modularity, hierarchy, and code reuse.

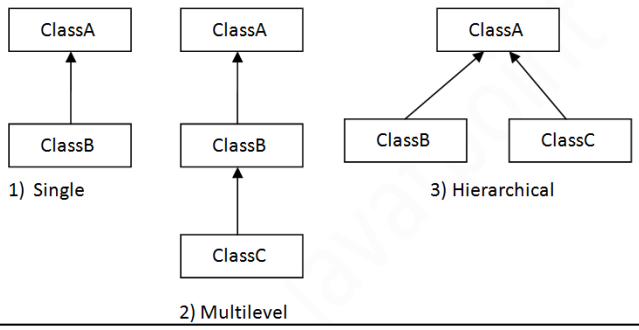


Fig: 7. Single, Multilevel and Hierarchal Representation

(v) POLYMORPHISM- Objects may assume many forms thanks to polymorphism, which is accomplished by overloading and overriding methods. It improves flexibility in Java by allowing several implementations to be represented by a single interface.

Fig: 8. Polymorphic Visualization of a Man

(vi) ABSTRACTION- Real-world elements are modelled in Java as abstract classes or interfaces through the process of abstraction. It makes important features stand out while concealing superfluous details, making code understanding and upkeep easier.

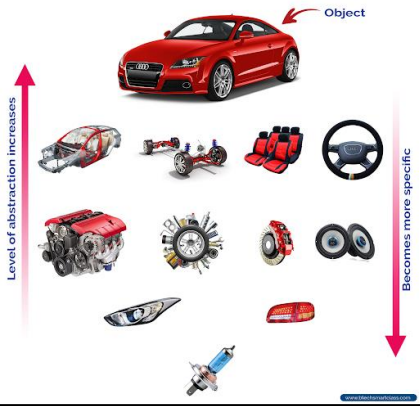


Fig: 9. Abstraction Depicted using a Car

(vii) INTERFACE- An interface in Java creates a contract that outlines the procedures that classes that implement it must follow. By defining a set of standard behaviours, it facilitates modular development, multiple inheritance, and code consistency.

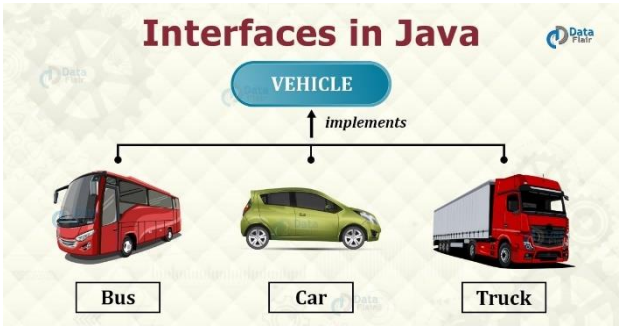


Fig: 10. Overview of Interfaces

2.7 EXCEPTION HANDLING

Runtime faults and unusual circumstances may be handled in an organised manner with Java exceptions. Try-catch blocks allow developers to separate code that is prone to errors, therefore increasing the resilience of the programme. While the finally block makes sure that all necessary cleaning procedures are carried out, several catch blocks provide customised responses to various errors. Java's "try-with-resources" feature makes managing resources easier. By enabling developers to handle mistakes without having to force abrupt shutdowns, exception handling encourages graceful programme termination and improves the general reliability and maintainability of code.

1. **TRY- CATCH**: The basic structure of Java exception handling is as follows: code that could raise an exception is enclosed in a try block, and any number of catch blocks that deal with particular exceptions come after.

```
//try- catch
try {
    // code that may throw an exception
} catch (ExceptionType e) {
    // handle the exception
}
```

Fig: 11. Try Catch Block Pseudocode

2. **MULTIPLE CATCH BLOCK**: makes it possible to handle various error kinds independently by following a single try block with numerous catch blocks.

```
//multiple catch
try {
    // code that may throw an exception
} catch (ExceptionType1 e1) {
    // handle exception type 1
} catch (ExceptionType2 e2) {
    // handle exception type 2
}
```

Fig: 12. Multiple Catch Block Pseudocode

3. **FINALLY**: an optional block that comes after the try-catch blocks and contains code that runs whether or not an exception is raised. It is frequently applied to cleaning tasks.

```
//finally
try {
    // code that may throw an exception
} catch (Exception e) {
    // handle the exception
} finally {
    // cleanup code (executed whether exception occurs or not)
}
```

Fig: 13. Finally Block Pseudocode

4. **TRY WITH RESOURCES**: Resource management is made easier with this Java 7 feature, which automatically closes resources like files and connections. Within the try block, resources are declared, and at the conclusion of the block, they are automatically closed.

```
//try with resources
try (ResourceType resource = new ResourceType()) {
    // code using the resource
} catch (Exception e) {
    // handle the exception
}
```

Fig: 14. Try with Resources Pseudocode

2.8 FILE HANDLING

One essential component of controlling data persistence in programmes is handling Java files. Path and Files are only two of the many utilities available in the java.nio.file package for flexible file handling. Readers/Writers and input/output streams make it easier to read and write data, which makes working with files easier. Exception handling guarantees that applications handle unforeseen situations with grace and is essential for strong error control during file operations. File handling includes a number of operations that are necessary for effective data management, including generating, deleting, copying, transferring, and verifying file properties. Try-with-resources improves resource management by automatically terminating files when they are no longer needed.

Method	Type	Description
canRead()	Boolean	It tests whether the file is readable or not
canWrite()	Boolean	It tests whether the file is writable or not
createNewFile()	Boolean	This method creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	It tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

Table: 2. Pre- Defined Functions used in File Handling

2.9. PACKAGES AND IMPORT

Packages are a way for classes and interfaces to be grouped together in Java according to their function or goal. Enclosing similar code in a package reduces name conflicts and improves code organization. Developers can leverage features from other areas of the program or external libraries by using the 'import' line to access classes or packages outside the current package. It facilitates the management and upkeep of big projects and simplifies code by offering a shorthand notation for class references. Java programming's modularity, reusability, and maintainability are enhanced by the efficient use of packages and import statements, which helps to create scalable and orderly programs.

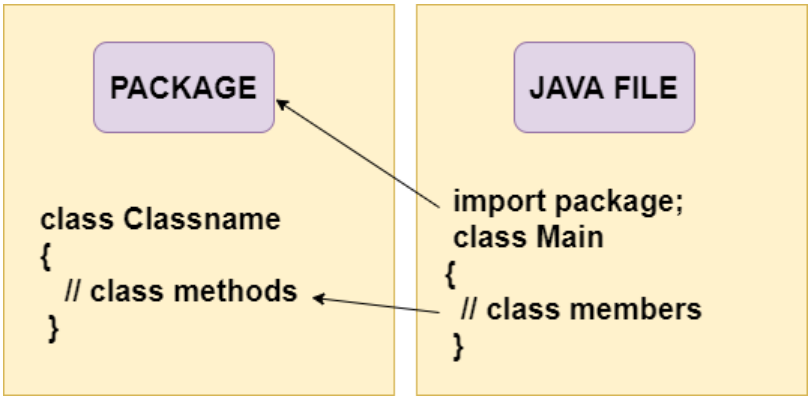


Fig: 15. Packages and Import in Java

2.10. INTERFACES

A template for building abstract types with method signatures but no implementation can be found in interfaces. They lay forth a contract that classes that implement the interface have to go by in order to maintain a uniform structure amongst various implementations. Interfaces allow for multiple inheritance, meaning that a class can implement numerous interfaces. Java interfaces encourage code consistency and modularity, which makes it easier to write loosely connected, highly maintainable programs. Java 8 included default and static methods, which increased the flexibility of interfaces and made it possible to create methods and utility methods inside of them. In

object-oriented programming, interfaces are essential for creating abstraction, encapsulation, and polymorphism. This promotes flexibility and extensibility of code in Java programs.

2.11. CONCURRENCY

In Java, concurrency is the ability for many threads to run simultaneously, enabling programs to carry out numerous activities at once. Through its java.util.concurrent package, which offers classes like Executors and ThreadPools for effective thread management, Java offers strong support for concurrency. In multithreaded contexts, the synchronised keyword prevents data corruption by guaranteeing secure access to shared resources. Java's Concurrency API allows programmers to create sophisticated synchronisation techniques with tools like locks, semaphores, and barriers. Concurrent collections, like ConcurrentHashMap, facilitate the development of high-performance and scalable systems. CompletableFuture and other features in the java.util.concurrent package make asynchronous programming easier and improve responsiveness. Good concurrency management in Java enhances application responsiveness, scalability, and resource efficiency, making it an essential component of contemporary software development.

```java
public class SimpleConcurrencyExample {

    public static void main(String[] args) {
        // Create and start two threads
        Thread thread1 = new Thread(() -> printNumbers("Thread 1"));
        Thread thread2 = new Thread(() -> printNumbers("Thread 2"));

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

    private static void printNumbers(String threadName) {
        for (int i = 1; i <= 5; i++) {
            System.out.println(threadName + ": " + i);
            try {
                // Introducing a short delay to emphasize concurrency
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Fig: 16. Simple code Demonstrating Concurrency in Java

CHAPTER 3

3. JAVA SWING GUIDE

3.1 INTRO TO JAVA SWING

A powerful GUI toolkit called Java Swing enables programmers to design dynamic and aesthetically pleasing user interfaces for Java applications. Java Swing is an add-on for the Abstract Window Toolkit (AWT) that offers a wide range of components, ranging from labels and buttons to more intricate features like tables and trees. Swing components behave consistently across different operating systems and are lightweight compared to AWT. Understanding Java Swing's essential function in GUI development—which enables the building of dynamic, responsive applications—is part of the introduction to the framework. Swing's versatility is used by developers to create programmable interfaces, add layout managers for effective component placement, and control user interactions via event-driven programming.

3.2 SWING COMPONENTS

Swing components in Java are essential elements for building graphical user interfaces (GUIs). A brief overview of some commonly used Swing components include:

1. **JButton**: A clickable element used to initiate operations inside a graphical user interface is called a JButton in Java Swing. It offers a way to record user input, including button clicks, and is frequently used to execute a range of functions, from starting programs to moving between different areas of an application, proving to be extremely responsive.

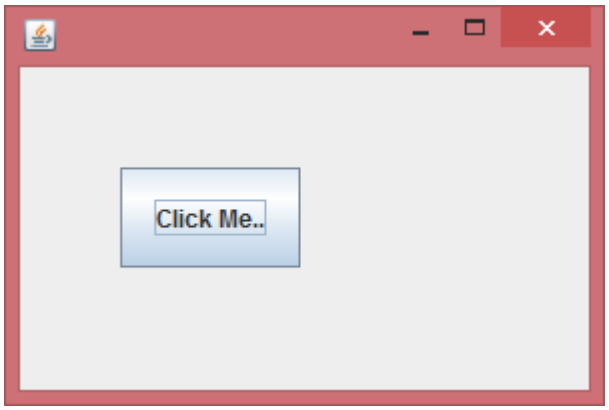


Fig: 17. JButton Using Java Swing

2. JLabel: JLabel is a Swing component used in graphical user interfaces to show non-editable text or pictures. It functions as a descriptive element by giving a GUI's users a means to include text, titles, or captions. JLabel is frequently used by developers to provide static information to the user or annotate other components.

Fig: 18. JLabel Using Java Swing

3. JTextField: Java Swing's JTextField text input component lets users enter and modify single-line text. It is frequently used to gather input from users in forms and dialogue boxes and offers a simple user interface for interacting with and storing text in Java applications.

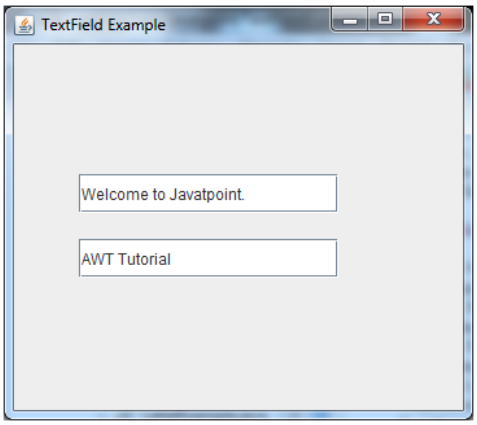


Fig: 19. JTextField Using Java Swing

4. JTextArea: A Swing component called JTextArea makes it easier to enter and see multiline text. Multiple lines of text may be input and edited by users, making it helpful in situations when a large amount of text must be input or outputted.

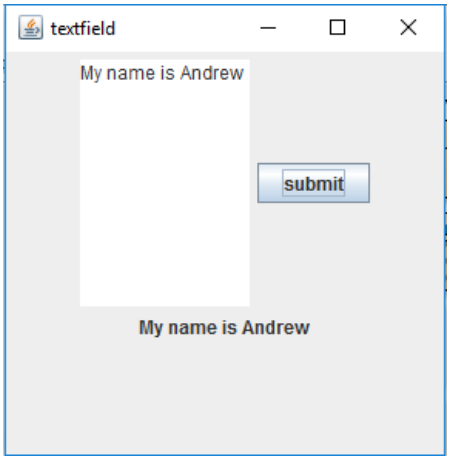


Fig: 20. JTextArea Using Java Swing

3.3 CONTAINERS

Components that store and arrange other components (such buttons, labels, and text fields) inside a graphical user interface (GUI) are known as containers in Java Swing. Containers give the GUI organisation and structure, enabling developers to efficiently manage and organise the visual components. Swing offers a variety of container types, such as:

I) JFrame: In Java Swing, a graphical application's primary window is represented by the JFrame. It is a top-level container that offers the UI's general framework. Title bars, borders, and the ability to maximise, minimise, and exit the programme are among the features that JFrame offers. JFrame is used by developers to build the main window of their graphical programmes, into which they embed different components.

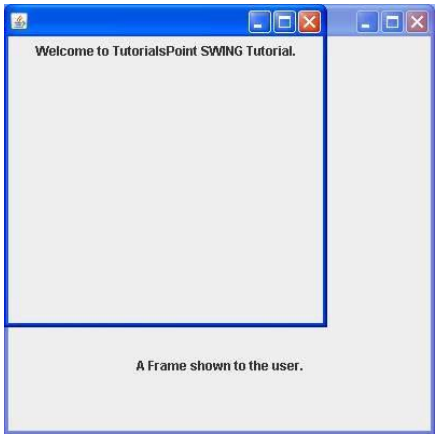


Fig: 21. JFrame Using Java Swing

II) JPanel: Components inside a graphical user interface can be arranged and grouped using the Java Swing generic container JPanel. It acts as a flexible building block that enables programmers to add elements like buttons, labels, or other panels to organise the application's layout. JPanel is frequently integrated into JFrame or other panels, making it easier to create user interfaces that are well-organized and structured.

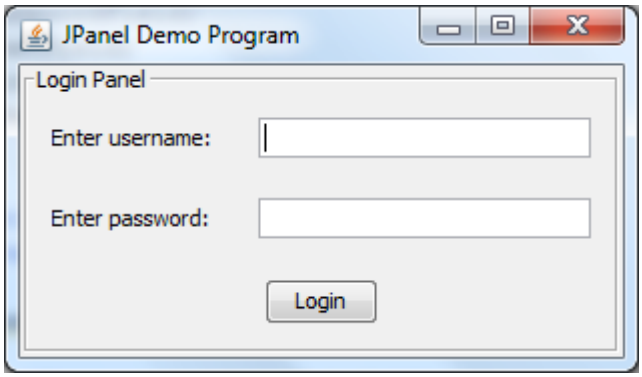


Fig: 22. JPanel Using Java Swing

3.4 LAYOUT MANAGER

A layout manager in Java Swing is a device that arranges and arranges GUI elements inside containers. It takes care of the component sizing and layout automatically, ensuring a uniform and adaptable display. By controlling the spatial connections between components, common layout managers like AbsoluteLayout, FlowLayout, BorderLayout, and GridLayout make it easier to develop responsive user interfaces that adapt better to various screen sizes and resolutions.

I) **FLOW LAYOUT**: When a row is filled, Java Swing's FlowLayout layout manager advances to the next one, arranging components in a left-to-right flow. It works well in situations when elements should be shown naturally without having their placements specified.

Fig: 23. Visualizing Flow Layout Feature in Java Swing

II) **BORDER LAYOUT**: Java's BorderLayout A container may be divided into five regions using the layout manager Swing: North, South, East, West, and Centre. These regions serve as locations for components, offering a straightforward but efficient method of organizing components.

Fig: 24. Visualizing Border Layout Feature in Java Swing

III) **GRID LAYOUT**: A layout manager called GridLayout arranges elements in a grid while guaranteeing that every cell has the same size. By automatically allocating the components equally across rows and columns, it streamlines the assembly process.

Fig: 25. Visualizing Grid Layout Feature in Java Swing

IV) **ABSOLUTE LAYOUT**: With the use of exact pixel coordinates and layout manager, developers may define the size and location of any component inside a container. Although it gives the most control over component's location, it may lead to less flexible layouts

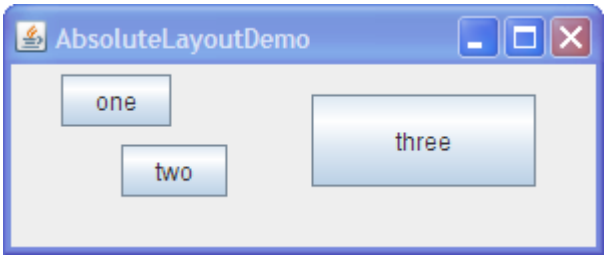


Fig: 26. Visualizing Absolute Layout Feature in Java Swing

3.5 EVENT HANDLING

A key component of developing interactive graphical user interfaces (GUIs) using Java Swing is event management. During user interactions, Swing components—such as buttons or mouse actions—generate events. Developers construct event listeners, which are interfaces specifying methods to handle certain events, in order to record and react to these events. Typical listeners are MouseListener for mouse-related activities and ActionListener for button clicks. By registering with the proper components, event listeners enable the system to call the relevant methods when events take place. This makes it possible for programmers to design responsive programmes in which user input-triggered GUI components perform predetermined actions. Building dynamic and user-friendly Java Swing applications requires event handling in order to make sure that the interface responds to user inputs naturally, which improves the overall user experience.

3.6 ACTION LISTENERS

When a user interacts with GUI elements like as buttons or menu items, an ActionListener in Java Swing is an interface that is used to handle action events. The actionPerformed() function, which outlines the actions to be performed when an event happens, must be provided by classes implementing ActionListener. When an associated component is manipulated, as by clicking a button, this function is automatically called. ActionListener allows the developers to react to user input, resulting in a responsive application.

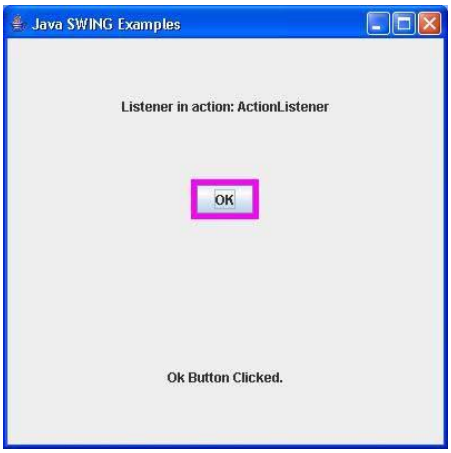


Fig: 27. Action Listeners to Handle Action Events in Java Swing

CHAPTER 4

4. FUNDAMENTALS OF DBMS

4.1 INTROUCTION TO DBMS

A software programme that makes the construction, maintenance, and organisation of databases easier is called a database management system (DBMS). It acts as a bridge between users and databases, offering a streamlined and organised method of storing, retrieving, and modifying data. Multiple users can access data concurrently and with data integrity and security thanks to DBMS. It communicates with databases using a query language, usually SQL. Widely utilised across many sectors, database management systems (DBMS) are essential for managing massive amounts of data, enabling effective data storage and retrieval, and supporting data-driven decision-making processes in contemporary applications and businesses.

4.2 FEATURES OF DBMS

. Data Integrity: By imposing constraints like primary keys, foreign keys, and unique constraints, DBMS guarantees data correctness and consistency, by also reducing redundancy.

. Data Security: To protect sensitive data, DBMS includes procedures for authentication and access control (with admin given the highest leeway), making sure that only those with the proper authorization may see and alter data enforcing both consistency and security.

. Concurrency control: ensures that numerous users may interact with the data at the same time without compromising its integrity, database and preventing conflicts which can be seen extensively while data transfer from multiple accounts to one at the same time.

. Data Independence: Logical data independence is achieved by DBMS, which hides from users the specifics of the physical storage. The application programs are unaffected by modifications to the overall database structure and schema making it highly independent.

. Query Language: A database management system (DBMS) may be used to retrieve, manipulate, and manage data by means of a query language, most commonly SQL (Structured Query Language).

. Transaction management: carries out transaction processing to guarantee that a sequence of actions is carried out completely or reversed in the event of a mistake, hence successfully preserving integrity and consistency of the system in remaining coherent. **. Scalability**: DBMSs are made to be scalable, meaning they can handle increasing user loads and data quantities without seeing a noticeable drop in performance. **. Data Dictionary Management**: Assists in schema creation, data description, and query optimization by keeping an organized collection of database metadata in a centralized location called a data dictionary.

. ACID qualities: (Atomicity, Consistency, Isolation, Durability) in order to guarantee the accuracy and dependability of database transactions proving its efficiency and consistency. **. Query optimization**: uses the engine embedded in the backend to facilitate strategies like indexing and query plan optimization to optimize query execution and improve speed.

4.3. DATA MODEL

A conceptual depiction of the data and its connections inside the database is called a data model. It provides an abstract perspective that enables database managers and developers to work with the data, acting as a guide for building and organizing the database. **Relational Data Model**: Specifies relationships between tables by representing data as tables (relations) with rows and columns.

Entity-Relationship (ER) Model: This model helps to visualise the database's structure by providing a graphical description of entities (objects) and their relationships. **Object-Oriented Data Model**: Uses the concepts of object-oriented programming to create databases by representing data as objects with properties and methods.

Hierarchical data model: Older database systems frequently employ the hierarchical data model, which organises data in a tree-like structure with parent-child connections. **Network Data Model**: Increases flexibility in describing complicated data structures by including numerous parent-child relationships, extending the hierarchical model.

4.4. 3- SCHEMA ARCHITECTURE

Database Management System (DBMS) three-schema design divides the database system into three levels: external schema (user level), conceptual schema (logical level), and internal schema (physical level). This design promotes data independence and makes it simpler to make changes to the system without impacting all levels at once. It does this by allowing a clear separation between the logical organisation of the data, how the data is seen by end users, and the specifics of physical storage.

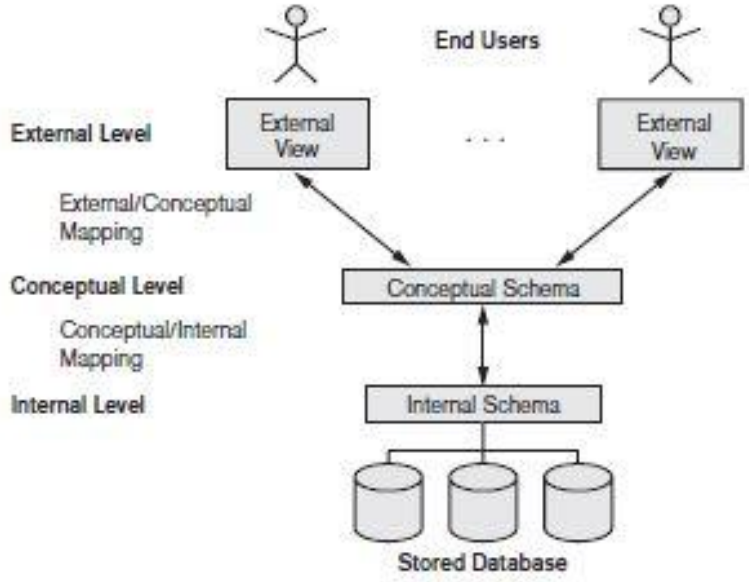


Fig: 28. Three Schema Architecture of DBMS

4.5. COMPONENT MODULES

Component modules are specialised components inside a Database Management System (DBMS) that manage certain functions for the system. These modules, which are in charge of performing file operations, processing queries, controlling concurrent access, guaranteeing transaction integrity, and managing metadata, are the Data Dictionary, Query Processor, File Manager, Transaction Manager, and Concurrency Control Manager. Together, these components enable effective user interactions and data administration inside a database management system.

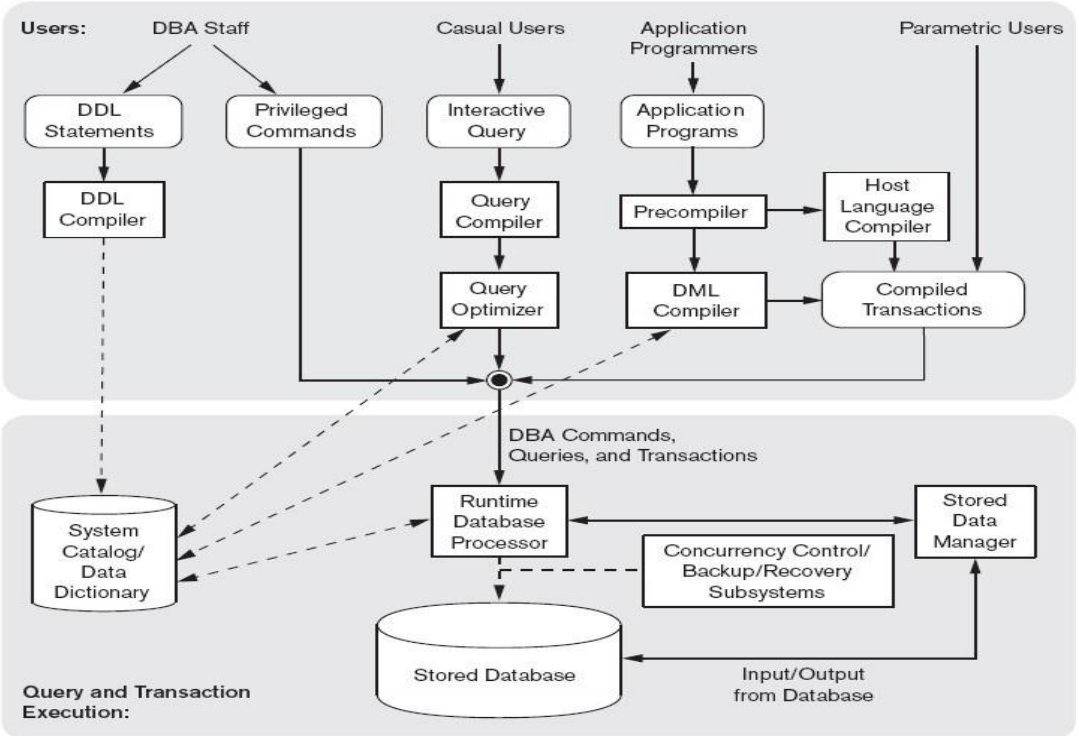


Fig: 29. Component Modules of DBMS

4.6. ENTITY- RELATIONSHIP MODEL

Database Management Systems (DBMS) employ the Entity-Relationship (E-R) model as a conceptual framework to design and portray the connections between things in a database. Relationships show the links between entities, which are actual things or ideas. The model creates a visual depiction of the database structure by using entities, properties, and relationships. Relationships are represented as diamond shapes, characteristics as ovals, and entities as rectangles. By offering an understandable and straightforward representation of the connections and interdependencies among different entities, the E-R model helps with database design by promoting efficient data organisation and guaranteeing the accuracy of the database architecture.

4.7. RELATIONAL SCHEMA

(i) Tables: Every table in the schema is associated with an entity, such as a client, a product, or a worker.

(ii) Attributes: Tables are made up of attributes, which stand for the qualities or traits of the entities. In every table, the columns or domains are defined by attributes.

(iii) Keys: Relational schemas create linkages between tables using keys. Foreign keys create relationships across tables, whereas primary keys uniquely identify records within a table.

(iv) Entities and Relationships: Tables represent entities, which are actual things or concepts. Relationships define the connections and interactions between things.

(v) Logical Design: By abstracting away specifics about physical storage, the relational schema offers a logical design for the database. Regardless of how the data is saved, its main focus is on how it is arranged and connected.

(vi) Normalization: In order to reduce data redundancy and guarantee data integrity, the schema complies with normalization principles, which aids in effective maintenance and storage.

CHAPTER 5

FUNDAMENTALS OF SQL

5.1. SQL OVERVIEW

Relational database management and manipulation are done with the help of a domain-specific language called Structured Query Language (SQL). SQL was first introduced in the 1970s and offers a standardised way to interface with databases, making it easier to create, retrieve, edit, and delete data. Because of the declarative syntax used by the language, users may communicate their objectives without having to detail the precise methods. Data Definition Language (DDL) is used to define and manage database structures; Data Manipulation Language (DML) is used to interface with data; and Data govern Language (DCL) is used to govern rights and access. The SQL commands SELECT, UPDATE, DELETE, and INSERT are frequently used to query data, add new records, and alter existing data. Because of its extensive use and adaptability, SQL is an essential tool for database administration.

5.2. SQL COMMANDS

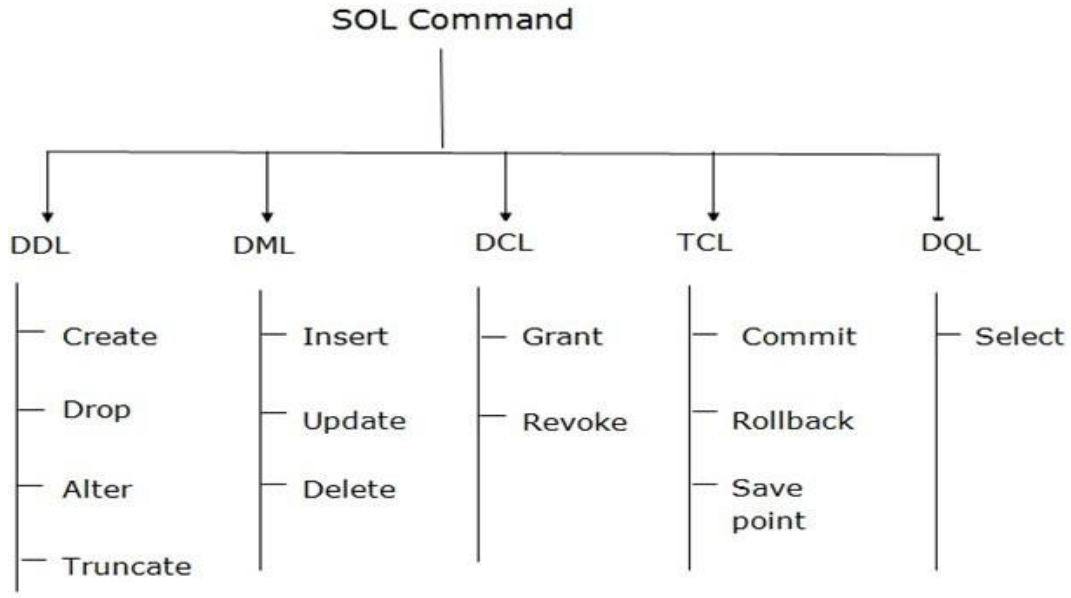


Fig: 30. Overview of MySQL Commands

5.3. DDL (Data Definition Language)

A subset of SQL (Structured Query Language) called DDL (Data Definition Language) is used to define and manage a database's structure. Database administrators may add, edit, and remove database objects including tables, indexes, and views using DDL commands. Some important DDL commands can be listed out as follows: **CREATE**: used in the creation of new tables, indexes, and views in databases. **ALTER**: alters the structure of already-existing database objects, changing a table's columns or adding new ones or even dropping unwanted columns from the table. **DROP**: removes already-existing database objects, such as indexes and tables. **TRUNCATE**: eliminates every record from a table while keeping the table structure intact for further usage.

5.4. DML (Data Manipulation Language)

A subset of SQL (Structured Query Language) called DML (Data Manipulation Language) is used to interact with and manipulate data inside of databases. Users may obtain, insert, update, and remove data from database tables using DML instructions. Important DML instructions consists:

SELECT: retrieves information from one or more tables according to predetermined standards.

INSERT: expands a table by adding more rows or records.

UPDATE: alters data that already exists in a table according to certain criteria.

DELETE: eliminates entries from a table according to predetermined standards.

5.5. DCL (Data Control Language)

A subset of SQL (Structured Query Language) called DCL (Data regulate Language) is used to manage and regulate data access within databases. The main purposes of DCL commands are to govern user privileges, set permissions, and oversee database security. There are two primary DCL commands that plays with this logic, they are: **GRANT**: Gives users or roles particular rights or privileges that enable them to carry out particular tasks on designated database objects, usually implemented as user creation.

REVOKE: limits access to particular database items for users or roles by removing previously given rights or permissions.

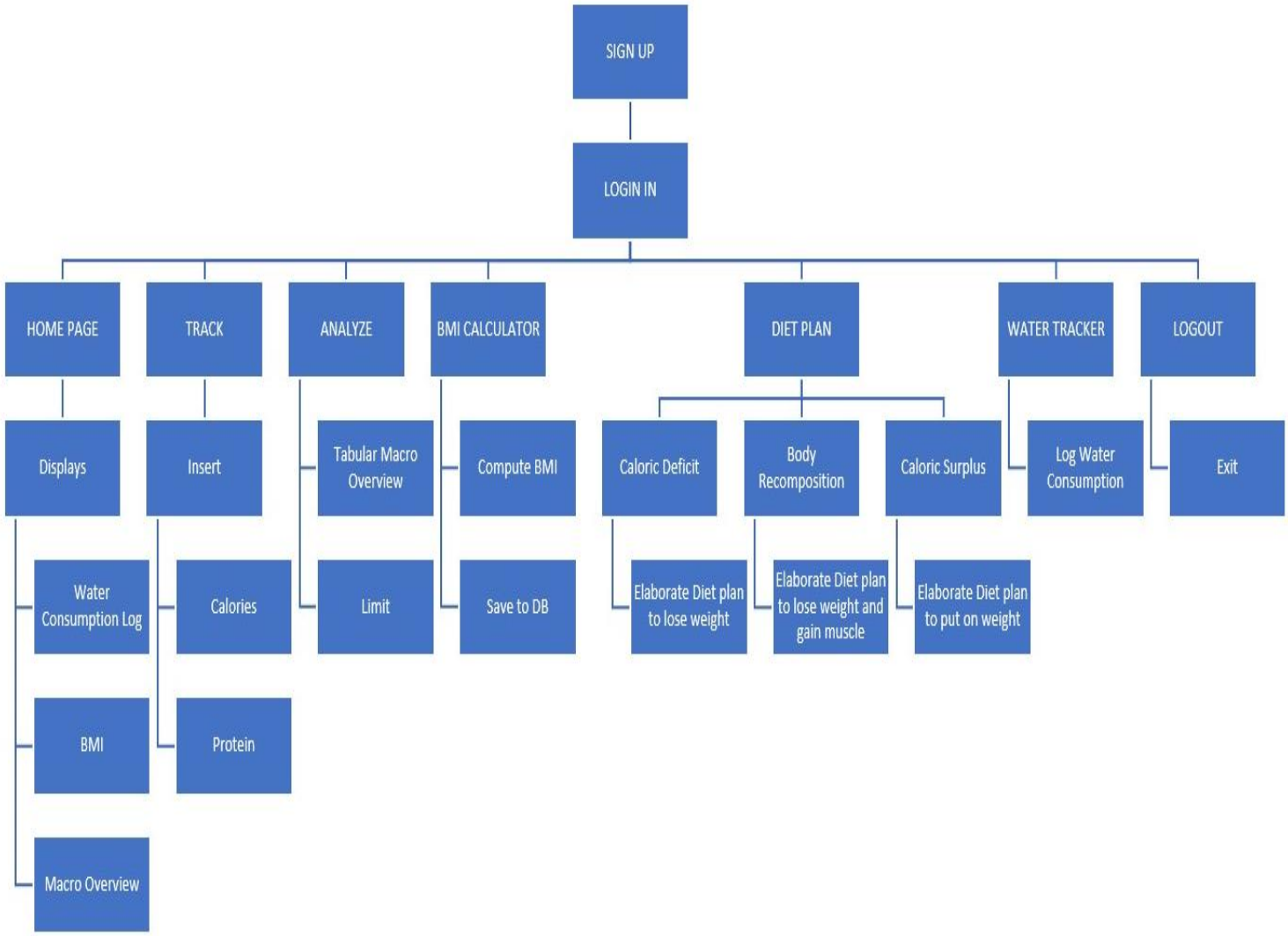
5.6. TCL (Transmission Control Language)

TCL is a subset of SQL (Structured Query Language) that is used mostly for database transaction management. Sequences of one or more SQL statements carried out as a single work unit are called transactions. By regulating the course and result of transactions, TCL commands serve to guarantee the consistency, integrity, and dependability of the database proving itself to be a reliable source for a database. **COMMIT**: To permanently apply the modifications made during the current transaction. The modifications are irreversible once a COMMIT is sent out and cannot be undone. It represents the transaction's successful conclusion.

ROLLBACK: To reverse the modifications performed during the current transaction, use the ROLLBACK command. The ROLLBACK command is used to return the database to its initial state at the beginning of the transaction in the event that an error occurs or if it becomes necessary to discard the modifications for whatever reason. **SAVEPOINT**: You can set a point in a transaction to which you may subsequently roll back by using the SAVEPOINT command. It lets you set a marker inside a transaction so you may roll back to that precise moment without having to reverse the entire transaction if necessary.

CHAPTER 6

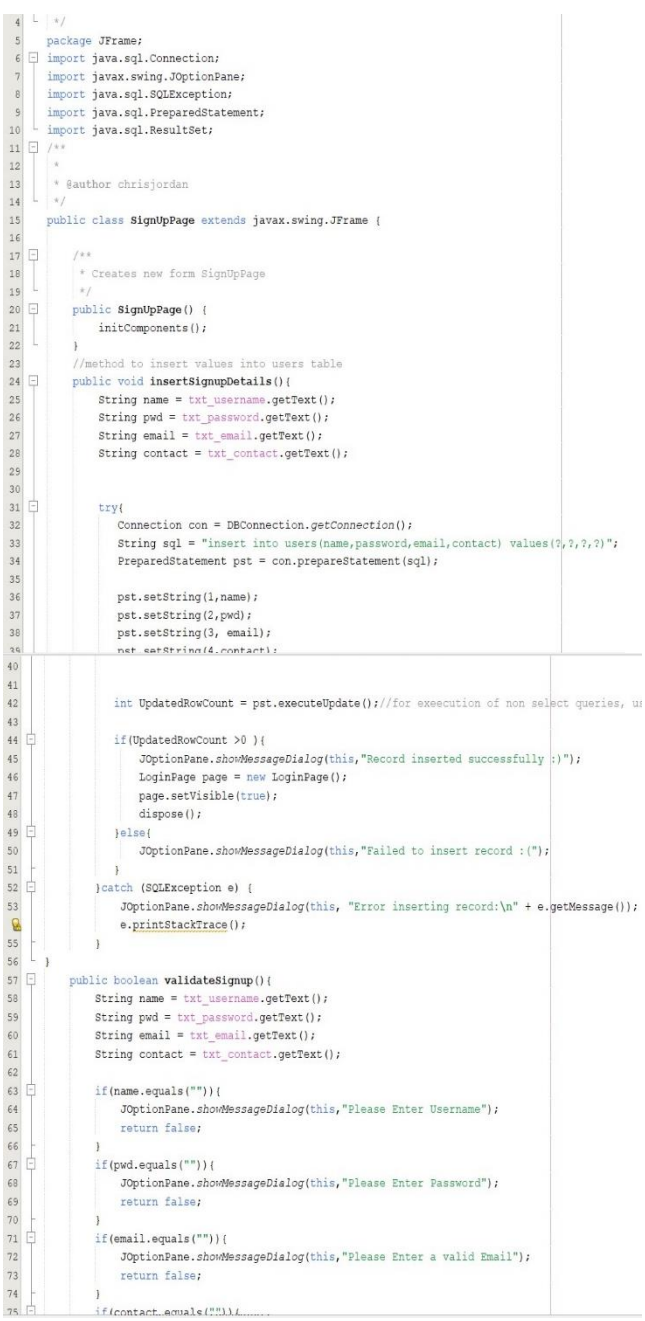
DESIGN AND ARCHITECTURE



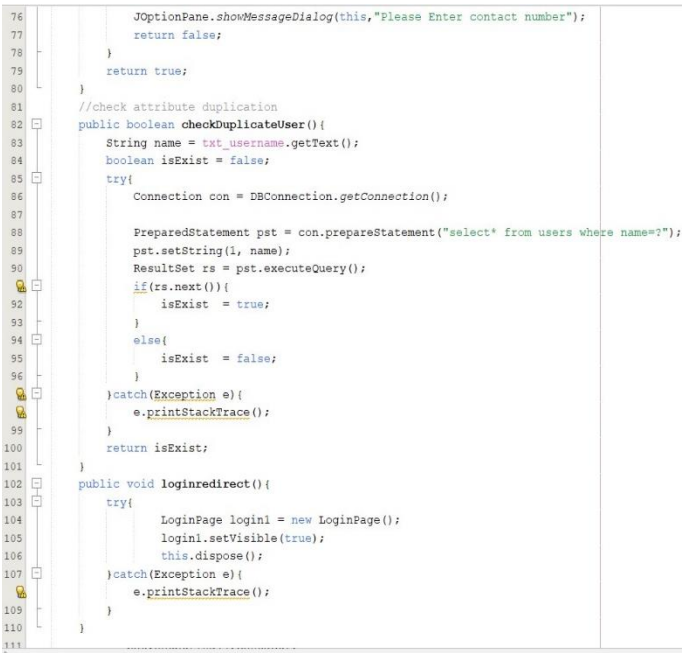
CHAPTER 7

7. IMPLEMENTATION

7.1 SignUpPage.java



```
*/
package JFrame;
import java.sql.Connection;
import javax.swing.JOptionPane;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
/**
 *
 * @author chrisjordan
 */
public class SignUpPage extends javax.swing.JFrame {

    /**
     * Creates new form SignUpPage
     */
    public SignUpPage() {
        initComponents();
    }
    //method to insert values into users table
    public void insertSignupDetails(){
        String name = txt_username.getText();
        String pwd = txt_password.getText();
        String email = txt_email.getText();
        String contact = txt_contact.getText();


        try{
           Connection con = DBConnection.getConnection();
           String sql = "insert into users(name,password,email,contact) values(?,?,?,?)";
           PreparedStatement pst = con.prepareStatement(sql);

           pst.setString(1,name);
           pst.setString(2,pwd);
           pst.setString(3, email);
           pst.setString(4,contact);


           int UpdatedRowCount = pst.executeUpdate();//for exeecution of non select queries, u

           if(UpdatedRowCount >0 ){
               JOptionPane.showMessageDialog(this,"Record inserted successfully :)");
               LoginPage page = new LoginPage();
               page.setVisible(true);
               dispose();
           }else{
               JOptionPane.showMessageDialog(this,"Failed to insert record :(");
           }
        }catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error inserting record:\n" + e.getMessage());
            e.printStackTrace();
        }
}
    public boolean validateSignup(){
        String name = txt_username.getText();
        String pwd = txt_password.getText();
        String email = txt_email.getText();
        String contact = txt_contact.getText();

        if(name.equals("")){
            JOptionPane.showMessageDialog(this,"Please Enter Username");
            return false;
        }
        if(pwd.equals("")){
            JOptionPane.showMessageDialog(this,"Please Enter Password");
            return false;
        }
        if(email.equals("")){
            JOptionPane.showMessageDialog(this,"Please Enter a valid Email");
            return false;
        }
        if(contact.equals("")){.....
```



```
            JOptionPane.showMessageDialog(this,"Please Enter contact number");
            return false;
        }
        return true;
    }
    //check attribute duplication
    public boolean checkDuplicateUser(){
        String name = txt_username.getText();
        boolean isExist = false;
        try{
            Connection con = DBConnection.getConnection();

            PreparedStatement pst = con.prepareStatement("select* from users where name=?");
            pst.setString(1, name);
            ResultSet rs = pst.executeQuery();
            if(rs.next()){
                isExist  = true;
            }
            else{
                isExist  = false;
            }
        }catch(Exception e){
            e.printStackTrace();
        }
        return isExist;
    }
    public void loginredirect(){
        try{
                LoginPage login1 = new LoginPage();
                login1.setVisible(true);
                this.dispose();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
```

```
    Generated Code

    private void txt_usernameActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txt_passwordActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txt_emailActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        loginredirect();

    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if(validateSignup() == true){
            if(checkDuplicateUser()==false){
                insertSignupDetails();
            } else{
                JOptionPane.showMessageDialog(this,"User already exists");
            }

        }
    }

    private void txt_usernameFocusLost(java.awt.event.FocusEvent evt) {
        // TODO add your handling code here:
        if (checkDuplicateUser()==true){
            JOptionPane.showMessageDialog(this,"Username already exists");
```

```
330         }
331     }
332
334     private void txt_contactActionPerformed(java.awt.event.ActionEvent evt) {
335         // TODO add your handling code here:
336     }
337
338     /**
339      * @param args the command line arguments
340      */
341     public static void main(String args[]) {
342         /* Set the Nimbus look and feel */
343         Look and feel setting code (optional)
364
366         /* Create and display the form */
368         java.awt.EventQueue.invokeLater(new Runnable() {
369             @Override
370             public void run() {
371                 new SignUpPage().setVisible(true);
372             }
373         });
374     }
375
376     // Variables declaration - do not modify
377     private javax.swing.JButton jButton1;
378     private javax.swing.JButton jButton2;
379     private javax.swing.JLabel jLabel1;
380     private javax.swing.JLabel jLabel10;
381     private javax.swing.JLabel jLabel11;
382     private javax.swing.JLabel jLabel12;
383     private javax.swing.JLabel jLabel14;
384     private javax.swing.JLabel jLabel2;
385     private javax.swing.JLabel jLabel3;
```

```
366             java.awt.EventQueue.invokeLater(new Runnable() {
368                 @Override
369                 public void run() {
370                     new SignUpPage().setVisible(true);
371                 }
372             });
373         }
374
375         // Variables declaration - do not modify
376         private javax.swing.JButton jButton1;
377         private javax.swing.JButton jButton2;
378         private javax.swing.JLabel jLabel1;
379         private javax.swing.JLabel jLabel10;
380         private javax.swing.JLabel jLabel11;
381         private javax.swing.JLabel jLabel12;
382         private javax.swing.JLabel jLabel14;
383         private javax.swing.JLabel jLabel2;
384         private javax.swing.JLabel jLabel3;
385         private javax.swing.JLabel jLabel4;
386         private javax.swing.JLabel jLabel5;
387         private javax.swing.JLabel jLabel6;
388         private javax.swing.JLabel jLabel7;
389         private javax.swing.JLabel jLabel8;
390         private javax.swing.JLabel jLabel9;
391         private javax.swing.JPanel jPanel1;
392         private javax.swing.JPanel jPanel2;
393         private javax.swing.JTextField txt_contact;
394         private javax.swing.JTextField txt_email;
395         private javax.swing.JTextField txt_password;
396         private javax.swing.JTextField txt_username;
397         // End of variables declaration
}
```

7.2. LoginInPage.java

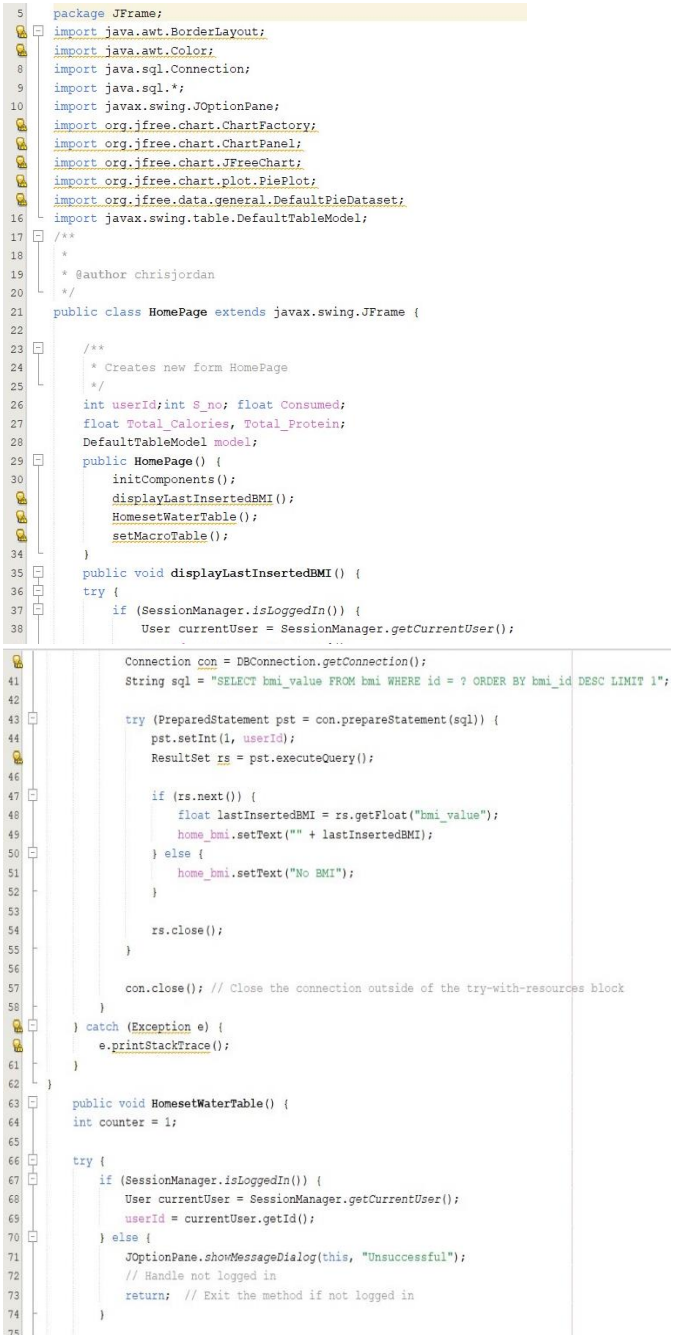
```java
package JFrame;
import java.sql.Connection;
import javax.swing.JOptionPane;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
/**
 *
 * @author chrisjordan
 */
public class LoginPage extends javax.swing.JFrame {

    /**
     * Creates new form SignUpPage
     */
    public LoginPage() {
        initComponents();
    }

    //login validation
    public boolean validateLogin(){
        String name=login_username.getText();
        String pwd=login_password.getText();

        if (name.equals("")){
            JOptionPane.showMessageDialog(this, "Please Enter Username");
            return false;
        }
        if (pwd.equals("")){
            JOptionPane.showMessageDialog(this, "Please Enter your password");
            return false;
        }
        return true;
    }
```

```java
    //Login validation
    public void Login(){
        String name=login_username.getText();
        String pwd=login_password.getText();
        try{
            Connection con = DBConnection.getConnection();
            PreparedStatement pst = con.prepareStatement("select id from users where name=? and password=?");
            pst.setString(1,name);
            pst.setString(2, pwd);
            ResultSet rs = pst.executeQuery();
            //redirection
            if(rs.next()){
                int userId = rs.getInt("id");

                // Create a User object
                User user = new User();
                user.setId(userId);

                // Set the logged-in user in the session
                SessionManager.setCurrentUser(user);
                /*int userId = rs.getInt("id");

                // Set the logged-in user ID in the session
                SessionManager.setLoggedInUserId(userId);*/
                JOptionPane.showMessageDialog(this,"Login successful");
                HomePage home = new HomePage();
                home.setVisible(true);
                this.dispose();
            }else{
                JOptionPane.showMessageDialog(this,"Incorrect Credentials");
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
```

```java
        try{
                SignUpPage signup1 = new SignUpPage();
                signup1.setVisible(true);
                this.dispose();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void login_usernameActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void login_passwordActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if(validateLogin()){
            Login();
        }
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        signupredirect();
```

```java
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                new LoginPage().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JTextField login_password;
    private javax.swing.JTextField login_username;
    // End of variables declaration
}
```

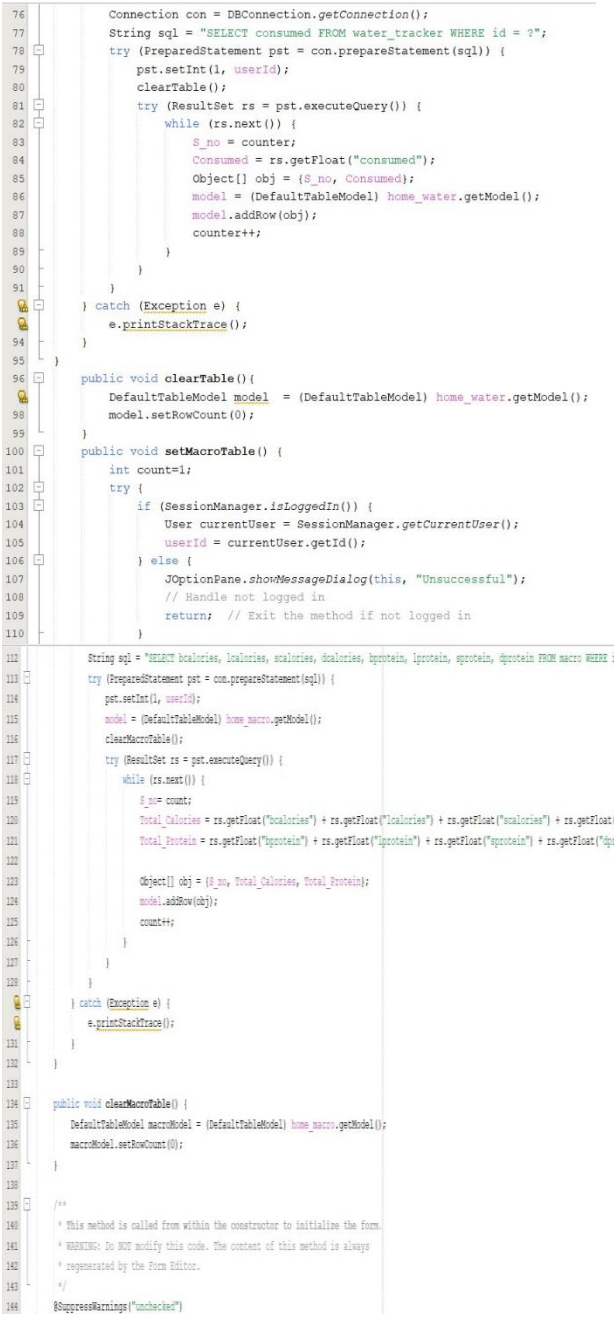
7.3. HomePage.java



```
package JFrame;
import java.awt.BorderLayout;
import java.awt.Color;
import java.sql.Connection;
import java.sql.*;
import javax.swing.JOptionPane;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PiePlot;
import org.jfree.data.general.DefaultPieDataset;
import javax.swing.table.DefaultTableModel;
/**
 *
 * @author chrisjordan
 */
public class HomePage extends javax.swing.JFrame {

    /**
     * Creates new form HomePage
     */
    int userId;int S_no; float Consumed;
    float Total_Calories, Total_Protein;
    DefaultTableModel model;
    public HomePage() {
        initComponents();
        displayLastInsertedBMI();
        HomesetWaterTable();
        setMacroTable();
    }
    public void displayLastInsertedBMI() {
    try {
        if (SessionManager.isLoggedIn()) {
            User currentUser = SessionManager.getCurrentUser();
```



```
            Connection con = DBConnection.getConnection();
            String sql = "SELECT bmi_value FROM bmi WHERE id = ? ORDER BY bmi_id DESC LIMIT 1";

            try (PreparedStatement pst = con.prepareStatement(sql)) {
                pst.setInt(1, userId);
                ResultSet rs = pst.executeQuery();

                if (rs.next()) {
                    float lastInsertedBMI = rs.getFloat("bmi_value");
                    home_bmi.setText("" + lastInsertedBMI);
                } else {
                    home_bmi.setText("No BMI");
                }

                rs.close();
            }

            con.close(); // Close the connection outside of the try-with-resources block
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
    public void HomesetWaterTable() {
    int counter = 1;

    try {
        if (SessionManager.isLoggedIn()) {
            User currentUser = SessionManager.getCurrentUser();
            userId = currentUser.getId();
        } else {
            JOptionPane.showMessageDialog(this, "Unsuccessful");
            // Handle not logged in
            return;  // Exit the method if not logged in
        }
```



```
        Connection con = DBConnection.getConnection();
        String sql = "SELECT consumed FROM water_tracker WHERE id = ?";
        try (PreparedStatement pst = con.prepareStatement(sql)) {
            pst.setInt(1, userId);
            clearTable();
            try (ResultSet rs = pst.executeQuery()) {
                while (rs.next()) {
                    S_no = counter;
                    Consumed = rs.getFloat("consumed");
                    Object[] obj = {S_no, Consumed};
                    model = (DefaultTableModel) home_water.getModel();
                    model.addRow(obj);
                    counter++;
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
    public void clearTable(){
        DefaultTableModel model  = (DefaultTableModel) home_water.getModel();
        model.setRowCount(0);
    }
    public void setMacroTable() {
        int count=1;
        try {
            if (SessionManager.isLoggedIn()) {
                User currentUser = SessionManager.getCurrentUser();
                userId = currentUser.getId();
            } else {
                JOptionPane.showMessageDialog(this, "Unsuccessful");
                // Handle not logged in
                return;  // Exit the method if not logged in
            }
```



```
        String sql = "SELECT bcalories, lcalories, scalories, dcalories, bprotein, lprotein, sprotein, dprotein FROM macro WHERE id = ?";
        try (PreparedStatement pst = con.prepareStatement(sql)) {
            pst.setInt(1, userId);
            model = (DefaultTableModel) home_macro.getModel();
            clearMacroTable();
            try (ResultSet rs = pst.executeQuery()) {
                while (rs.next()) {
                    S_no= count;
                    Total_Calories = rs.getFloat("bcalories") + rs.getFloat("lcalories") + rs.getFloat("scalories") + rs.getFloat("dcalories");
                    Total_Protein = rs.getFloat("bprotein") + rs.getFloat("lprotein") + rs.getFloat("sprotein") + rs.getFloat("dprotein");

                    Object[] obj = {S_no, Total_Calories, Total_Protein};
                    model.addRow(obj);
                    count++;
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void clearMacroTable() {
    DefaultTableModel macroModel = (DefaultTableModel) home_macro.getModel();
    macroModel.setRowCount(0);
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
```

```
    public void trackredirect(){
        try{
                TrackPage track1 = new TrackPage();
                track1.setVisible(true);
                this.dispose();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    private void TracklabelMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        trackredirect();
    }

    private void h_analyzeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void h_bmiMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void h_dietMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void h_waterMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void jPanel12MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    /**...3 lines */
    public static void main(String args[]) {...31 lines }


    // Variables declaration - do not modify
    private javax.swing.JPanel Tracklabel;
    private javax.swing.JPanel h_analyze;
    private javax.swing.JPanel h_bmi;
    private javax.swing.JPanel h_diet;
    private javax.swing.JPanel h_water;
    private javax.swing.JTextArea home_bmi;
    private rojeru_san.complementos.RSTableMetro home_macro;
    private rojeru_san.complementos.RSTableMetro home_water;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel12;
    private javax.swing.JPanel jPanel13;
    private javax.swing.JPanel jPanel14;
    private javax.swing.JPanel jPanel17;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    // End of variables declaration
}
```

7.4. TrackPage.java

```
package JFrame;
import java.sql.Connection;
import java.sql.PreparedStatement;
import javax.swing.JOptionPane;
/**
 *
 * @author chrisjordan
 */
public class TrackPage extends javax.swing.JFrame {

    /**
     * Creates new form TrackPage
     */
    long breakfast_calories,breakfast_protein,lunch_calories,lunch_protein, snacks_calories,snacks_protein,dinner_calories,dinner_protein;
    int userId;
    public TrackPage() {
        initComponents();
    }
    public boolean inserttrack(){
        boolean isAdded = false;
        breakfast_calories= Long.parseLong(b_calories.getText());
        breakfast_protein= Long.parseLong(b_protein.getText());
        lunch_calories= Long.parseLong(l_calories.getText());
        lunch_protein= Long.parseLong(l_protein.getText());
        snacks_calories= Long.parseLong(s_calories.getText());
        snacks_protein= Long.parseLong(s_protein.getText());
        dinner_calories= Long.parseLong(d_calories.getText());
        dinner_protein= Long.parseLong(d_protein.getText());

        try{
            if (SessionManager.isLoggedIn()) {
                User currentUser = SessionManager.getCurrentUser();
                userId = currentUser.getId();
                } else {
                JOptionPane.showMessageDialog(this, "Unsuccessful");
            }
            Connection con = DBConnection.getConnection();
            String sql = "insert into macro(id, bcalories,bprotein,lcalories,lprotein,scalories,sprotein,dcalories,dprotein) values(?,?,?,?,?,?,?,?,?)";
            //String sql = "update macro set bcalories=?, bprotein=?, lcalories=?, lprotein=?, scalories=?, sprotein=?, dcalories=?, dprotein=? where id
            PreparedStatement pst = con.prepareStatement(sql);
            pst.setInt(1,userId);
            pst.setLong(2,breakfast_calories);
            pst.setLong(3,breakfast_protein);
            pst.setLong(4,lunch_calories);
            pst.setLong(5,lunch_protein);
            pst.setLong(6,snacks_calories);
            pst.setLong(7,snacks_protein);
            pst.setLong(8,dinner_calories);
            pst.setLong(9,dinner_protein);

            int rowCount = pst.executeUpdate();
            if (rowCount > 0) {
                isAdded= true;
            } else {
                isAdded= false;
            }

    } catch (Exception e) {
        e.printStackTrace(System.out);
        isAdded = false;

    }
        return isAdded;

    public boolean updatetrack(){
        boolean isUpdated = false;
        breakfast_calories= Integer.parseInt(b_calories.getText());
        breakfast_protein= Integer.parseInt(b_protein.getText());
        lunch_calories= Integer.parseInt(l_calories.getText());
        lunch_protein= Integer.parseInt(l_protein.getText());
        snacks_protein= Integer.parseInt(s_protein.getText());
        dinner_calories= Integer.parseInt(d_calories.getText());
        dinner_protein= Integer.parseInt(d_protein.getText());

    try{
        if (SessionManager.isLoggedIn()) {
            User currentUser = SessionManager.getCurrentUser();
            userId = currentUser.getId();
            } else {
            JOptionPane.showMessageDialog(this, "Unsuccessful");
        }
        Connection con = DBConnection.getConnection();
        //String sql = "insert into nutrition(bcalories,bprotein,lcalories,lprotein,scalories,sprotein,dcalories,dprotein) values(?,?,?,?,?,?,?,?,?)";
        String sql = "update nutrition set bcalories=?, bprotein=?, lcalories=?, lprotein=?, scalories=?, sprotein=?, dcalories=?, dprotein=? where id = ?)";
        PreparedStatement pst = con.prepareStatement(sql);
        pst.setLong(1,breakfast_calories);
        pst.setLong(2,breakfast_protein);
        pst.setLong(3,lunch_calories);
        pst.setLong(4,lunch_protein);
        pst.setLong(5,snacks_calories);
        pst.setLong(6,snacks_protein);
        pst.setLong(7,dinner_calories);
        pst.setLong(8,dinner_protein);
        pst.setInt(9,userId);

        int rowCount = pst.executeUpdate();
        if (rowCount > 0) {
            isUpdated= true;
        } else {
            isUpdated= false;
        }

    catch (Exception e) {
      e.printStackTrace(System.out);
      isUpdated = false;
```

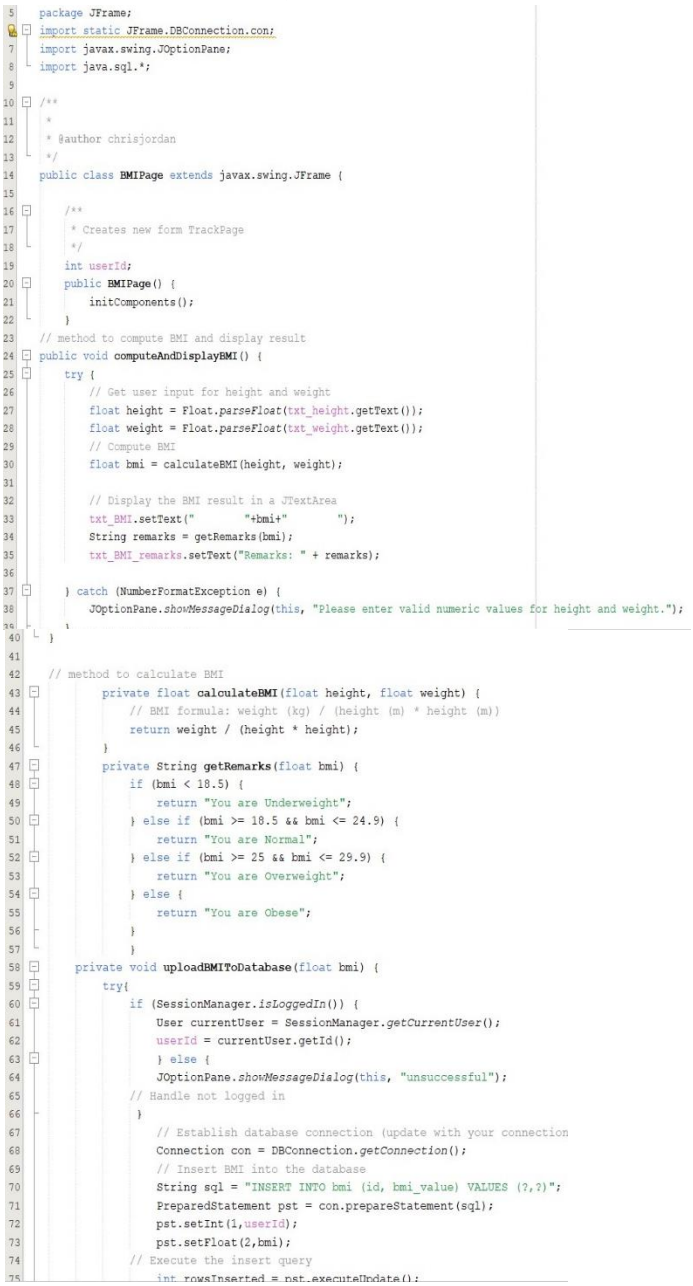
```
    public void homeredirect(){
        try{
                HomePage home1 = new HomePage();
                home1.setVisible(true);
                this.dispose();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    //update
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void t_update1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if (updatetrack() == true){
            JOptionPane.showMessageDialog(this,"Update successful!!!");
        }
        else{
            JOptionPane.showMessageDialog(this,"Update unsuccessful :(");
        }
    }

    private void t_insert1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if (inserttrack() == true){
            JOptionPane.showMessageDialog(this,"Insert successful!!!");
        }
        else{
            JOptionPane.showMessageDialog(this,"Insert unsuccessful :(");
        }
    }

    private void TracklabelMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
    }

    private void t_analyzeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void jPanel5MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void jPanel6MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void jPanel7MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void jPanel9MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void jPanel12MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    /**...3 lines */
    public static void main(String args[]) {...31 lines }

    // Variables declaration - do not modify
    private javax.swing.JPanel Tracklabel;
    private javax.swing.JTextField b_calories;
    private javax.swing.JTextField b_protein;
    private javax.swing.JTextField d_calories;
    private javax.swing.JTextField d_protein;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel30;
    private javax.swing.JLabel jLabel31;
    private javax.swing.JLabel jLabel32;
    private javax.swing.JLabel jLabel33;
    private javax.swing.JLabel jLabel34;
    private javax.swing.JLabel jLabel35;
    private javax.swing.JLabel jLabel36;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel12;
    private javax.swing.JPanel jPanel13;
    private javax.swing.JPanel jPanel14;
    private javax.swing.JPanel jPanel15;
    private javax.swing.JPanel jPanel16;
    private javax.swing.JPanel jPanel17;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    private javax.swing.JPanel jPanel7;
    private javax.swing.JPanel jPanel9;
    private javax.swing.JTextField l_calories;
    private javax.swing.JTextField l_protein;
    private javax.swing.JTextField s_calories;
    private javax.swing.JTextField s_protein;
    private javax.swing.JPanel t_analyze;
    private javax.swing.JButton t_insert1;
    private javax.swing.JButton t_update1;
    // End of variables declaration
}
```

7.5. AnalyzePage.java

```
package JFrame;
import static JFrame.DBConnection.con;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import java.sql.*;
/**
 *
 * @author chrisjordan
 */
public class AnalyzePage extends javax.swing.JFrame {

    /**
     * Creates new form TrackPage
     */
    long breakfast_calories,breakfast_protein,lunch_calories,lunch_protein, snacks_calories,snacks_protein,dinner_calories,dinner_protein;
    int userId,counter,s_no;
    DefaultTableModel model;
    public AnalyzePage() {
        initComponents();
        setAnalyzeTable();
        calculateAndDisplayTotal();
    }
    public void setAnalyzeTable(){
        int counter = 1;
        try{
            if (SessionManager.isLoggedIn()) {
               User currentUser = SessionManager.getCurrentUser();
               userId = currentUser.getId();
               } else {
               JOptionPane.showMessageDialog(this, "unsuccessful");
            // Handle not logged in
            }
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/test1","root","christ123");
            //Statement st = con.createStatement();
            //ResultSet rs = st.executeQuery("select bcalories,bprotein, lcalories, lprotein, scalories, sprotein, dcalories, dprotein from
            //pst.setInt(1, userId);
            String sql = "SELECT bcalories, bprotein, lcalories, lprotein, scalories, sprotein, dcalories, dprotein FROM macro WHERE id=?";
            PreparedStatement pst = con.prepareStatement(sql) ;
            pst.setInt(1, userId);
            clearTable();
            ResultSet rs = pst.executeQuery();
            while (rs.next()){
                s_no = counter;
                breakfast_calories = rs.getLong("bcalories");
                breakfast_protein = rs.getLong("bprotein");
                lunch_calories = rs.getLong("lcalories");
                lunch_protein  = rs.getLong("lprotein");
                snacks_calories = rs.getLong("scalories");
                snacks_protein = rs.getLong("sprotein");
                dinner_calories = rs.getLong("dcalories");
                dinner_protein =rs.getLong("dprotein");
                Object[] obj = {s_no, breakfast_calories,breakfast_protein,lunch_calories,lunch_protein,snacks_calories,
                    snacks_protein,dinner_calories,dinner_protein };
                model = (DefaultTableModel)tbl_analyze.getModel();
                model.addRow(obj);
                counter++;
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    public void clearTable(){
        DefaultTableModel model  = (DefaultTableModel)tbl_analyze.getModel();
        model.setRowCount(0);
    }
    private void calculateAndDisplayTotal() {
    }
    private void calculateAndDisplayTotal() {
    // Iterate through the table rows
    for (int i = 0; i < tbl_analyze.getRowCount(); i++) {
        long sumCalories = 0;
        long sumProtein = 0;

        // Calculate the sum of bcalories, lcalories, scalories, and dcalories for each row
        sumCalories += (long) tbl_analyze.getValueAt(i, 1); // breakfast_calories
        sumCalories += (long) tbl_analyze.getValueAt(i, 3); // lunch_calories
        sumCalories += (long) tbl_analyze.getValueAt(i, 5); // snacks_calories
        sumCalories += (long) tbl_analyze.getValueAt(i, 7); // dinner_calories

        // Calculate the sum of bprotein, lprotein, sprotein, and dprotein for each row
        sumProtein += (long) tbl_analyze.getValueAt(i, 2); // breakfast_protein
        sumProtein += (long) tbl_analyze.getValueAt(i, 4); // lunch_protein
        sumProtein += (long) tbl_analyze.getValueAt(i, 6); // snacks_protein
        sumProtein += (long) tbl_analyze.getValueAt(i, 8); // dinner_protein

        // Set the total protein in the appropriate column
        tbl_analyze.setValueAt(sumCalories, i, 9);
        tbl_analyze.setValueAt(sumProtein, i, 10); // Update "total_protein" column
    }
}

    private void compareAndSetRemarks() {
        // Fetch the value from the "set limits" field
        long dailyLimit = Long.parseLong(s_limit.getText());

        // Iterate through the table rows
        for (int i = 0; i < tbl_analyze.getRowCount(); i++) {
            long sumCalories = 0;
            long sumProtein = 0;

            // Calculate the sum of bcalories, lcalories, scalories, and dcalories for each row
```

```
            sumCalories += (long) tbl_analyze.getValueAt(i, 3); // lunch_calories
            sumCalories += (long) tbl_analyze.getValueAt(i, 5); // snacks_calories
            sumCalories += (long) tbl_analyze.getValueAt(i, 7); // dinner_calories

            // Calculate the sum of bprotein, lprotein, sprotein, and dprotein for each row
            sumProtein += (long) tbl_analyze.getValueAt(i, 2); // breakfast_protein
            sumProtein += (long) tbl_analyze.getValueAt(i, 4); // lunch_protein
            sumProtein += (long) tbl_analyze.getValueAt(i, 6); // snacks_protein
            sumProtein += (long) tbl_analyze.getValueAt(i, 8); // dinner_protein

            // Compare the calculated sum with the daily limit
            if (sumCalories < dailyLimit) {
                tbl_analyze.setValueAt("SubOptimal", i, 11); // Update "remarks" column to "Less"
            } else if (sumCalories == dailyLimit) {
                tbl_analyze.setValueAt("Optimal", i, 11); // Update "remarks" column to "Perfect"
            } else {
                tbl_analyze.setValueAt("Excessive", i, 11); // Update "remarks" column to "Exceed"
            }

            // Set the total protein in the appropriate column
            tbl_analyze.setValueAt(sumCalories, i, 9);
            tbl_analyze.setValueAt(sumProtein, i, 10); // Update "total_protein" column
        }
    }

    public void homeredirect(){
        try{
                HomePage home1 = new HomePage();
                home1.setVisible(true);
                this.dispose();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
    //update
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void a_limitActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void a_backActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        HomePage h = new HomePage();
        h.setVisible(true);
        this.dispose();
    }

    private void analyze_buttActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        compareAndSetRemarks();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new AnalyzePage().setVisible(true);
            }
        });
    }


    // Variables declaration - do not modify
    private javax.swing.JButton a_back;
    private javax.swing.JTextField a_limit;
    private javax.swing.JButton analyze_butt;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel13;
    private javax.swing.JPanel jPanel14;
    private javax.swing.JPanel jPanel15;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JScrollPane jScrollPane1;
    private rojeru_san.complementos.RSTableMetro tbl_analyze;
    // End of variables declaration
}
```

7.6. BMIPage.java



```
package JFrame;
import static JFrame.DBConnection.con;
import javax.swing.JOptionPane;
import java.sql.*;

/**
 *
 * @author chrisjordan
 */
public class BMIPage extends javax.swing.JFrame {

    /**
     * Creates new form TrackPage
     */
    int userId;
    public BMIPage() {
        initComponents();
    }
// method to compute BMI and display result
public void computeAndDisplayBMI() {
    try {
        // Get user input for height and weight
        float height = Float.parseFloat(txt_height.getText());
        float weight = Float.parseFloat(txt_weight.getText());
        // Compute BMI
        float bmi = calculateBMI(height, weight);

        // Display the BMI result in a JTextArea
        txt_BMI.setText("        "+bmi+"        ");
        String remarks = getRemarks(bmi);
        txt_BMI_remarks.setText("Remarks: " + remarks);

    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Please enter valid numeric values for height and weight.");
    }
}

// method to calculate BMI
    private float calculateBMI(float height, float weight) {
        // BMI formula: weight (kg) / (height (m) * height (m))
        return weight / (height * height);
    }
    private String getRemarks(float bmi) {
        if (bmi < 18.5) {
            return "You are Underweight";
        } else if (bmi >= 18.5 && bmi <= 24.9) {
            return "You are Normal";
        } else if (bmi >= 25 && bmi <= 29.9) {
            return "You are Overweight";
        } else {
            return "You are Obese";
        }
        }
  private void uploadBMIToDatabase(float bmi) {
      try{
        if (SessionManager.isLoggedIn()) {
            User currentUser = SessionManager.getCurrentUser();
            userId = currentUser.getId();
            } else {
            JOptionPane.showMessageDialog(this, "unsuccessful");
        // Handle not logged in
         }
            // Establish database connection (update with your connection
            Connection con = DBConnection.getConnection();
            // Insert BMI into the database
            String sql = "INSERT INTO bmi (id, bmi_value) VALUES (?,?)";
            PreparedStatement pst = con.prepareStatement(sql);
            pst.setInt(1,userId);
            pst.setFloat(2,bmi);
        // Execute the insert query
            int rowsInserted = pst.executeUpdate();
            if (rowsInserted > 0) {
                JOptionPane.showMessageDialog(this, "BMI uploaded to the database successfully.");
            } else {
                JOptionPane.showMessageDialog(this, "Failed to upload BMI to the database.");
            }
        }catch (SQLException e) {
            JOptionPane.showMessageDialog(this, "Error uploading BMI to the database:\n" + e.getMessage());
            e.printStackTrace();
        }
    }
    private void saveBMIToDatabase() {
        try {
            // Get the computed BMI from the JTextArea
            String bmiString = txt_BMI.getText().replace("                ", "");
            float bmi = Float.parseFloat(bmiString);

            // Upload BMI to the database
            uploadBMIToDatabase(bmi);

        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Please compute BMI first.");
        }
    }
//FOR HOMEPAGE DISPLAY
    public class DatabaseHandler {
        private BMICallback callback;
        public DatabaseHandler(BMICallback callback) {
        this.callback = callback;
    }
        // Method to get the last inserted BMI value
        public double getLastInsertedBMI() {
            double lastBMI = 0.0;

            try (Connection con = DBConnection.getConnection();
                 Statement stmt = con.createStatement()) {
                String sql = "SELECT bmi_value FROM bmi ORDER BY bmi_id DESC LIMIT 1";
                ResultSet rs = stmt.executeQuery(sql);

                if (rs.next()) {
                    lastBMI = rs.getDouble("bmi_value");
                }

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

            return lastBMI;
        }
    }
    public interface BMICallback {
    void updateBMI(double lastBMI);
}
//update
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void a_trackMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void a_homeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void b_computeActionPerformed(java.awt.event.ActionEvent evt) {...4 lines }

    private void bmi_databaseActionPerformed(java.awt.event.ActionEvent evt) {...4 lines }

    private void jPanel8MouseClicked(java.awt.event.MouseEvent evt) {
    private void jPanel8MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }
    private void a_dietMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }
    private void a_waterMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }
    private void a_logoutMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    /**...3 lines */
    public static void main(String args[]) {...34 lines }

    // Variables declaration - do not modify
    private javax.swing.JPanel a_bmi;
    private javax.swing.JPanel a_diet;
    private javax.swing.JPanel a_home;
    private javax.swing.JPanel a_logout;
    private javax.swing.JPanel a_track;
    private javax.swing.JPanel a_water;
    private javax.swing.JButton b_compute;
    private javax.swing.JButton bmi_database;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel8;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextArea txt_BMI;
    private javax.swing.JTextArea txt_BMI_remarks;
    private javax.swing.JTextField txt_height;
    private javax.swing.JTextField txt_weight;
    // End of variables declaration
}
```

7.7. DietPlanPage.java

```
package JFrame;
import static JFrame.DBConnection.con;
import javax.swing.JOptionPane;
import java.sql.*;

/**
 *
 * @author chrisjordan
 */
public class DietPlanPage extends javax.swing.JFrame {

    /**
     * Creates new form TrackPage
     */

    public DietPlanPage() {
        initComponents();
    }
    @SuppressWarnings("unchecked")
    Generated Code

    private void a_trackMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void a_homeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void deficit_EXPLORE_buttonActionPerformed(java.awt.event.ActionEvent evt) {...6 lines }

    private void recomp_EXPLORE_button1ActionPerformed(java.awt.event.ActionEvent evt) {...6 lines }

    private void surplus_EXPLORE_button2ActionPerformed(java.awt.event.ActionEvent evt) {...6 lines }

    private void jPanel8MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void a_bmiMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void a_waterMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void a_logoutMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }
// CARD 2 [204,204,255], CARD 3 [255,102,102]
    /**...3 lines */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {...5 lines });
    }

    // Variables declaration - do not modify
    private javax.swing.JPanel a_bmi;
    private javax.swing.JPanel a_diet;
    private javax.swing.JPanel a_home;
    private javax.swing.JPanel a_logout;
    private javax.swing.JPanel a_track;
    private javax.swing.JPanel a_water;
    private javax.swing.JButton deficit_EXPLORE_button;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    private javax.swing.JPanel jPanel8;
    private javax.swing.JButton recomp_EXPLORE_button1;
    private javax.swing.JButton surplus_EXPLORE_button2;
    // End of variables declaration
}
```

7.7.1. CaloricDeficitPage.java

```
package JFrame;
/**
 *
 * @author chrisjordan
 */
public class CaloricDeficitPage extends javax.swing.JFrame {
    public CaloricDeficitPage() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void dietD_backActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        DietPlanPage d1 = new DietPlanPage();
        d1.setVisible(true);
        this.dispose();
    }

    /**...3 lines */
    public static void main(String args[]) {...34 lines }

    // Variables declaration - do not modify
    private javax.swing.JButton dietD_back;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    // End of variables declaration
}
```

7.7.2. BodyRecompPage.java

```
package JFrame;
/**
 *
 * @author chrisjordan
 */
public class BodyRecompPage extends javax.swing.JFrame {
    public BodyRecompPage() {
        initComponents();
    }
    @SuppressWarnings("unchecked")
    Generated Code

    private void DietR_backActionPerformed(java.awt.event.ActionEvent evt) {...6 lines }

    /**...3 lines */
    public static void main(String args[]) {...38 lines }

    // Variables declaration - do not modify
    private javax.swing.JButton DietR_back;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    // End of variables declaration
}
```

7.7.3. CaloricSurplusPage.java

```
package JFrame;
/**
 *
 * @author chrisjordan
 */
public class CaloricSurplusPage extends javax.swing.JFrame {
    public CaloricSurplusPage() {
        initComponents();
    }
    @SuppressWarnings("unchecked")
    Generated Code

    private void dietS_backActionPerformed(java.awt.event.ActionEvent evt) {...7 lines }

    /**...3 lines */
    public static void main(String args[]) {...46 lines }

    // Variables declaration - do not modify
    private javax.swing.JButton dietS_back;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    // End of variables declaration
}
```

7.8. WaterTrackerPage.java

```
package JFrame;
import static JFrame.DBConnection.con;
import javax.swing.JOptionPane;
import java.sql.*;
import javax.swing.table.DefaultTableModel;
/**
 *
 * @author chrisjordan
 */
public class WaterTracker extends javax.swing.JFrame {

    /**
     * Creates new form TrackPage
     */
    int S_no, userId; float Consumed;
    DefaultTableModel model;
    public WaterTracker() {
        initComponents();
        setWaterTable();
    }
    public boolean insertWater(){
        boolean isAdded = false;
        Consumed= Float.parseFloat(water_textfield.getText());

        try{
            if (SessionManager.isLoggedIn()) {
                User currentUser = SessionManager.getCurrentUser();
                userId = currentUser.getId();
                } else {
                JOptionPane.showMessageDialog(this, "unsuccessful");
            // Handle not logged in
             }
            Connection con = DBConnection.getConnection();
            String sql = "insert into water_tracker(id, consumed) values(?,?)";
            //String sql = "update macro set bcalories=?, bprotein=?, lcalories=?, lprotein=?, scalories
            PreparedStatement pst = con.prepareStatement(sql);
            pst.setFloat(1, userId);
            pst.setFloat(2,Consumed);

            int rowCount = pst.executeUpdate();
            if (rowCount > 0) {
                isAdded= true;
            } else {
                isAdded= false;
            }

    } catch (Exception e) {
        e.printStackTrace(System.out);
        isAdded = false;

    }
        return isAdded;
}
    public boolean updatewater(){
        boolean isUpdated = false;
        Consumed= Float.parseFloat(water_textfield.getText());
        try{
            if (SessionManager.isLoggedIn()) {
                User currentUser = SessionManager.getCurrentUser();
                userId = currentUser.getId();
                } else {
                JOptionPane.showMessageDialog(this, "Unsuccessful");
            // Handle not logged in
             }
            Connection con = DBConnection.getConnection();
            //String sql = "update water_tracker set consumed=? where id = ?";
            String sql = "UPDATE water_tracker SET consumed = ? WHERE id = ? ORDER BY wid DESC LIMIT 1";
            PreparedStatement pst = con.prepareStatement(sql);
            pst.setFloat(1,Consumed);
            int rowCount = pst.executeUpdate();
            if (rowCount > 0) {
                isUpdated= true;
            } else {
                isUpdated= false;
            }

    } catch (Exception e) {
        e.printStackTrace(System.out);
        isUpdated = false;

    }
        return isUpdated;
}
    public void setWaterTable() {
    int counter = 1;

    try {
        if (SessionManager.isLoggedIn()) {
            User currentUser = SessionManager.getCurrentUser();
            userId = currentUser.getId();
        } else {
            JOptionPane.showMessageDialog(this, "Unsuccessful");
            // Handle not logged in
            return;  // Exit the method if not logged in
        }

        Connection con = DBConnection.getConnection();
        String sql = "SELECT consumed FROM water_tracker WHERE id = ?";
        try (PreparedStatement pst = con.prepareStatement(sql)) {
            pst.setInt(1, userId);

            clearTable();

            try (ResultSet rs = pst.executeQuery()) {
                while (rs.next()) {
                    S_no = counter;
                    Consumed = rs.getFloat("consumed");
                    Object[] obj = {S_no, Consumed};
                    model = (DefaultTableModel) water_table.getModel();
                    model.addRow(obj);
                    counter++;
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

    public void clearTable(){...4 lines }
// method to compute BMI and display result

    @SuppressWarnings("unchecked")
    Generated Code

    private void a_trackMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void a_homeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void water_textfieldActionPerformed(java.awt.event.ActionEvent evt) {...3 lines }

    private void WATER_ADDActionPerformed(java.awt.event.ActionEvent evt) {...13 lines }

    private void UPDATE_WATERActionPerformed(java.awt.event.ActionEvent evt) {...11 lines }

    private void jPanel8MouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    private void a_bmiMouseClicked(java.awt.event.MouseEvent evt) {...6 lines }

    public static void main(String args[]) {...38 lines }

    // Variables declaration - do not modify
    private javax.swing.JButton UPDATE_WATER;
    private javax.swing.JButton WATER_ADD;
    private javax.swing.JPanel a_bmi;
    private javax.swing.JPanel a_diet;
    private javax.swing.JPanel a_home;
    private javax.swing.JPanel a_logout;
    private javax.swing.JPanel a_track;
    private javax.swing.JPanel a_water;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel8;
    private javax.swing.JScrollPane jScrollPane1;
    private rojeru_san.complementos.RSTableMetro water_table;
    private javax.swing.JTextField water_textfield;
    // End of variables declaration
}
```

7.9. DBConnection.java

```java
package JFrame;

import java.sql.Connection;
import java.sql.DriverManager;
/**
 *
 * @author chrisjordan
 */
public class DBConnection {
    static Connection con = null;

    public static Connection getConnection(){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/test1","root","christ123");
        }catch(Exception e){
            e.printStackTrace();
        }
        return con;
    }
}
```

7.10. SessionManager.java

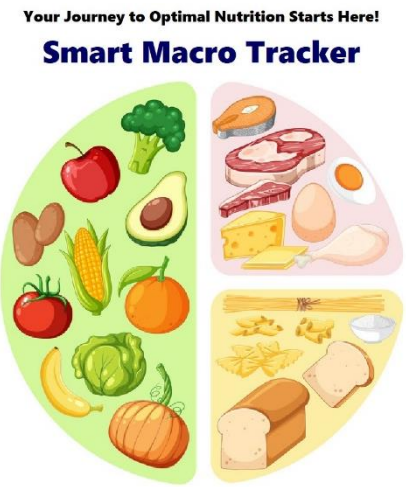
```java
package JFrame;

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */

/**
 *
 * @author chrisjordan
 */
public class SessionManager {
    private static User currentUser;

    public static void setCurrentUser(User user) {
        currentUser = user;
    }

    public static User getCurrentUser() {
        return currentUser;
    }

    public static boolean isLoggedIn() {
        return currentUser != null;
    }

    public static void logout() {
        currentUser = null;
    }
}
```

7.11. Users.java

```java
package JFrame;
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
/**
 *
 * @author chrisjordan
 */
public class User {
    private int id;
    private String username;

    // Constructors
    public User() {
        // Default constructor
    }
    public User(int id, String username) {
        this.id = id;
        this.username = username;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
}
```

CHAPTER 8

RESULTS



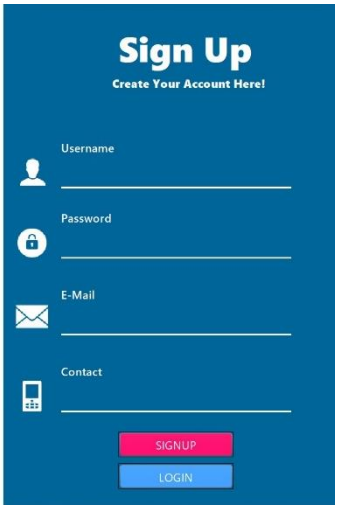
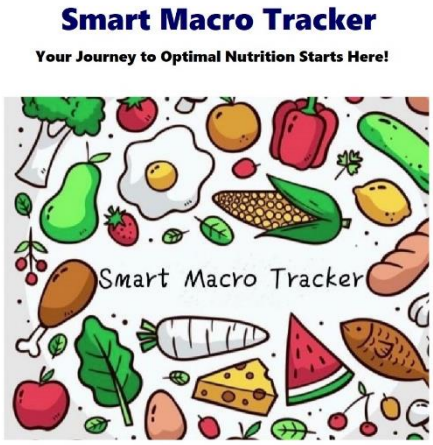


Fig: 8.1. Sign Up Page



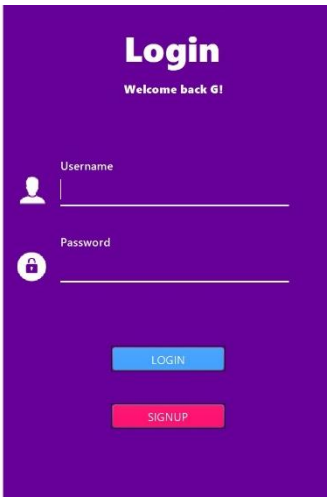


Fig: 8.2. Login Page

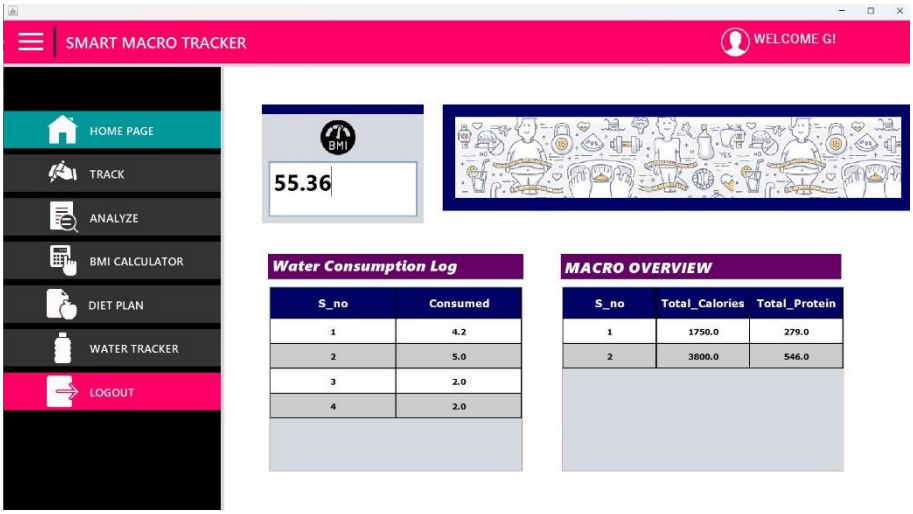


Fig: 8.3. Home Page

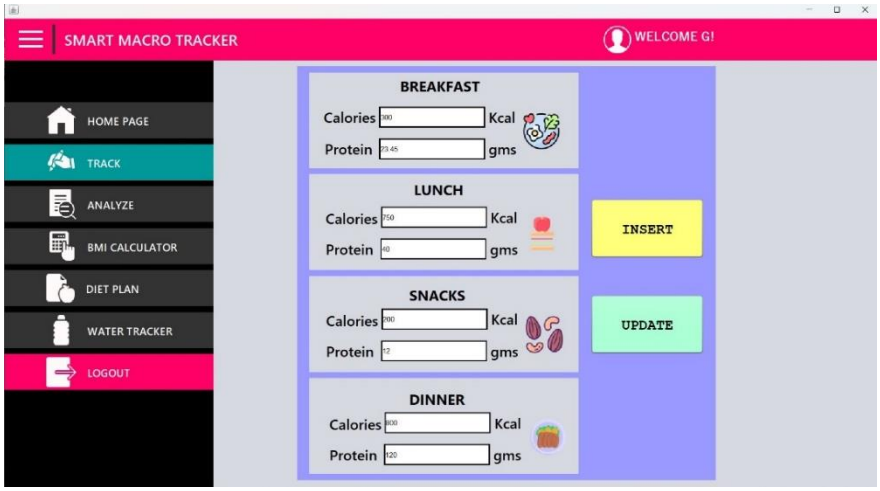
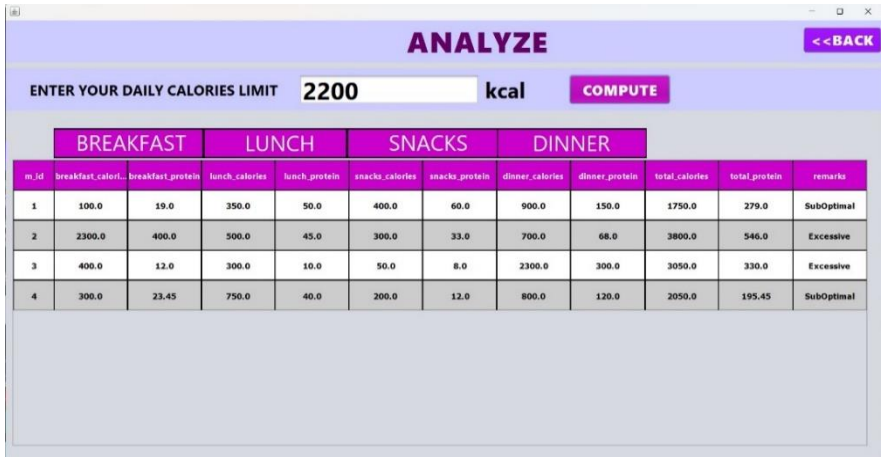


Fig: 8.4. Track Page



m_id	breakfast_calori..	breakfast_protein	lunch_calories	lunch_protein	snacks_calories	snacks_protein	dinner_calories	dinner_protein	total_calories	total_protein	remarks
1	100.0	19.0	350.0	50.0	400.0	60.0	900.0	150.0	1750.0	279.0	SubOptimal
2	2300.0	400.0	500.0	45.0	300.0	33.0	700.0	68.0	3800.0	546.0	Excessive
3	400.0	12.0	300.0	10.0	50.0	8.0	2300.0	300.0	3050.0	330.0	Excessive
4	300.0	23.45	750.0	40.0	200.0	12.0	800.0	120.0	2050.0	195.45	SubOptimal

Fig: 8.5. Analyze Page

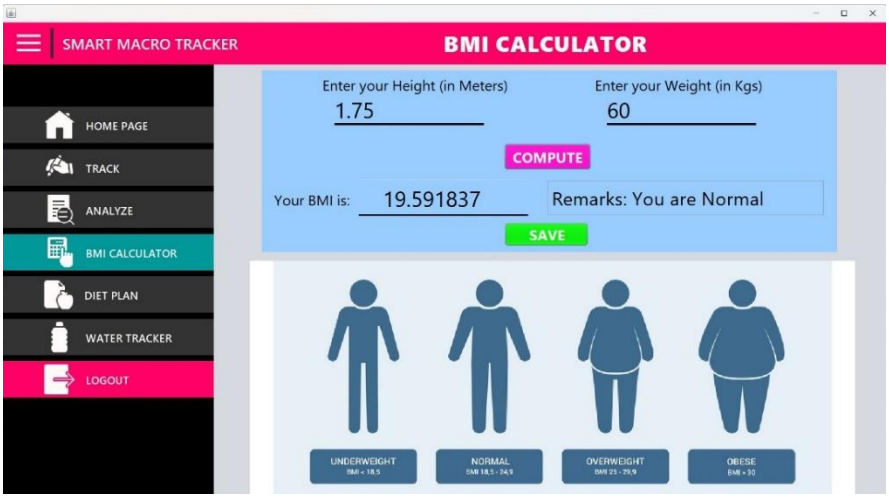


Fig: 8.6. BMI Calculator

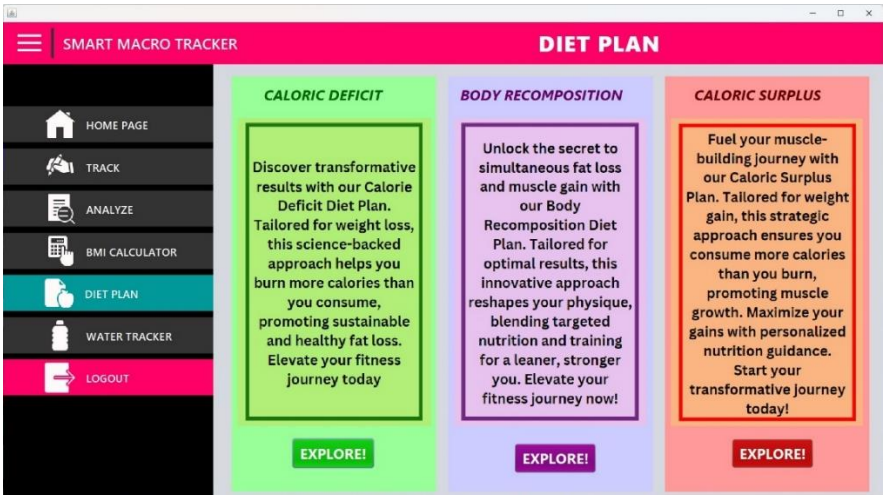


Fig: 8.7. Diet Plan Page

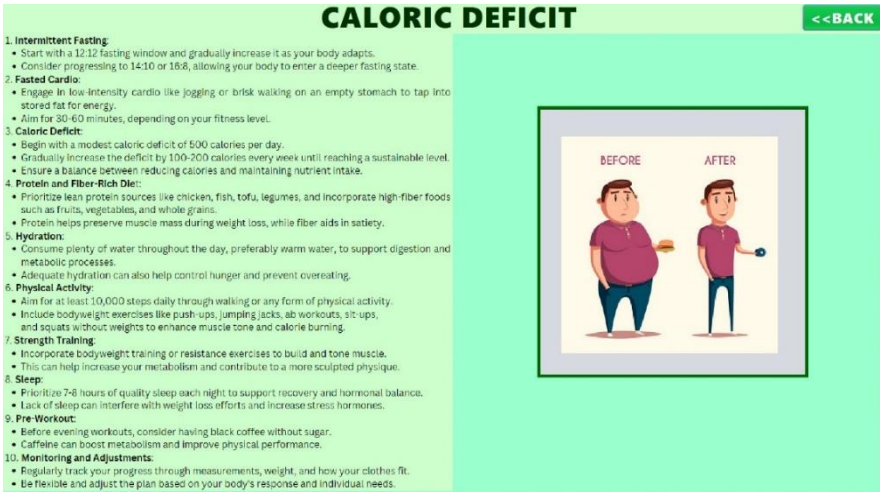


Fig: 8.7.1. Caloric Deficit Page

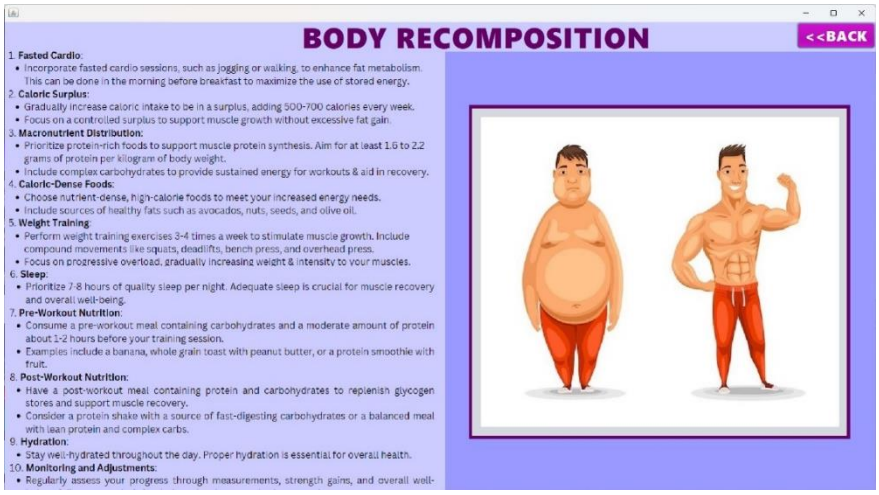


Fig: 8.7.2. Body Re-composition Page

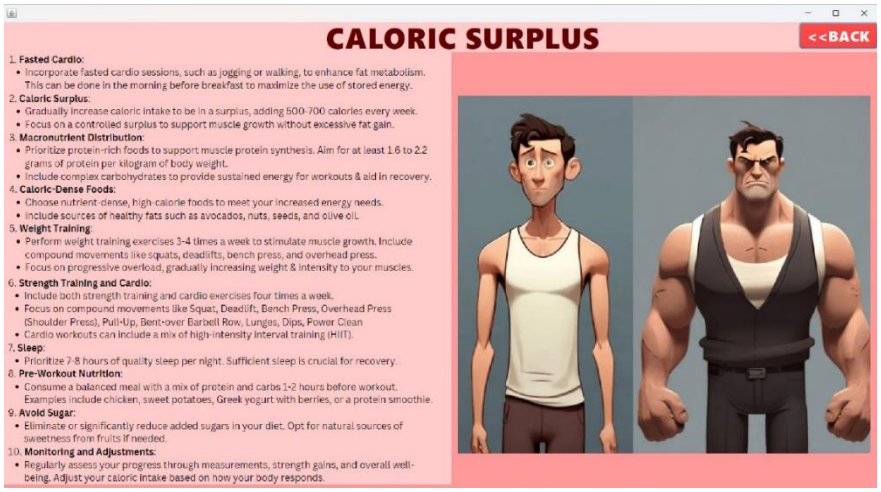


Fig: 8.7.3. Caloric Surplus Page

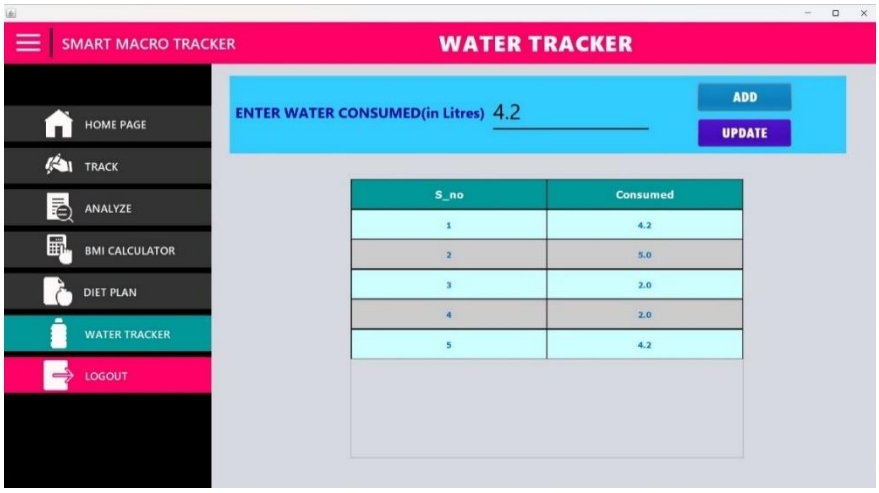


Fig: 8.8. Water Tracker Page

CHAPTER 9

CONCLUSION

In summary, the creation and execution of the Smart Macro Tracker project mark a critical turning point in the field of nutrition and health management. The project offers a complete solution with capabilities including counting calories and protein, calculating BMI, creating diet programmes, and keeping track of water intake. It does this by utilising Java, MySQL, and Swing for the user interface. Through an analysis table, the programme not only gives users the ability to track their nutritional consumption, but it also offers insightful information. In the future, integrating personalised food recommendations, being compatible with activity trackers, and having a more aesthetically pleasing user interface are possible areas for development. The application's usefulness might also be increased by investigating cloud-based storage alternatives for easy data accessing and remaining current with dietary information. The Smart Macro Tracker establishes a strong basis, and its further improvements might have a long-lasting influence on those looking for a comprehensive approach to health and wellbeing, enforcing the need to empower one’s physic to reach its maximum potential, through efficient recording and analysis of nutrition.

9.1. REFERENCES

. Downs, C., & McLaughlin, C. (2009). "Java Programming 24-Hour Trainer." Wrox.

. Kofler, M. (2008). "Java 6 - Swing: Guis & Network/Socket Programming." Pearson.

. Beaulieu, A. (2009). "Learning SQL." O'Reilly Media.

. Eckstein, R., Loy, D., & Wood, J. (2005). "Java Swing." O'Reilly Media.

. Whitney, S. R., & Crowe, T. (2015). "Understanding Nutrition." Cengage Learning.

. GeeksforGeeks

. W3Schools

. JavaTpoint

. TutotrialPoint