



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

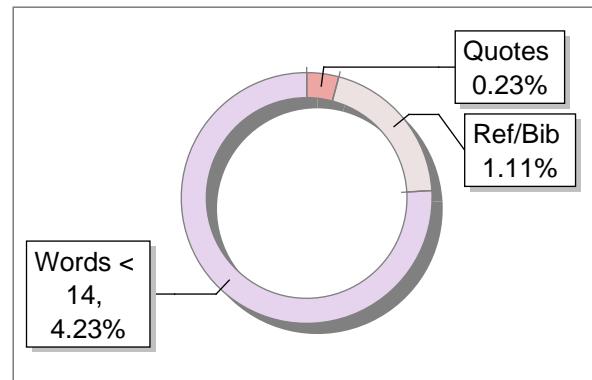
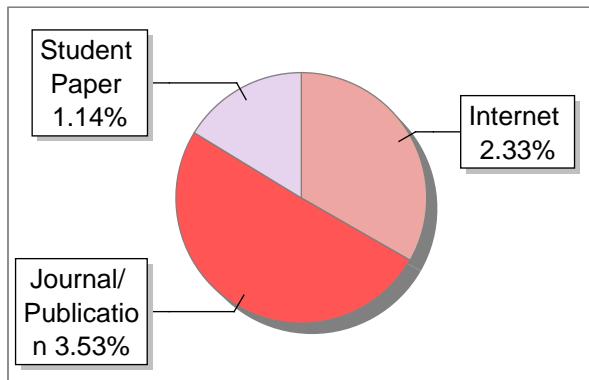
Academic Year: 2023-24

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 Similarity Report

7

SIMILARITY %

31

MATCHED SOURCES

A

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

ABSTRACT	I
ACKNOWLEDGEMENT	II
LIST OF FIGURES	VI
LIST OF TABLES	VII
1. INTRODUCTION	
1.1. PROBLEM DEFINITION	2
1.2. OBJECTIVES	2
1.3. METHODOLOGY	3
1.4. EXPECTED OUTCOMES	4
1.5. HARDWARE AND SOFTWARE REQUIREMENTS	4
2. FUNDAMENTALS OF JAVA	
2.1. INTRO TO JAVA	5
2.2. PROS OF JAVA	6
2.3. DATA TYPES	7
2.4. CONTROL FLOW	8
2.5. METHODS	10
2.6. OBJECT ORIENTED PROGRAMMING	10
2.7. EXCEPTION HANDLING	13
2.8. FILE HANDLING	14
2.9. PACKAGES AND IMPORT	15
2.10. INTERFACES	15
2.11. CONCURRENCY	16

3. JAVA SWING GUIDE	
3.1. INTRO TO JAVA SWING	17
3.2. SWING COMPONENTS	17
3.3. SWING CONTAINERS	19
3.4. LAYOUT MANAGER	20
3.5. EVENT HANDLING	22
3.6. ACTION LISTENERS	22
4. FUNDAMENTALS OF DBMS	
4.1. INTRODUCTION TO DBMS	23
4.2. FEATURES OF DBMS	23
4.3. DATA MODEL	24
4.4. 3 - SCHEMA ARCHITECTURE	25
4.5. COMPONENT MODULES	25
4.6. ENTITY-RELATIONSHIP (ER) MODEL	26
4.7. RELATIONAL SCHEMA	27
5. FUNDAMENTALS OF SQL	
5.1. SQL OVERVIEW	28
5.2. SQL COMMANDS	28
5.3. DDL (DATA DEFINITION LANGUAGE)	29
5.4. DML (DATA MANIPULATION LANGUAGE)	29
5.5. DCL (DATA CONTROL LANGUAGE)	29
5.6. TCL (TRANSACTION CONTROL LANGUAGE)	30
6. DESIGN AND ARCHITECTURE	31

7. IMPLEMENTATION	
7.1. SIGNUP PAGE	32
7.2. LOGIN PAGE	34
7.3. HOME PAGE	35
7.4. TRACK PAGE	37
7.5. ANALYZE PAGE	39
7.6. BMI CALCULATOR PAGE	41
7.7. DIET PLAN PAGE	42
7.7.1. CALORIC DEFICIT PAGE	43
7.7.2. BODY RECOMPOSITION PAGE	43
7.7.3. CALORIC SURPLUS PAGE	43
7.8. WATER TRACKER PAGE	44
7.9. DB CONNECTION PAGE	45
7.10. SESSION MANAGER PAGE	45
7.11. USERS PAGE	45
8. RESULTS	
8.1. SIGN UP PAGE	46
8.2. LOGIN PAGE	46
8.3. HOME PAGE	46
8.4. TRACK PAGE	47
8.5. ANALYZE PAGE	47
8.6. BMI CALCULATOR	47
8.7. DIET PLAN	48
8.8. CALORIC DEFICIT PAGE	48
8.9. BODY RE-COMOSITION PAGE	48
8.10. CALORIC SURPLUS PAGE	49
8.11. WATER TRACKER PAGE	49
9. CONCLUSION	50
9.1. REFERENCES	50

LIST OF FIGURES

Figure No	Figure Description	Page No
1.	Overview of Basic Conditional Statements	8
2.	Flow Chart of Looping Statements in Java	9
3.	Working of Break Statement	9
4.	Working of Continue Statement	9
5.	Pictorial representation of a Class and Objects in Java	11
6.	Encapsulation	11
7.	Single, Multilevel and Hierarchical Representation	11
8.	Polymorphic Visualization of a Man	12
9.	Abstraction Depicted using a Car	12
10.	Overview of Interfaces	12
11.	Try Catch Block Pseudocode	13
12.	Multiple Catch Block Pseudocode	13
13.	Finally Block Pseudocode	13
14.	Try with Resources Pseudocode	14
15.	Packages and Import in Java	15
16.	Simple code Demonstrating Concurrency in Java	16
17.	JButton Using Java Swing	17
18.	JLabel Using Java Swing	18
19.	JTextField Using Java Swing	18

20.	JTextArea Using Java Swing	19
21.	JFrame Using Java Swing	19
22.	JPanel Using Java Swing	20
23.	Visualizing Flow Layout Feature in Java Swing	20
24.	Visualizing Border Layout Feature in Java Swing	21
25.	Visualizing Grid Layout Feature in Java Swing	21
26.	Visualizing Absolute Layout Feature in Java Swing	21
27.	Action Listeners to Handle Action Events in Java Swing	22
28.	Three Schema Architecture of DBMS	25
29.	Component Modules of DBMS	26
30.	Overview of MySQL Commands	28

LIST OF TABLES

Table No	Table Description	Page No
1.	Description of Data Types in Java	7
2.	Pre- Defined Functions used in File Handling	14

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 sure 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', '\beta'	characters representation of ASCII values 0 to 255
int	two's-complement integer	0	32 bits	-2,-1,0,1,2	-2,147,483,648 to 2,147,483,647
long	two's-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 in further applications that requires multiple condition processing.

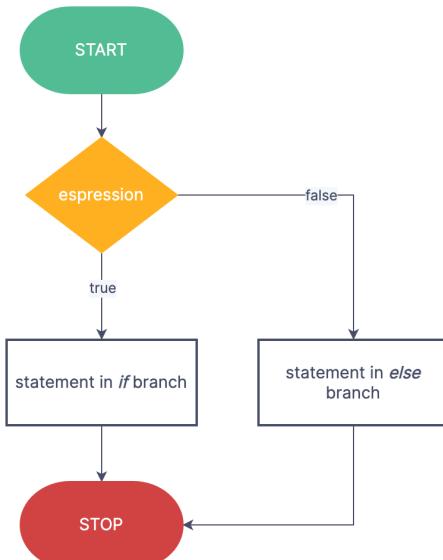


Fig: 1. Overview of Basic Conditional Statements

(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.

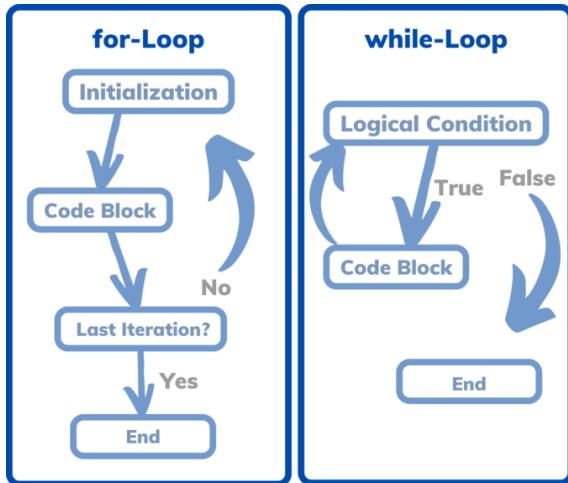


Fig: 2. Flow Chart of Looping Statements in Java

(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.

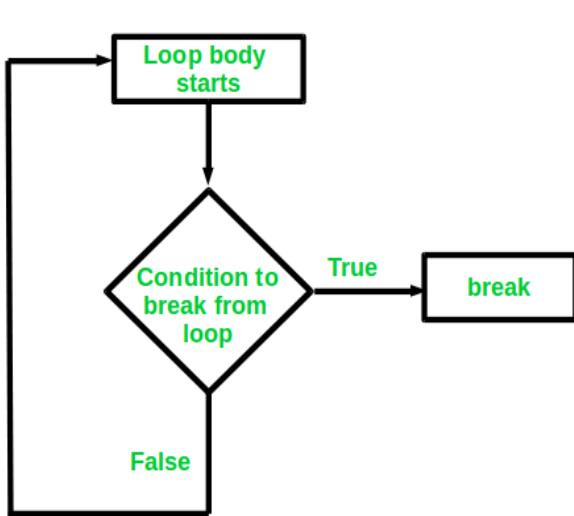


Fig: 3. Working of Break Statement

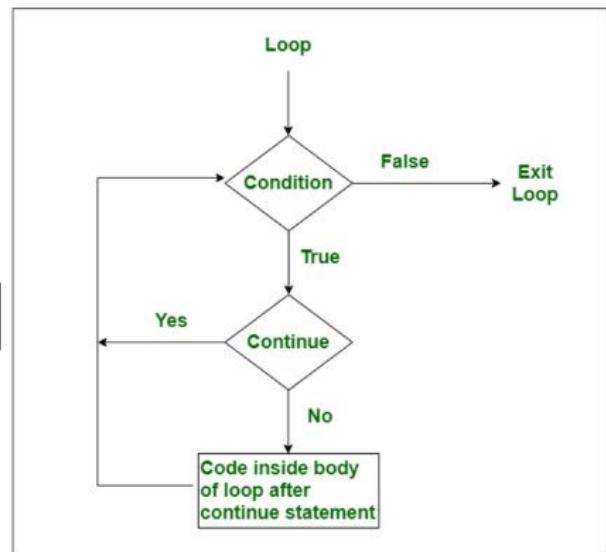


Fig: 4. Working of Continue Statement

2.5 METHODS

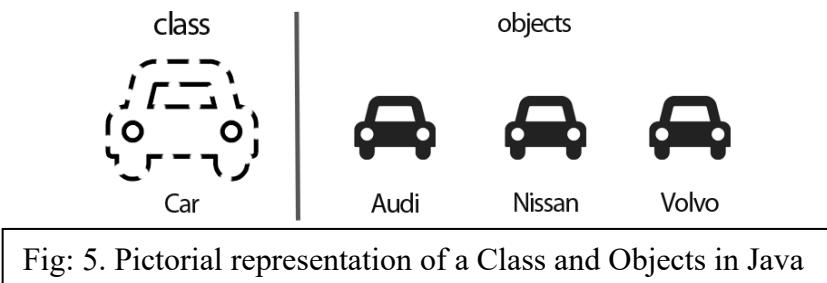
Methods are modular chunks of code that are intended to carry out certain tasks; this helps to improve code organisation and encourage reusability. A method is defined by defining its return type, which indicates the data type of the result or "void" if none, its visibility (public, private, or protected), 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

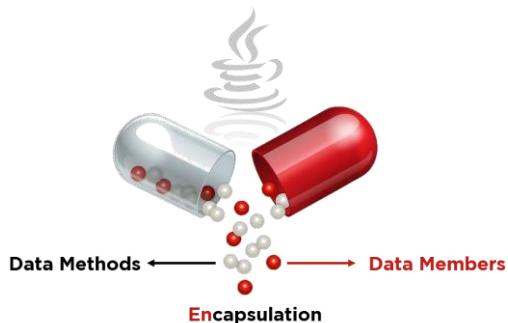
The programming paradigm known as object-oriented programming, or OOP, organises code around objects, which are just instances of classes. It encourages polymorphism, which allows objects to take on different forms, inheritance, which permits classes to inherit traits and behaviours, and encapsulation, 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.

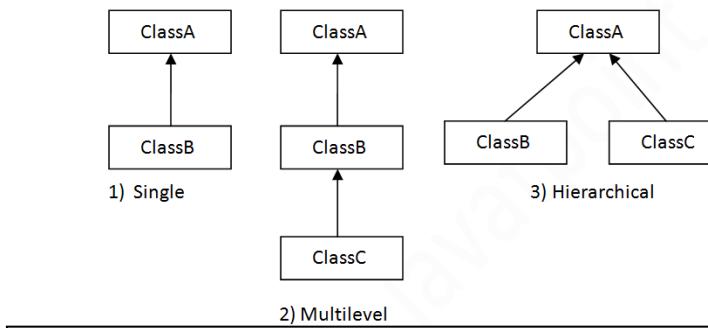
(ii) CLASS: In Java, a class is a blueprint that specifies an object's composition and actions. It acts as a template, including properties and functions, directing the production of object instances.



(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.



(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.



(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.

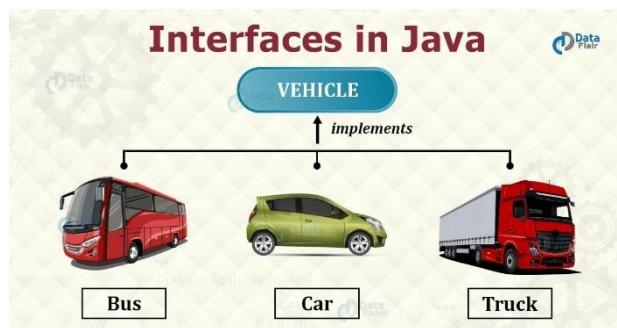


Fig: 10. Overview of Interfaces

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.

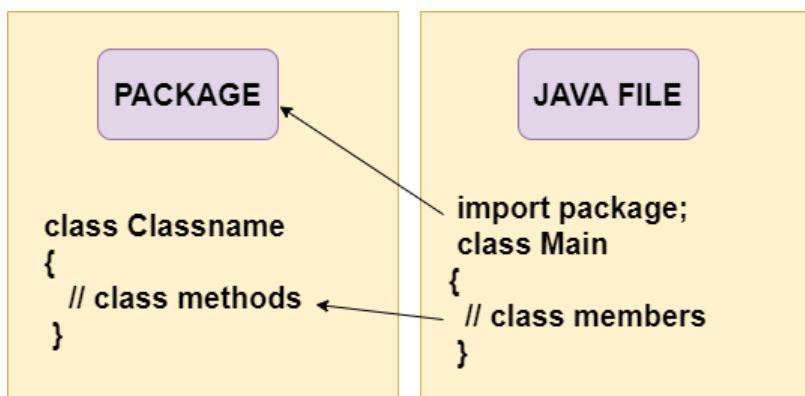


Fig: 15. Packages and Import in Java

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 `synchronized` 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.

```
1  public class SimpleConcurrencyExample {  
2  
3      public static void main(String[] args) {  
4          // Create and start two threads  
5          Thread thread1 = new Thread(() -> printNumbers("Thread 1"));  
6          Thread thread2 = new Thread(() -> printNumbers("Thread 2"));  
7  
8          thread1.start();  
9          thread2.start();  
10     }  
11  
12     private static void printNumbers(String threadName) {  
13         for (int i = 1; i <= 5; i++) {  
14             System.out.println(threadName + ": " + i);  
15             try {  
16                 // Introducing a short delay to emphasize concurrency  
17                 Thread.sleep(500);  
18             } catch (InterruptedException e) {  
19                 e.printStackTrace();  
20             }  
21         }  
22     }  
23 }
```

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.

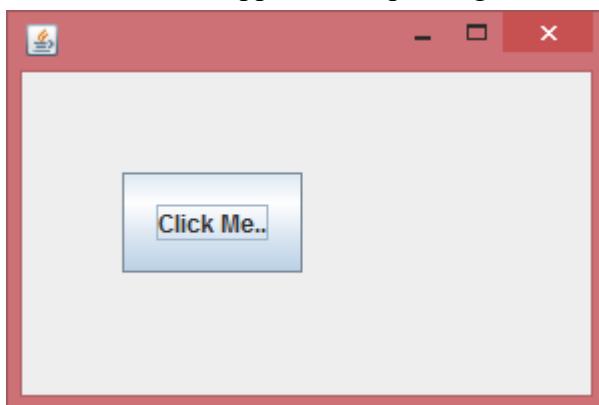


Fig: 17. JButton Using Java Swing

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.

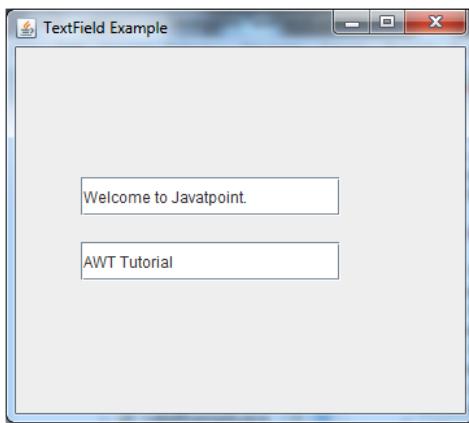


Fig: 19. JTextField Using Java Swing

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.

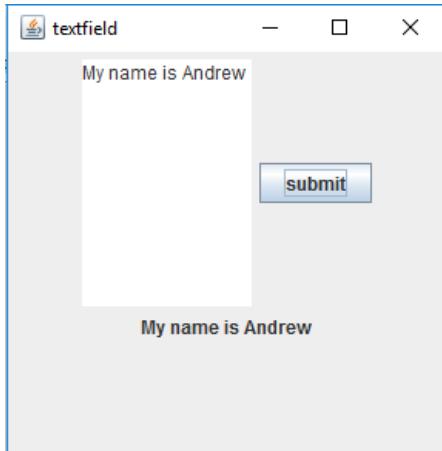


Fig: 20. JTextArea Using Java Swing

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.

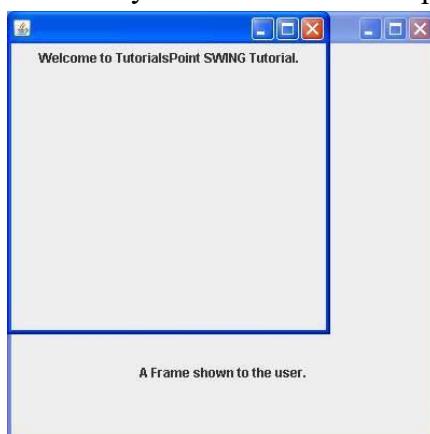


Fig: 21. JFrame Using Java Swing

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.

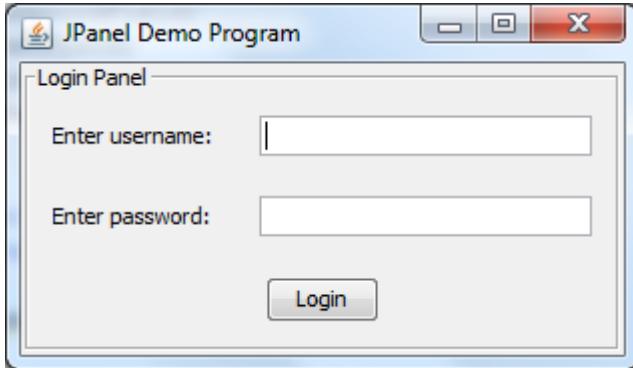


Fig: 22. JPanel Using Java Swing

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.

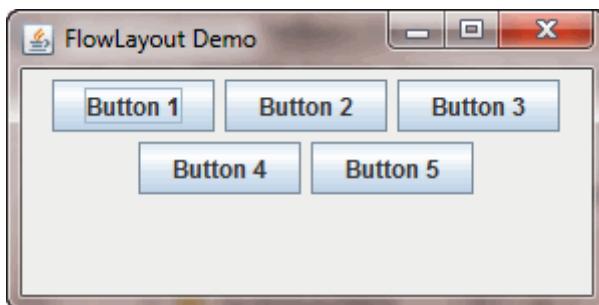


Fig: 23. Visualizing Flow Layout Feature in Java Swing

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.

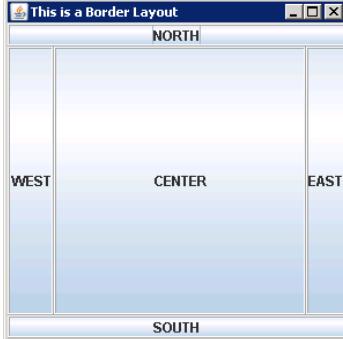


Fig: 24. Visualizing Border Layout Feature in Java Swing

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

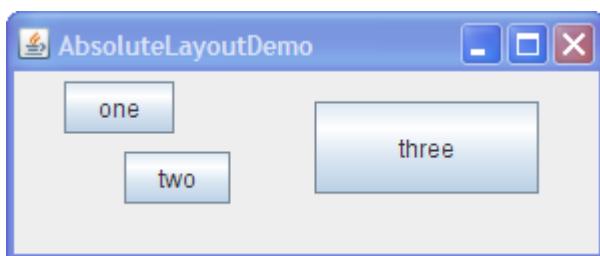


Fig: 26. Visualizing Absolute Layout Feature in Java Swing

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.

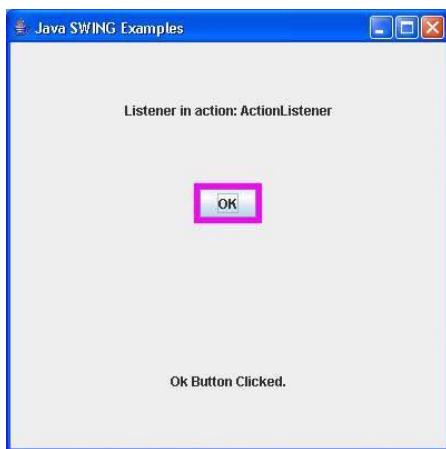


Fig: 27. Action Listeners to Handle Action Events in Java Swing

CHAPTER 4

4. FUNDAMENTALS OF DBMS

4.1 INTRODUCTION 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.

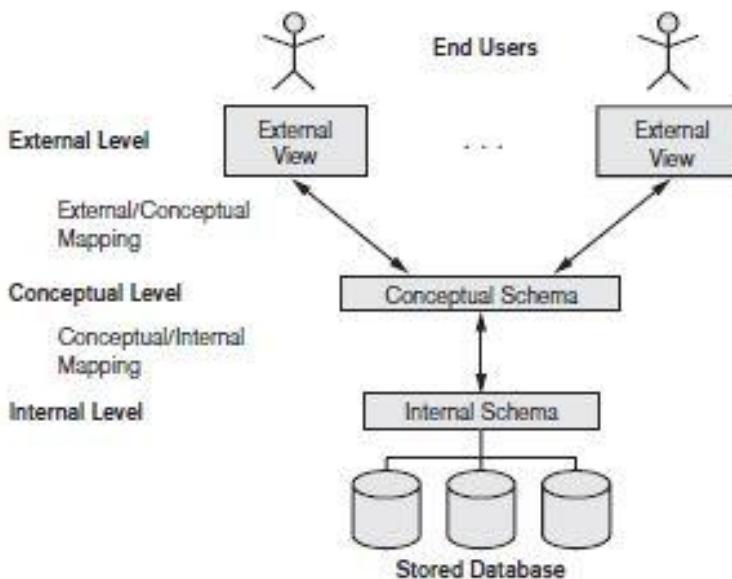


Fig: 28. Three Schema Architecture of DBMS

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.

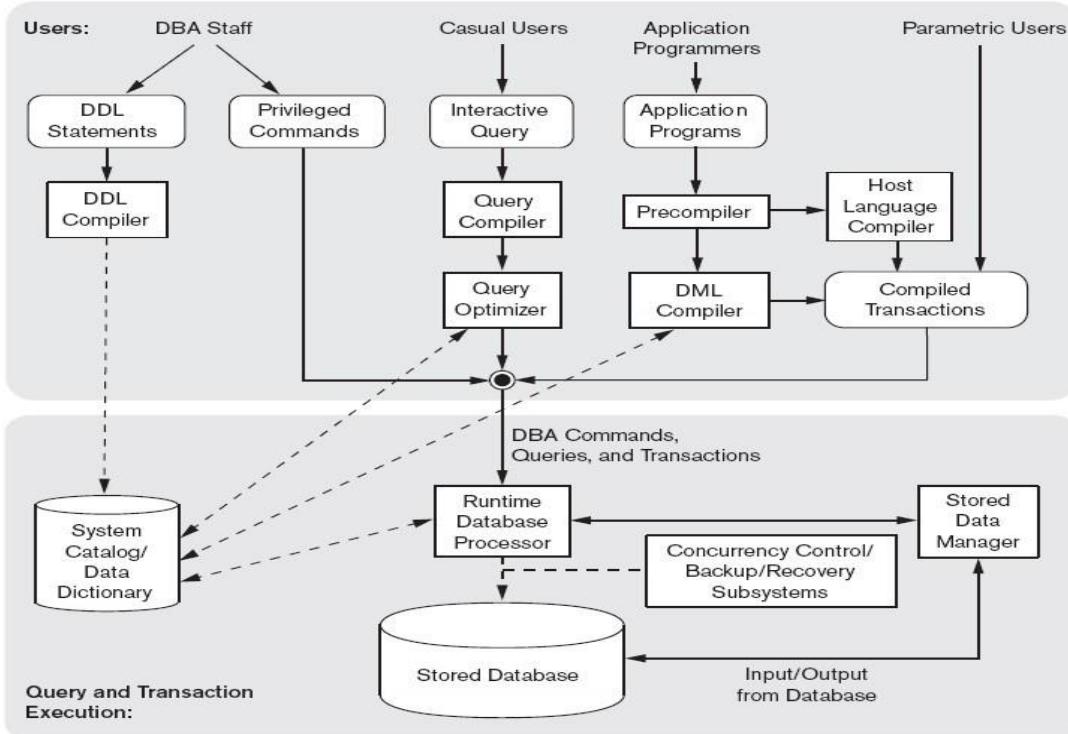


Fig: 29. Component Modules of DBMS

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

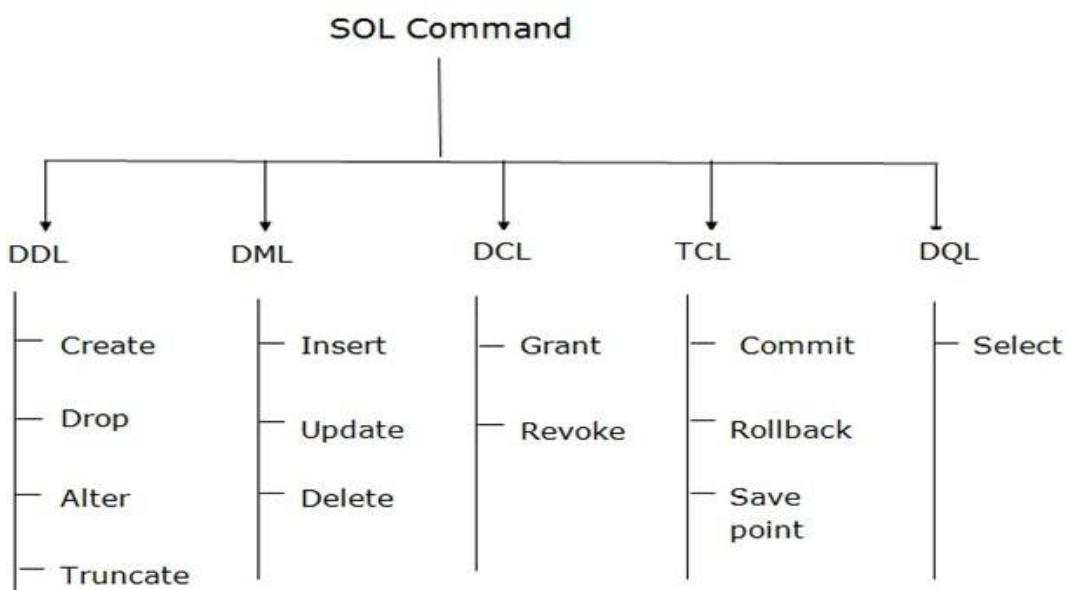


Fig: 30. Overview of MySQL Commands

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.

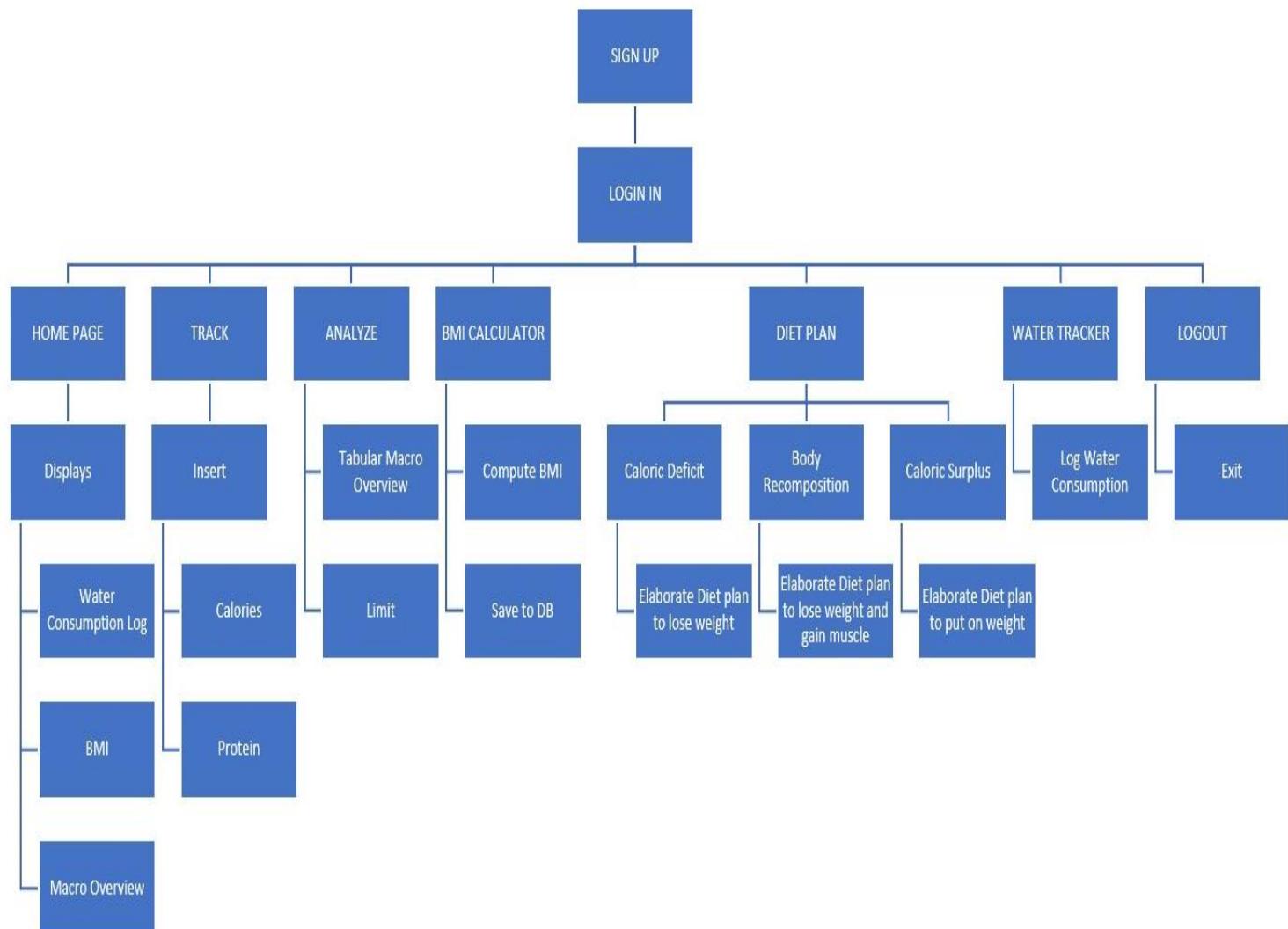
COMMIT: To permanently apply the modifications made during the current transaction. The modifications are irreversible once a COMMIT is sent out and cannot be undone. It represents the transaction's successful conclusion.

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

Smart Macro Tracker

```
330     }
331 }
332 
333     private void txt_contactActionPerformed(java.awt.event.ActionEvent evt) {
334         // TODO add your handling code here:
335     }
336 
337 /**
338 * @param args the command line arguments
339 */
340 public static void main(String args[]) {
341     /* Set the Nimbus look and feel */
342     // Look and feel setting code (optional)
343 
344     /* Create and display the form */
345     java.awt.EventQueue.invokeLater(new Runnable() {
346 
347         @Override
348         public void run() {
349             new SignUpPage().setVisible(true);
350         }
351     });
352 }
353 
354 // Variables declaration - do not modify
355 private javax.swing.JButton jButton1;
356 private javax.swing.JButton jButton2;
357 private javax.swing.JLabel jLabel1;
358 private javax.swing.JLabel jLabel10;
359 private javax.swing.JLabel jLabel11;
360 private javax.swing.JLabel jLabel12;
361 private javax.swing.JLabel jLabel14;
362 private javax.swing.JLabel jLabel16;
363 private javax.swing.JLabel jLabel17;
364 private javax.swing.JLabel jLabel18;
365 private javax.swing.JLabel jLabel19;
366 private javax.swing.JLabel jLabel2;
367 private javax.swing.JLabel jLabel20;
368 private javax.swing.JLabel jLabel21;
369 private javax.swing.JLabel jLabel22;
370 private javax.swing.JLabel jLabel23;
371 private javax.swing.JLabel jLabel24;
372 private javax.swing.JLabel jLabel25;
373 private javax.swing.JLabel jLabel26;
374 private javax.swing.JLabel jLabel27;
375 private javax.swing.JLabel jLabel28;
376 private javax.swing.JLabel jLabel29;
377 private javax.swing.JLabel jLabel3;
378 private javax.swing.JLabel jLabel30;
379 private javax.swing.JLabel jLabel31;
380 private javax.swing.JLabel jLabel32;
381 private javax.swing.JLabel jLabel33;
382 private javax.swing.JLabel jLabel34;
383 private javax.swing.JLabel jLabel35;
384 private javax.swing.JLabel jLabel36;
385 private javax.swing.JLabel jLabel37;
386 private javax.swing.JLabel jLabel38;
387 private javax.swing.JLabel jLabel39;
388 private javax.swing.JLabel jLabel4;
389 private javax.swing.JLabel jLabel40;
390 private javax.swing.JPanel jPanel1;
391 private javax.swing.JPanel jPanel2;
392 private javax.swing.JTextField txt_contact;
393 private javax.swing.JTextField txt_email;
394 private javax.swing.JTextField txt_password;
395 private javax.swing.JTextField txt_username;
396 // End of variables declaration
397 }
```

7.2. LoginInPage.java

```

5   package JFrame;
6   import java.sql.Connection;
7   import javax.swing.JOptionPane;
8   import java.sql.SQLException;
9   import java.sql.PreparedStatement;
10  import java.sql.ResultSet;
11 
12  /**
13   * @author chrisjordan
14   */
15  public class LoginPage extends javax.swing.JFrame {
16 
17  /**
18   * Creates new form SignUpPage
19   */
20  public LoginPage() {
21      initComponents();
22  }
23 
24  //login validation
25  public boolean validateLogin(){
26      String name=login_username.getText();
27      String pwd=login_password.getText();
28 
29      if (name.equals("")) {
30          JOptionPane.showMessageDialog(this, "Please Enter Username");
31          return false;
32      }
33      if (pwd.equals("")) {
34          JOptionPane.showMessageDialog(this, "Please Enter your password");
35          return false;
36      }
37      return true;
38  }
39 
40  //Login validation
41  public void Login(){
42      String name=login_username.getText();
43      String pwd=login_password.getText();
44      try{
45          Connection con = DBConnection.getConnection();
46          PreparedStatement pst = con.prepareStatement("select id from users where name=? and password=?");
47          pst.setString(1,name);
48          pst.setString(2,pwd);
49          ResultSet rs = pst.executeQuery();
45          //redirection
46          if(rs.next()){
47              int userId = rs.getInt("id");
48 
49              // Create a User object
50              User user = new User();
51              user.setid(userId);
52 
53              // Set the logged-in user in the session
54              SessionManager.setCurrentUser(user);
55              /*int userId = rs.getInt("id");
56 
57              // Set the logged-in user ID in the session
58              SessionManager.setLoggedInUserId(userId);*/
59              JOptionPane.showMessageDialog(this,"Login successful");
60              HomePage home = new HomePage();
61              home.setVisible(true);
62              this.dispose();
63          }else{
64              JOptionPane.showMessageDialog(this,"Incorrect Credentials");
65          }
66      }catch(Exception e){
67          e.printStackTrace();
68      }
69  }
70 
71  /**
72   * This method is called from within the constructor to initialize the form.
73   * WARNING: Do NOT modify this code. The content of this method is always
74   * regenerated by the Form Editor.
75   */
76  try{
77      SignUpPage signup1 = new SignUpPage();
78      signup1.setVisible(true);
79      this.dispose();
80  }catch(Exception e){
81      e.printStackTrace();
82  }
83 
84  /**
85   * This method is called from within the constructor to initialize the form.
86   * WARNING: Do NOT modify this code. The content of this method is always
87   * regenerated by the Form Editor.
88   */
89  @SuppressWarnings("unchecked")
90  Generated Code
91 
92  private void login_usernameActionPerformed(java.awt.event.ActionEvent evt) {
93      // TODO add your handling code here:
94  }
95 
96  private void login_passwordActionPerformed(java.awt.event.ActionEvent evt) {
97      // TODO add your handling code here:
98  }
99 
100 private void jButtonActionPerformed(java.awt.event.ActionEvent evt) {
101     // TODO add your handling code here:
102     if(validateLogin()){
103         Login();
104     }
105 }
106 
107 private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
108     // TODO add your handling code here:
109     signupredirect();
110 }
111 
112 /**
113  * @param args the command line arguments
114  */
115 public static void main(String args[]) {
116     /* Set the Nimbus look and feel */
117     /* Look and feel setting code (optional) */
118     //
119 
120     /* Create and display the form */
121     java.awt.EventQueue.invokeLater(new Runnable() {
122         @Override
123         public void run() {
124             new LoginPage().setVisible(true);
125         }
126     });
127 
128     // Variables declaration - do not modify
129     private javax.swing.JButton jButton1;
130     private javax.swing.JButton jButton2;
131     private javax.swing.JLabel jLabel1;
132     private javax.swing.JLabel jLabel2;
133     private javax.swing.JLabel jLabel3;
134     private javax.swing.JLabel jLabel4;
135     private javax.swing.JLabel jLabel5;
136     private javax.swing.JLabel jLabel6;
137     private javax.swing.JLabel jLabel7;
138     private javax.swing.JLabel jLabel8;
139     private javax.swing.JLabel jLabel9;
140     private javax.swing.JPanel jPanel1;
141     private javax.swing.JPanel jPanel2;
142     private javax.swing.JTextField login_password;
143     private javax.swing.JTextField login_username;
144     // End of variables declaration
145 }
146 
```

7.3. HomePage.java

```

5  package JFrame;
6  import java.awt.BorderLayout;
7  import java.awt.Color;
8  import java.sql.Connection;
9  import java.sql.*;
10 import javax.swing.JOptionPanel;
11 import org.jfree.chart.ChartFactory;
12 import org.jfree.chart.ChartPanel;
13 import org.jfree.chart.JFreeChart;
14 import org.jfree.chart.plot.PiePlot;
15 import org.jfree.data.general.DefaultPieDataset;
16 import javax.swing.table.DefaultTableModel;
17 /**
18  * 
19  * @author chrisjordan
20  */
21 public class HomePage extends javax.swing.JFrame {
22
23     /**
24      * Creates new form HomePage
25      */
26     int userId; int S_no; float Consumed;
27     float Total_Calories, Total_Protein;
28     DefaultTableModel model;
29     public HomePage() {
30         initComponents();
31         displayLastInsertedBMI();
32         HomeSetWaterTable();
33         setMacroTable();
34     }
35     public void displayLastInsertedEMI() {
36         try {
37             if (SessionManager.isLoggedIn()) {
38                 User currentUser = SessionManager.getCurrentUser();
39
40                 Connection con = DBConnection.getConnection();
41                 String sql = "SELECT bmi_value FROM bmi WHERE id = ? ORDER BY bmi_id DESC LIMIT 1";
42
43                 try (PreparedStatement pst = con.prepareStatement(sql)) {
44                     pst.setInt(1, userId);
45                     ResultSet rs = pst.executeQuery();
46
47                     if (rs.next()) {
48                         float lastInsertedBMI = rs.getFloat("bmi_value");
49                         home_bmi.setText("" + lastInsertedBMI);
50                     } else {
51                         home_bmi.setText("No BMI");
52                     }
53
54                     rs.close();
55                 }
56
57                 con.close(); // Close the connection outside of the try-with-resources block
58             } catch (Exception e) {
59                 e.printStackTrace();
60             }
61         }
62     }
63     public void HomeSetWaterTable() {
64         int counter = 1;
65
66         try {
67             if (SessionManager.isLoggedIn()) {
68                 User currentUser = SessionManager.getCurrentUser();
69                 userId = currentUser.getId();
70             } else {
71                 JOptionPane.showMessageDialog(this, "Unsuccessful");
72                 // Handle not logged in
73                 return; // Exit the method if not logged in
74             }
75
76             Connection con = DBConnection.getConnection();
77             String sql = "SELECT consumed FROM water_tracker WHERE id = ?";
78             try (PreparedStatement pst = con.prepareStatement(sql)) {
79                 pst.setInt(1, userId);
80                 clearTable();
81                 try (ResultSet rs = pst.executeQuery()) {
82                     while (rs.next()) {
83                         S_no = counter;
84                         Consumed = rs.getFloat("consumed");
85                         Object[] obj = {S_no, Consumed};
86                         model = (DefaultTableModel) home_water.getModel();
87                         model.addRow(obj);
88                         counter++;
89                     }
90                 }
91             } catch (Exception e) {
92                 e.printStackTrace();
93             }
94         }
95         public void clearTable() {
96             DefaultTableModel model = (DefaultTableModel) home_water.getModel();
97             model.setRowCount(0);
98         }
99         public void setMacroTable() {
100            int count=1;
101            try {
102                if (SessionManager.isLoggedIn()) {
103                    User currentUser = SessionManager.getCurrentUser();
104                    userId = currentUser.getId();
105                } else {
106                    JOptionPane.showMessageDialog(this, "Unsuccessful");
107                    // Handle not logged in
108                    return; // Exit the method if not logged in
109                }
110
111                String sql = "SELECT calories, kcalories, scalories, dcalories, lprotein, lprotein, sprotein, dprotein FROM macro WHERE id = ?";
112                try (PreparedStatement pst = con.prepareStatement(sql)) {
113                    pst.setInt(1, userId);
114                    model = (DefaultTableModel) home_macro.getModel();
115                    clearMacroTable();
116                    try (ResultSet rs = pst.executeQuery()) {
117                        while (rs.next()) {
118                            S_no= count;
119                            Total_Calories = rs.getFloat("calories") + rs.getFloat("kcalories") + rs.getFloat("scalories") + rs.getFloat("dcalories");
120                            Total_Protein = rs.getFloat("lprotein") + rs.getFloat("sprotein") + rs.getFloat("dprotein") + rs.getFloat("dprotein");
121
122                            Object[] obj = {S_no, Total_Calories, Total_Protein};
123                            model.addRow(obj);
124                            count++;
125                        }
126                    }
127                } catch (Exception e) {
128                    e.printStackTrace();
129                }
130            }
131            public void clearMacroTable() {
132                DefaultTableModel macroModel = (DefaultTableModel) home_macro.getModel();
133                macroModel.setRowCount(0);
134            }
135        /**
136         * This method is called from within the constructor to initialize the form.
137         * WARNING: Do NOT modify this code. The content of this method is always
138         * regenerated by the Form Editor.
139         */
140         $SuppressWarnings("unchecked")

```

Smart Macro Tracker

```
484     public void trackredirect(){
485         try{
486             TrackPage track1 = new TrackPage();
487             track1.setVisible(true);
488             this.dispose();
489         }catch(Exception e){
490             e.printStackTrace();
491         }
492     }
493     private void TracklabelMouseClicked(java.awt.event.MouseEvent evt) {
494         // TODO add your handling code here:
495         trackredirect();
496     }
497     private void h_analyzeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
504     private void h_bmiMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
511     private void h_dietMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
518     private void h_waterMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
525     private void jPanel12MouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
532
533     /*...3 lines*/
536     public static void main(String args[]) {...31 lines}
567
568
569     // Variables declaration - do not modify
570     private javax.swing.JPanel Tracklabel;
571     private javax.swing.JPanel h_analyze;
572     private javax.swing.JPanel h_bmi;
573     private javax.swing.JPanel h_diet;
574     private javax.swing.JTextArea home_bmi;
575     private rojeru_san.complementos.RSTableMetro home_macro;
576     private rojeru_san.complementos.RSTableMetro home_water;
577     private javax.swing.JLabel jLabel11;
578     private javax.swing.JLabel jLabel111;
579     private javax.swing.JLabel jLabel112;
580     private javax.swing.JLabel jLabel113;
581     private javax.swing.JLabel jLabel115;
582     private javax.swing.JLabel jLabel116;
583     private javax.swing.JLabel jLabel117;
584     private javax.swing.JLabel jLabel12;
585     private javax.swing.JLabel jLabel13;
586     private javax.swing.JLabel jLabel14;
587     private javax.swing.JLabel jLabel15;
588     private javax.swing.JLabel jLabel16;
589     private javax.swing.JLabel jLabel17;
590     private javax.swing.JLabel jLabel18;
591     private javax.swing.JLabel jLabel19;
592     private javax.swing.JPanel jPanel11;
593     private javax.swing.JPanel jPanel12;
594     private javax.swing.JPanel jPanel13;
595     private javax.swing.JPanel jPanel14;
596     private javax.swing.JPanel jPanel17;
597     private javax.swing.JPanel jPanel18;
598     private javax.swing.JPanel jPanel19;
599     private javax.swing.JPanel jPanel111;
600     private javax.swing.JPanel jPanel112;
601     private javax.swing.JPanel jPanel113;
602     private javax.swing.JPanel jPanel114;
603     private javax.swing.JPanel jPanel117;
604     private javax.swing.JPanel jPanel12;
605     private javax.swing.JPanel jPanel13;
606     private javax.swing.JPanel jPanel14;
607     private javax.swing.JPanel jPanel15;
608     private javax.swing.JPanel jPanel16;
609     private javax.swing.JScrollPane jScrollPane1;
610     private javax.swing.JScrollPane jScrollPane2;
611     private javax.swing.JScrollPane jScrollPane3;
612     // End of variables declaration
613 }
```

7.4. TrackPage.java

```

5  package JFrame;
6  import java.sql.Connection;
7  import java.sql.PreparedStatement;
8  import javax.swing.JOptionPane;
9  /**
10   * 
11   * @author chrisjordan
12   */
13  public class TrackPage extends javax.swing.JFrame {
14
15  /**
16   * Creates new form TrackPage
17   */
18  long breakfast_calories,breakfast_protein,lunch_calories,lunch_protein, snacks_calories,snacks_protein,dinner_calories,dinner_protein;
19  int userId;
20  public TrackPage() {
21      initComponents();
22  }
23  public boolean inserttrack(){
24      boolean isAdded = false;
25      breakfast_calories= Long.parseLong(b_calories.getText());
26      breakfast_protein= Long.parseLong(b_protein.getText());
27      lunch_calories= Long.parseLong(l_calories.getText());
28      lunch_protein= Long.parseLong(l_protein.getText());
29      snacks_calories= Long.parseLong(s_calories.getText());
30      snacks_protein= Long.parseLong(s_protein.getText());
31      dinner_calories= Long.parseLong(d_calories.getText());
32      dinner_protein= Long.parseLong(d_protein.getText());
33
34      try{
35          if (SessionManager.isLoggedIn()) {
36              User currentUser = SessionManager.getCurrentUser();
37              userId = currentUser.getId();
38          } else {
39              JOptionPane.showMessageDialog(this, "Unsuccessful");
40          }
41          Connection con = DBConnection.getConnection();
42          String sql = "insert into macro(id, bcalories,bprotein,lcalories,lprotein,sprotein,dcalories,dprotein) values(?,?,?,?,?,?,?,?)";
43          //String sql = "update macro set bcalories=?, bprotein=?, lcalories=?, lprotein=?, sprotein=?, dcalories=?, dprotein=? where id = ?";
44          PreparedStatement pst = con.prepareStatement(sql);
45          pst.setInt(1,userId);
46          pst.setLong(2,breakfast_calories);
47          pst.setLong(3,breakfast_protein);
48          pst.setLong(4,lunch_calories);
49          pst.setLong(5,lunch_protein);
50          pst.setLong(6,snacks_calories);
51          pst.setLong(7,snacks_protein);
52          pst.setLong(8,dinner_calories);
53          pst.setLong(9,dinner_protein);
54
55          int rowCount = pst.executeUpdate();
56          if (rowCount > 0) {
57              isAdded= true;
58          } else {
59              isAdded= false;
60          }
61      } catch (Exception e) {
62          e.printStackTrace();
63          isAdded = false;
64      }
65
66      return isAdded;
67  }
68  public boolean updatetrack(){
69      boolean isUpdated = false;
70      breakfast_calories= Integer.parseInt(b_calories.getText());
71      breakfast_protein= Integer.parseInt(b_protein.getText());
72      lunch_calories= Integer.parseInt(l_calories.getText());
73      lunch_protein= Integer.parseInt(l_protein.getText());
74
75      snacks_protein= Integer.parseInt(s_protein.getText());
76      dinner_calories= Integer.parseInt(d_calories.getText());
77      dinner_protein= Integer.parseInt(d_protein.getText());
78
79      try{
80          if (SessionManager.isLoggedIn()) {
81              User currentUser = SessionManager.getCurrentUser();
82              userId = currentUser.getId();
83          } else {
84              JOptionPane.showMessageDialog(this, "Unsuccessful");
85          }
86          Connection con = DBConnection.getConnection();
87          //String sql = "insert into nutrition(calories,bprotein,lcalories,lprotein,scalories,sprotein,dcalories,dprotein) values(?,?,?,?,?,?,?,?)";
88          String sql = "update nutrition set bcalories=?, bprotein=?, lcalories=?, lprotein=?, scalaries=?, sprotein=?, dcalories=?, dprotein=? where id = ?";
89          PreparedStatement pst = con.prepareStatement(sql);
90          pst.setLong(1,breakfast_calories);
91          pst.setLong(2,breakfast_protein);
92          pst.setLong(3,lunch_calories);
93          pst.setLong(4,lunch_protein);
94          pst.setLong(5,snacks_calories);
95          pst.setLong(6,snacks_protein);
96          pst.setLong(7,dinner_calories);
97          pst.setLong(8,dinner_protein);
98          pst.setLong(9,userId);
99
100         int rowCount = pst.executeUpdate();
101        if (rowCount > 0) {
102            isUpdated= true;
103        } else {
104            isUpdated= false;
105        }
106    } catch (Exception e) {
107        e.printStackTrace();
108        isUpdated = false;
109    }
110}

```

Smart Macro Tracker

```
132     public void homeredirect(){
133         try{
134             HomePage home1 = new HomePage();
135             home1.setVisible(true);
136             this.dispose();
137         }catch(Exception e){
138             e.printStackTrace();
139         }
140     }
141     //update
142     /**
143      * This method is called from within the constructor to initialize the form.
144      * WARNING: Do NOT modify this code. The content of this method is always
145      * regenerated by the Form Editor.
146      */
147     @SuppressWarnings("unchecked")
148     // Generated Code
149
150     private void t_update1ActionPerformed(java.awt.event.ActionEvent evt) {
151         // TODO add your handling code here:
152         if (updatetrack() == true){
153             JOptionPane.showMessageDialog(this,"Update successful!!!");
154         }
155         else{
156             JOptionPane.showMessageDialog(this,"Update unsuccessful :(");
157         }
158     }
159
160     private void t_insert1ActionPerformed(java.awt.event.ActionEvent evt) {
161         // TODO add your handling code here:
162         if (insertrtrack() == true){
163             JOptionPane.showMessageDialog(this,"Insert successful!!!");
164         }
165         else{
166             JOptionPane.showMessageDialog(this,"Insert unsuccessful :(");
167         }
168     }
169
170     private void TracklabelMouseClicked(java.awt.event.MouseEvent evt) {
171         // TODO add your handling code here:
172     }
173
174     private void t_analyzeMouseClicked(java.awt.event.MouseEvent evt) [....6 lines]
175
176     private void jPanel5MouseClicked(java.awt.event.MouseEvent evt) [....6 lines]
177
178     private void jPanel6MouseClicked(java.awt.event.MouseEvent evt) [....6 lines]
179
180     private void jPanel7MouseClicked(java.awt.event.MouseEvent evt) [....6 lines]
181
182     private void jPanel9MouseClicked(java.awt.event.MouseEvent evt) [....6 lines]
183
184     private void jPanel11MouseClicked(java.awt.event.MouseEvent evt) [....6 lines]
185
186
187     /*....3 lines */
188     public static void main(String args[]) [....31 lines]
189
190
191     // Variables declaration - do not modify
192     private javax.swing.JPanel Tracklabel;
193     private javax.swing.JTextField b_calories;
194     private javax.swing.JTextField b_protein;
195     private javax.swing.JTextField d_calories;
196     private javax.swing.JTextField d_protein;
197     private javax.swing.JLabel jLabel1;
198     private javax.swing.JLabel jLabel12;
199     private javax.swing.JLabel jLabel13;
200     private javax.swing.JLabel jLabel14;
201     private javax.swing.JLabel jLabel15;
202     private javax.swing.JLabel jLabel16;
203     private javax.swing.JLabel jLabel17;
204     private javax.swing.JLabel jLabel18;
```

```
825     private javax.swing.JLabel jLabel130;
826     private javax.swing.JLabel jLabel131;
827     private javax.swing.JLabel jLabel132;
828     private javax.swing.JLabel jLabel133;
829     private javax.swing.JLabel jLabel134;
830     private javax.swing.JLabel jLabel135;
831     private javax.swing.JLabel jLabel136;
832     private javax.swing.JLabel jLabel14;
833     private javax.swing.JLabel jLabel15;
834     private javax.swing.JLabel jLabel16;
835     private javax.swing.JLabel jLabel17;
836     private javax.swing.JLabel jLabel18;
837     private javax.swing.JLabel jLabel19;
838     private javax.swing.JPanel jPanel1;
839     private javax.swing.JPanel jPanel11;
840     private javax.swing.JPanel jPanel12;
841     private javax.swing.JPanel jPanel13;
842     private javax.swing.JPanel jPanel14;
843     private javax.swing.JPanel jPanel15;
844     private javax.swing.JPanel jPanel16;
845     private javax.swing.JPanel jPanel17;
846     private javax.swing.JPanel jPanel18;
847     private javax.swing.JPanel jPanel19;
848     private javax.swing.JPanel jPanel16;
849     private javax.swing.JPanel jPanel17;
850     private javax.swing.JPanel jPanel18;
851     private javax.swing.JTextField l_calories;
852     private javax.swing.JTextField l_protein;
853     private javax.swing.JTextField s_calories;
854     private javax.swing.JTextField s_protein;
855     private javax.swing.JPanel t_analyze;
856     private javax.swing.JButton t_insert1;
857     private javax.swing.JButton t_update1;
858
859 } // End of variables declaration
```

7.5. AnalyzePage.java

```

5  package JFrame;
6  import static JFrame.DBConnection.con;
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9  import java.sql.PreparedStatement;
10 import javax.swing.JOptionPane;
11 import javax.swing.table.DefaultTableModel;
12 import java.sql.*;
13 /**
14  *
15  * @author chrisjordan
16  */
17 public class AnalyzePage extends javax.swing.JFrame {
18
19 /**
20  * Creates new form TrackPage
21  */
22 long breakfast_calories,breakfast_protein,lunch_calories,lunch_protein, snacks_calories,snacks_protein,dinner_calories,dinner_protein;
23 int userId,counter,s_no;
24 DefaultTableModel model;
25 public AnalyzePage() {
26     initComponents();
27     setAnalyzeTable();
28     calculateAndDisplayTotal();
29 }
30 public void setAnalyzeTable(){
31     int counter = 1;
32     try{
33         if (SessionManager.isLoggedIn()) {
34             User currentUser = SessionManager.getCurrentUser();
35             userId = currentUser.getId();
36         } else {
37             JOptionPane.showMessageDialog(this, "unsuccessful");
38         }
39     } catch (Exception e){
40         e.printStackTrace();
41     }
42     Class.forName("com.mysql.jdbc.Driver");
43     con = DriverManager.getConnection("jdbc:mysql://localhost:3306/test1","root","christ123");
44     //Statement st = con.createStatement();
45     //ResultSet rs = st.executeQuery("select bcalories,bprotein, lcalories, lprotein, scalories, sprotein, dcalories, dprotein from
46     //pst.setInt(1,userId);
47     String sql = "SELECT bcalories, bprotein, lcalories, lprotein, scalories, sprotein, dcalories, dprotein FROM macro WHERE id=?";
48     PreparedStatement pst = con.prepareStatement(sql) ;
49     pst.setInt(1,userId);
50     clearTable();
51     ResultSet rs = pst.executeQuery();
52     while (rs.next()){
53         S_no = counter;
54         breakfast_calories = rs.getLong("bcalories");
55         breakfast_protein = rs.getLong("bprotein");
56         lunch_calories = rs.getLong("lcalories");
57         lunch_protein = rs.getLong("lprotein");
58         snacks_calories = rs.getLong("scalories");
59         snacks_protein = rs.getLong("sprotein");
60         dinner_calories = rs.getLong("dcalories");
61         dinner_protein = rs.getLong("dprotein");
62         Object[] obj = {S_no,breakfast_calories,breakfast_protein,lunch_calories,lunch_protein,snacks_calories,
63                         snacks_protein,dinner_calories,dinner_protein };
64         model = (DefaultTableModel)tbl_analyze.getModel();
65         model.addRow(obj);
66         counter++;
67     }
68 } catch (Exception e){
69     e.printStackTrace();
70 }
71 public void clearTable(){
72     DefaultTableModel model = (DefaultTableModel)tbl_analyze.getModel();
73     model.setRowCount(0);
74 }
75 private void calculateAndDisplayTotal() {
76
77     // Iterate through the table rows
78     for (int i = 0; i < tbl_analyze.getRowCount(); i++) {
79         long sumCalories = 0;
80         long sumProtein = 0;
81
82         // Calculate the sum of bcalories, lcalories, scalories, and dcalories for each row
83         sumCalories += (long)tbl_analyze.getValueAt(i, 1); // breakfast_calories
84         sumCalories += (long)tbl_analyze.getValueAt(i, 3); // lunch_calories
85         sumCalories += (long)tbl_analyze.getValueAt(i, 5); // snacks_calories
86         sumCalories += (long)tbl_analyze.getValueAt(i, 7); // dinner_calories
87
88         // Calculate the sum of bprotein, lprotein, sprotein, and dprotein for each row
89         sumProtein += (long)tbl_analyze.getValueAt(i, 2); // breakfast_protein
90         sumProtein += (long)tbl_analyze.getValueAt(i, 4); // lunch_protein
91         sumProtein += (long)tbl_analyze.getValueAt(i, 6); // snacks_protein
92         sumProtein += (long)tbl_analyze.getValueAt(i, 8); // dinner_protein
93
94         // Set the total protein in the appropriate column
95         tbl_analyze.setValueAt(sumCalories, i, 9);
96         tbl_analyze.setValueAt(sumProtein, i, 10); // Update "total_protein" column
97     }
98 }
99 private void compareAndSetRemarks() {
100     // Fetch the value from the "set limits" field
101     long dailyLimit = Long.parseLong(a_limit.getText());
102
103     // Iterate through the table rows
104     for (int i = 0; i < tbl_analyze.getRowCount(); i++) {
105         long sumCalories = 0;
106         long sumProtein = 0;
107
108         // Calculate the sum of bcalories, lcalories, scalories, and dcalories for each row
109     }
110 }
111 }

```

Smart Macro Tracker

```
109     sumCalories += (long) tbl_analyze.getValueAt(i, 3); // lunch_calories
110     sumCalories += (long) tbl_analyze.getValueAt(i, 5); // snacks_calories
111     sumCalories += (long) tbl_analyze.getValueAt(i, 7); // dinner_calories
112
113     // Calculate the sum of bprotein, lprotein, sprotein, and dprotein for each row
114     sumProtein += (long) tbl_analyze.getValueAt(i, 2); // breakfast_protein
115     sumProtein += (long) tbl_analyze.getValueAt(i, 4); // lunch_protein
116     sumProtein += (long) tbl_analyze.getValueAt(i, 6); // snacks_protein
117     sumProtein += (long) tbl_analyze.getValueAt(i, 8); // dinner_protein
118
119     // Compare the calculated sum with the daily limit
120     if (sumCalories < dailyLimit) {
121         tbl_analyze.setValueAt("SubOptimal", i, 11); // Update "remarks" column to "Less"
122     } else if (sumCalories == dailyLimit) {
123         tbl_analyze.setValueAt("Optimal", i, 11); // Update "remarks" column to "Perfect"
124     } else {
125         tbl_analyze.setValueAt("Excessive", i, 11); // Update "remarks" column to "Exceed"
126     }
127
128     // Set the total protein in the appropriate column
129     tbl_analyze.setValueAt(sumCalories, i, 9);
130     tbl_analyze.setValueAt(sumProtein, i, 10); // Update "total_protein" column
131 }
132
133
134     public void homeredirect(){
135         try{
136             HomePage hme1 = new HomePage();
137             hme1.setVisible(true);
138             this.dispose();
139         }catch(Exception e){
140             e.printStackTrace();
141         }
142     }
143 //update
144 /**
145 * @SuppressWarnings("unchecked")
146 */
147 Generated Code
360
148     private void a_limitActionPerformed(java.awt.event.ActionEvent evt) {
149         // TODO add your handling code here:
150     }
151
152     private void a_backActionPerformed(java.awt.event.ActionEvent evt) {
153         // TODO add your handling code here:
154         HomePage h = new HomePage();
155         h.setVisible(true);
156         this.dispose();
157     }
158
159     private void analyze_butActionPerformed(java.awt.event.ActionEvent evt) {
160         // TODO add your handling code here:
161         compareAndSetRemarks();
162     }
163
164 /**
165 * @param args the command line arguments
166 */
167 public static void main(String args[] ) {
168     /* Set the Nimbus look and feel */
169     /* Look and feel setting code (optional)
170      */
171
172     /* Create and display the form */
173     java.awt.EventQueue.invokeLater(new Runnable() {
174         public void run() {
175             new AnalyzePage().setVisible(true);
176         }
177     });
178 }
179
180
181 // Variables declaration - do not modify
182 private javax.swing.JButton a_back;
183 private javax.swing.JTextField a_limit;
184 private javax.swing.JButton analyze_but;
185 private javax.swing.JLabel jLabel13;
186 private javax.swing.JLabel jLabel14;
187 private javax.swing.JLabel jLabel15;
188 private javax.swing.JLabel jLabel16;
189 private javax.swing.JLabel jLabel17;
190 private javax.swing.JLabel jLabel18;
191 private javax.swing.JLabel jLabel19;
192 private javax.swing.JPanel jPanel1;
193 private javax.swing.JPanel jPanel13;
194 private javax.swing.JPanel jPanel14;
195 private javax.swing.JPanel jPanel15;
196 private javax.swing.JPanel jPanel16;
197 private javax.swing.JPanel jPanel17;
198 private javax.swing.JPanel jPanel18;
199 private javax.swing.JPanel jPanel19;
200 private javax.swing.JPanel jPanel11;
201 private javax.swing.JPanel jPanel12;
202 private javax.swing.JScrollPane jScrollPane1;
203 private rojelu_san.complementos.RSTableMetro tbl_analyze;
204 // End of variables declaration
205 }
```

7.6. BMIPage.java

```

5  package JFrame;
6  import static JFrame.DBConnection.con;
7  import javax.swing.JOptionPane;
8  import java.sql.*;
9
10 /**
11  * 
12  * @author chrisjordan
13  */
14 public class BMIPage extends javax.swing.JFrame {
15
16 /**
17  * Creates new form TrackPage
18  */
19 int userId;
20 public BMIPage() {
21     initComponents();
22 }
23 // method to compute BMI and display result
24 public void computeAndDisplayBMI() {
25     try {
26         // Get user input for height and weight
27         float height = Float.parseFloat(txt_height.getText());
28         float weight = Float.parseFloat(txt_weight.getText());
29         // Compute BMI
30         float bmi = calculateBMI(height, weight);
31
32         // Display the BMI result in a JTextArea
33         txt_BMI.setText("      "+bmi"      ");
34         String remarks = getRemarks(bmi);
35         txt_BMI_remarks.setText("Remarks: " + remarks);
36
37     } catch (NumberFormatException e) {
38         JOptionPane.showMessageDialog(this, "Please enter valid numeric values for height and weight.");
39     }
40 }
41 // method to calculate BMI
42 private float calculateBMI(float height, float weight) {
43     // BMI formula: weight (kg) / (height (m) * height (m))
44     return weight / (height * height);
45 }
46 private String getRemarks(float bmi) {
47     if (bmi < 18.5) {
48         return "You are Underweight";
49     } else if (bmi >= 18.5 & bmi <= 24.9) {
50         return "You are Normal";
51     } else if (bmi >= 25 & bmi <= 29.9) {
52         return "You are Overweight";
53     } else {
54         return "You are Obese";
55     }
56 }
57 private void uploadBMIToDatabase(float bmi) {
58     try{
59         if (SessionManager.isLoggedIn()) {
60             User currentUser = SessionManager.getCurrentUser();
61             userId = currentUser.getId();
62         } else {
63             JOptionPane.showMessageDialog(this, "unsuccessful");
64         }
65         // Handle not logged in
66
67         // Establish database connection (update with your connection
68         Connection con = DBConnection.getConnection();
69         // Insert BMI into the database
70         String sql = "INSERT INTO bmi (id, bmi_value) VALUES (?,?)";
71         PreparedStatement pst = con.prepareStatement(sql);
72         pst.setInt(1,userId);
73         pst.setFloat(2,bmi);
74         // Execute the insert query
75         int rowsInserted = pst.executeUpdate();
76     } catch (SQLException e) {
77         if (rowsInserted > 0) {
78             JOptionPane.showMessageDialog(this, "BMI uploaded to the database successfully.");
79         } else {
80             JOptionPane.showMessageDialog(this, "Failed to upload BMI to the database.");
81         }
82     } catch (Exception e) {
83         JOptionPane.showMessageDialog(this, "Error uploading BMI to the database:\n" + e.getMessage());
84     }
85 }
86 private void saveBMIToDatabase() {
87     try {
88         // Get the computed BMI from the JTextArea
89         String bmiString = txt_BMI.getText().replace("      ", " ");
90         float bmi = Float.parseFloat(bmiString);
91
92         // Upload BMI to the database
93         uploadBMIToDatabase(bmi);
94
95     } catch (NumberFormatException e) {
96         JOptionPane.showMessageDialog(this, "Please compute BMI first.");
97     }
98 }
99 //FOR HOMEPAGE DISPLAY
100 public class DatabaseHandler {
101     private BMICallback callback;
102     public DatabaseHandler(BMICallback callback) {
103         this.callback = callback;
104     }
105
106     // Method to get the last inserted BMI value
107     public double getLastInsertedBMI() {
108         double lastBMI = 0.0;
109
110         try (Connection con = DBConnection.getConnection();
111              Statement stmt = con.createStatement();) {
112             String sql = "SELECT bmi_value FROM bmi ORDER BY bmi_id DESC LIMIT 1";
113             ResultSet rs = stmt.executeQuery(sql);
114
115             if (rs.next()) {
116                 lastBMI = rs.getDouble("bmi_value");
117             }
118         } catch (SQLException e) {
119             e.printStackTrace();
120         }
121
122         return lastBMI;
123     }
124
125     public interface BMICallback {
126         void updateBMI(double lastBMI);
127     }
128 }
129
130 //update |
131 /**
132  * This method is called from within the constructor to initialize the form.
133  * WARNING: Do NOT modify this code. The content of this method is always
134  * regenerated by the Form Editor.
135  */
136 @SuppressWarnings("unchecked")
137 Generated Code
138
139
140     private void a_trackMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
141     private void a_homeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
142     private void b_computeActionPerformed(java.awt.event.ActionEvent evt) {...4 lines}
143     private void bmi_databaseActionPerformed(java.awt.event.ActionEvent evt) {...4 lines}
144     private void jPanelBMouseClicked(java.awt.event.MouseEvent evt) {...}
145     private void jPanelBMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
146     private void a_dietMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
147     private void a_waterMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
148     private void a_logoutMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
149
150     /**
151      * @param args
152     */
153     public static void main(String args[]) {...34 lines}
154
155     // Variables declaration - do not modify
156     private javax.swing.JPanel a_bmi;
157     private javax.swing.JPanel a_diet;
158     private javax.swing.JPanel a_home;
159     private javax.swing.JPanel a_logout;
160     private javax.swing.JPanel a_track;
161     private javax.swing.JButton b_compute;
162     private javax.swing.JLabel jLabel1;
163     private javax.swing.JLabel jLabel11;
164     private javax.swing.JLabel jLabel12;
165     private javax.swing.JLabel jLabel13;
166     private javax.swing.JLabel jLabel14;
167     private javax.swing.JLabel jLabel15;
168     private javax.swing.JLabel jLabel16;
169     private javax.swing.JLabel jLabel17;
170     private javax.swing.JLabel jLabel18;
171     private javax.swing.JLabel jLabel19;
172     private javax.swing.JLabel jLabel2;
173     private javax.swing.JLabel jLabel21;
174     private javax.swing.JLabel jLabel22;
175     private javax.swing.JLabel jLabel23;
176     private javax.swing.JLabel jLabel24;
177     private javax.swing.JLabel jLabel25;
178     private javax.swing.JLabel jLabel26;
179     private javax.swing.JLabel jLabel27;
180     private javax.swing.JLabel jLabel28;
181     private javax.swing.JLabel jLabel29;
182     private javax.swing.JLabel jLabel3;
183     private javax.swing.JLabel jLabel30;
184     private javax.swing.JLabel jLabel31;
185     private javax.swing.JLabel jLabel32;
186     private javax.swing.JLabel jLabel33;
187     private javax.swing.JLabel jLabel34;
188     private javax.swing.JLabel jLabel35;
189     private javax.swing.JLabel jLabel36;
190     private javax.swing.JLabel jLabel37;
191     private javax.swing.JLabel jLabel38;
192     private javax.swing.JLabel jLabel39;
193     private javax.swing.JLabel jLabel4;
194     private javax.swing.JLabel jLabel40;
195     private javax.swing.JLabel jLabel41;
196     private javax.swing.JLabel jLabel42;
197     private javax.swing.JLabel jLabel43;
198     private javax.swing.JLabel jLabel44;
199     private javax.swing.JLabel jLabel45;
200     private javax.swing.JLabel jLabel46;
201     private javax.swing.JLabel jLabel47;
202     private javax.swing.JLabel jLabel48;
203     private javax.swing.JLabel jLabel49;
204     private javax.swing.JLabel jLabel5;
205     private javax.swing.JLabel jLabel50;
206     private javax.swing.JLabel jLabel51;
207     private javax.swing.JLabel jLabel52;
208     private javax.swing.JLabel jLabel53;
209     private javax.swing.JLabel jLabel54;
210     private javax.swing.JLabel jLabel55;
211     private javax.swing.JLabel jLabel56;
212     private javax.swing.JLabel jLabel57;
213     private javax.swing.JLabel jLabel58;
214     private javax.swing.JLabel jLabel59;
215     private javax.swing.JLabel jLabel6;
216     private javax.swing.JLabel jLabel60;
217     private javax.swing.JLabel jLabel61;
218     private javax.swing.JLabel jLabel62;
219     private javax.swing.JLabel jLabel63;
220     private javax.swing.JLabel jLabel64;
221     private javax.swing.JLabel jLabel65;
222     private javax.swing.JLabel jLabel66;
223     private javax.swing.JLabel jLabel67;
224     private javax.swing.JLabel jLabel68;
225     private javax.swing.JLabel jLabel69;
226     private javax.swing.JLabel jLabel60;
227     private javax.swing.JLabel jLabel61;
228     private javax.swing.JLabel jLabel62;
229     private javax.swing.JLabel jLabel63;
230     private javax.swing.JLabel jLabel64;
231     private javax.swing.JLabel jLabel65;
232     private javax.swing.JLabel jLabel66;
233     private javax.swing.JLabel jLabel67;
234     private javax.swing.JLabel jLabel68;
235     private javax.swing.JLabel jLabel69;
236     private javax.swing.JLabel jLabel70;
237     private javax.swing.JLabel jLabel71;
238     private javax.swing.JLabel jLabel72;
239     private javax.swing.JLabel jLabel73;
240     private javax.swing.JLabel jLabel74;
241     private javax.swing.JLabel jLabel75;
242     private javax.swing.JLabel jLabel76;
243     private javax.swing.JLabel jLabel77;
244     private javax.swing.JLabel jLabel78;
245     private javax.swing.JLabel jLabel79;
246     private javax.swing.JLabel jLabel70;
247     private javax.swing.JLabel jLabel71;
248     private javax.swing.JLabel jLabel72;
249     private javax.swing.JLabel jLabel73;
250     private javax.swing.JLabel jLabel74;
251     private javax.swing.JLabel jLabel75;
252     private javax.swing.JLabel jLabel76;
253     private javax.swing.JLabel jLabel77;
254     private javax.swing.JLabel jLabel78;
255     private javax.swing.JLabel jLabel79;
256     private javax.swing.JScrollPane jScrollPane1;
257     private javax.swing.JTextArea txt_BMI;
258     private javax.swing.JTextArea txt_BMI_remarks;
259     private javax.swing.JTextField txt_height;
260     private javax.swing.JTextField txt_weight;
261
262     // End of variables declaration
263 }
```

7.7. DietPlanPage.java

```

5  package JFrame;
6  import static JFrame.DBConnection.conn;
7  import javax.swing.JOptionPane;
8  import java.sql.*;
9
10 /**
11  *
12  * @author chrisjordan
13  */
14 public class DietPlanPage extends javax.swing.JFrame {
15
16 /**
17  * Creates new form TrackPage
18 */
19
20 public DietPlanPage() {
21     initComponents();
22 }
23 @SuppressWarnings("unchecked")
24 Generated Code
25
26 private void a_trackMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
27
28 private void a_homeMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
29
30 private void deficit_EXPLORE_buttonActionPerformed(java.awt.event.ActionEvent evt) {...6 lines}
31
32 private void recomp_EXPLORE_button1ActionPerformed(java.awt.event.ActionEvent evt) {...6 lines}
33
34 private void surplus_EXPLORE_button2ActionPerformed(java.awt.event.ActionEvent evt) {...6 lines}
35
36 private void jPanel8MouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
37
38 private void a_bmiMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
39
40 private void a_waterMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
41
42 private void a_logoutMouseClicked(java.awt.event.MouseEvent evt) {...6 lines}
43 // CARD 2 [204,204,255], CARD 3 [255,102,102]
44 /**
45  */
46 public static void main(String args[]) {
47     /* Set the Nimbus look and feel */
48     Look and feel setting code (optional)
49     //</editor-fold>
50     //</editor-fold>
51     //</editor-fold>
52     //</editor-fold>
53     //</editor-fold>
54     //</editor-fold>
55     //</editor-fold>
56
57     /* Create and display the form */
58     java.awt.EventQueue.invokeLater(new Runnable() {...5 lines});
59 }
60
61 // Variables declaration - do not modify
62 private javax.swing.JPanel a_bmi;
63 private javax.swing.JPanel a_diet;
64 private javax.swing.JPanel a_home;
65 private javax.swing.JPanel a_logout;
66 private javax.swing.JPanel a_track;
67 private javax.swing.JButton deficit_EXPLORE_button;
68 private javax.swing.JLabel jLabel1;
69 private javax.swing.JLabel jLabel11;
70 private javax.swing.JLabel jLabel12;
71 private javax.swing.JLabel jLabel13;
72 private javax.swing.JLabel jLabel14;
73 private javax.swing.JLabel jLabel16;
74 private javax.swing.JLabel jLabel17;
75 private javax.swing.JLabel jLabel18;
76 private javax.swing.JLabel jLabel19;
77 private javax.swing.JLabel jLabel2;
78 private javax.swing.JPanel jPanel1;
79 private javax.swing.JPanel jPanel2;
80 private javax.swing.JPanel jPanel3;
81 private javax.swing.JPanel jPanel4;
82 private javax.swing.JPanel jPanel5;
83 private javax.swing.JPanel jPanel6;
84 private javax.swing.JPanel jPanel7;
85 private javax.swing.JPanel jPanel8;
86 private javax.swing.JButton recomp_EXPLORE_button1;
87 private javax.swing.JButton surplus_EXPLORE_button2;
88 // End of variables declaration
89 }

```

7.7.1. CaloricDeficitPage.java

```

5  package JFrame;
6  /**
7   * 
8   * @author chrisjordan
9   */
10  public class CaloricDeficitPage extends javax.swing.JFrame {
11      /**
12       * 
13       */
14      @SuppressWarnings("unchecked")
15      /**
16       * Generated Code
17       */
18      private void dietD_backActionPerformed(java.awt.event.ActionEvent evt) {
19          /**
20           * TODO add your handling code here:
21           */
22          DietPlanPage d1 = new DietPlanPage();
23          d1.setVisible(true);
24          this.dispose();
25      }
26      /**
27       * ****.3 lines */
28      public static void main(String args[]) {...34 lines...}
29
30      // Variables declaration - do not modify
31      private javax.swing.JButton dietD_back;
32      private javax.swing.JLabel jLabel1;
33      private javax.swing.JLabel jLabel11;
34      private javax.swing.JLabel jLabel12;
35      private javax.swing.JLabel jLabel13;
36      private javax.swing.JPanel jPanel1;
37      private javax.swing.JPanel jPanel11;
38      private javax.swing.JPanel jPanel12;
39      private javax.swing.JPanel jPanel13;
40      // End of variables declaration
41  }
42

```

7.7.2. BodyRecompPage.java

```

5  package JFrame;
6  /**
7   * 
8   * @author chrisjordan
9   */
10  public class BodyRecompPage extends javax.swing.JFrame {
11      /**
12       * 
13       */
14      @SuppressWarnings("unchecked")
15      /**
16       * Generated Code
17       */
18      private void DietR_backActionPerformed(java.awt.event.ActionEvent evt) {...6 lines...}
19      /**
20       * ****.3 lines */
21      public static void main(String args[]) {...39 lines...}
22
23      // Variables declaration - do not modify
24      private javax.swing.JButton DietR_back;
25      private javax.swing.JLabel jLabel1;
26      private javax.swing.JLabel jLabel19;
27      private javax.swing.JLabel jLabel2;
28      private javax.swing.JLabel jLabel3;
29      private javax.swing.JPanel jPanel1;
30      private javax.swing.JPanel jPanel11;
31      private javax.swing.JPanel jPanel12;
32      private javax.swing.JPanel jPanel13;
33      // End of variables declaration
34  }
35

```

7.7.3. CaloricSurplusPage.java

```

5  package JFrame;
6  /**
7   * 
8   * @author chrisjordan
9   */
10  public class CaloricSurplusPage extends javax.swing.JFrame {
11      /**
12       * 
13       */
14      @SuppressWarnings("unchecked")
15      /**
16       * Generated Code
17       */
18      private void dietsS_backActionPerformed(java.awt.event.ActionEvent evt) {...7 lines...}
19      /**
20       * ****.3 lines */
21      public static void main(String args[]) {...46 lines...}
22
23      // Variables declaration - do not modify
24      private javax.swing.JButton dietsS_back;
25      private javax.swing.JLabel jLabel1;
26      private javax.swing.JLabel jLabel11;
27      private javax.swing.JLabel jLabel12;
28      private javax.swing.JLabel jLabel13;
29      private javax.swing.JPanel jPanel1;
30      private javax.swing.JPanel jPanel11;
31      private javax.swing.JPanel jPanel12;
32      // End of variables declaration
33  }
34

```

7.8. WaterTrackerPage.java

```

5  package JFrame;
6  import static JFrame.DBConnection.con;
7  import javax.swing.JOptionPane;
8  import java.sql.*;
9  import javax.swing.table.DefaultTableModel;
10 
11 /**
12  * @author chrisjordan
13  */
14 public class WaterTracker extends javax.swing.JFrame {
15 
16     /**
17      * Creates new form TrackPage
18     */
19     int S_no, userId; float Consumed;
20     DefaultTableModel model;
21     public WaterTracker() {
22         initComponents();
23         setWaterTable();
24     }
25     public boolean insertWater(){
26         boolean isAdded = false;
27         Consumed= Float.parseFloat(water_textfield.getText());
28 
29         try{
30             if (SessionManager.isLoggedIn()) {
31                 User currentUser = SessionManager.getCurrentUser();
32                 userId = currentUser.getId();
33             } else {
34                 JOptionPane.showMessageDialog(this, "unsuccessful");
35             }
36             Connection con = DBConnection.getConnection();
37             String sql = "insert into water_tracker(id, consumed) values(?,?)";
38 
39             //String sql = "update macro set bcalories=? , bprotein=? , lcalories=? , lprotein=? , scalories=?";
40             PreparedStatement pst = con.prepareStatement(sql);
41             pst.setFloat(1, userId);
42             pst.setFloat(2,Consumed);
43 
44             int rowCount = pst.executeUpdate();
45             if (rowCount > 0) {
46                 isAdded= true;
47             } else {
48                 isAdded= false;
49             }
50 
51         } catch (Exception e) {
52             e.printStackTrace(System.out);
53             isAdded = false;
54         }
55 
56         return isAdded;
57     }
58     public boolean updatewater(){
59         boolean isUpdated = false;
60         Consumed= Float.parseFloat(water_textfield.getText());
61         try{
62             if (SessionManager.isLoggedIn()) {
63                 User currentUser = SessionManager.getCurrentUser();
64                 userId = currentUser.getId();
65             } else {
66                 JOptionPane.showMessageDialog(this, "Unsuccessful");
67             }
68             Connection con = DBConnection.getConnection();
69             //String sql = "update water_tracker set consumed=? where id = ?";
70             String sql = "UPDATE water_tracker SET consumed = ? WHERE id = ? ORDER BY wid DESC LIMIT 1";
71             PreparedStatement pst = con.prepareStatement(sql);
72             pst.setFloat(1,Consumed);
73         }
74 
75         int rowCount = pst.executeUpdate();
76         if (rowCount > 0) {
77             isUpdated= true;
78         } else {
79             isUpdated= false;
80         }
81 
82     } catch (Exception e) {
83         e.printStackTrace(System.out);
84         isUpdated = false;
85     }
86 
87     return isUpdated;
88 }
89 
90 public void setWaterTable() {
91     int counter = 1;
92 
93     try {
94         if (SessionManager.isLoggedIn()) {
95             User currentUser = SessionManager.getCurrentUser();
96             userId = currentUser.getId();
97         } else {
98             JOptionPane.showMessageDialog(this, "Unsuccessful");
99             // Handle not logged in
100            return; // Exit the method if not logged in
101        }
102 
103        Connection con = DBConnection.getConnection();
104        String sql = "SELECT consumed FROM water_tracker WHERE id = ?";
105        try (PreparedStatement pst = con.prepareStatement(sql)) {
106            pst.setInt(1, userId);
107 
108            clearTable();
109 
110            try (ResultSet rs = pst.executeQuery()) {
111                while (rs.next()) {
112                    S_no = counter;
113                    Consumed = rs.getFloat("consumed");
114                    Object[] obj = {S_no, Consumed};
115                    model = (DefaultTableModel) water_table.getModel();
116                    model.addRow(obj);
117                    counter++;
118                }
119            }
120        } catch (Exception e) {
121            e.printStackTrace();
122        }
123    }
124 
125    public void clearTable(){...4 lines}
126    // method to compute BMI and display result
127    @SuppressWarnings("unchecked")
128    @Generated Code
129    private void a_trackMouseClicked(java.awt.event.MouseEvent evt){...6 lines}
130    private void a_homeMouseClicked(java.awt.event.MouseEvent evt){...6 lines}
131    private void water_textfieldActionPerformed(java.awt.event.ActionEvent evt){...3 lines}
132    private void WATER_ADDActionPerformed(java.awt.event.ActionEvent evt){...13 lines}
133    private void UPDATE_WATERActionPerformed(java.awt.event.ActionEvent evt){...11 lines}
134    private void JPanelMouseClicked(java.awt.event.MouseEvent evt){...6 lines}
135    private void a_bmiMouseClicked(java.awt.event.MouseEvent evt){...6 lines}
136 
137    public static void main(String args[]){...38 lines}
138 
139    // Variables Declaration - do not modify
140    private javax.swing.JButton UPDATE_WATER;
141    private javax.swing.JButton WATER_ADD;
142    private javax.swing.JPanel a_bmi;
143    private javax.swing.JPanel a_diet;
144    private javax.swing.JPanel a_home;
145    private javax.swing.JPanel a_logout;
146    private javax.swing.JPanel a_track;
147    private javax.swing.JPanel a_water;
148    private javax.swing.JLabel jLabel1;
149    private javax.swing.JLabel jLabel11;
150    private javax.swing.JLabel jLabel12;
151    private javax.swing.JLabel jLabel13;
152    private javax.swing.JLabel jLabel14;
153    private javax.swing.JLabel jLabel15;
154    private javax.swing.JLabel jLabel16;
155    private javax.swing.JLabel jLabel17;
156    private javax.swing.JLabel jLabel18;
157    private javax.swing.JLabel jLabel19;
158    private javax.swing.JPanel jPanel1;
159    private javax.swing.JPanel jPanel12;
160    private javax.swing.JPanel jPanel13;
161    private javax.swing.JPanel jPanel14;
162    private javax.swing.JPanel jPanel18;
163    private javax.swing.JScrollPane jScrollPane;
164    private rojeru_san.complementos.RSTableMetro water_table;
165    private javax.swing.JTextField water_textfield;
166 
167    // End of variables declaration
168 }
```

7.9. DBConnection.java

```

5  package JFrame;
6
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9
10 /**
11  * @author chrisjordan
12 */
13 public class DBConnection {
14     static Connection con = null;
15
16     public static Connection getConnection(){
17         try{
18             Class.forName("com.mysql.jdbc.Driver");
19             con = DriverManager.getConnection("jdbc:mysql://localhost:3306/test1","root","christ123");
20         }catch(Exception e){
21             e.printStackTrace();
22         }
23         return con;
24     }
25 }
26

```

7.10. SessionManager.java

```

1  package JFrame;
2
3  /**
4   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
5   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
6   */
7
8  /**
9   * @author chrisjordan
10 */
11 public class SessionManager {
12     private static User currentUser;
13
14     public static void setCurrentUser(User user) {
15         currentUser = user;
16     }
17
18     public static User getCurrentUser() {
19         return currentUser;
20     }
21
22     public static boolean isLoggedIn() {
23         return currentUser != null;
24     }
25
26     public static void logout() {
27         currentUser = null;
28     }
29 }
30

```

7.11. Users.java

```

1  package JFrame;
2
3  /**
4   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
5   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
6   */
7
8  /**
9   * @author chrisjordan
10 */
11 public class User {
12     private int id;
13     private String username;
14
15     // Constructors
16     public User() {
17         // Default constructor
18     }
19     public User(int id, String username) {
20         this.id = id;
21         this.username = username;
22     }
23     public int getId() {
24         return id;
25     }
26     public void setId(int id) {
27         this.id = id;
28     }
29     public String getUsername() {
30         return username;
31     }
32     public void setUsername(String username) {
33         this.username = username;
34     }
35 }
36

```

CHAPTER 8

RESULTS

Sign Up
Create Your Account Here!

Username: _____
Password: _____
E-Mail: _____
Contact: _____

SIGNUP | LOGIN

Login
Welcome back G!

Username: _____
Password: _____

LOGIN | SIGNUP

Fig: 8.1. Sign Up Page

SMART MACRO TRACKER

WELCOME G!

- HOME PAGE
- TRACK
- ANALYZE
- BMI CALCULATOR
- DIET PLAN
- WATER TRACKER
- LOGOUT

BMI
55.36

Water Consumption Log

S_no	Consumed
1	4.2
2	5.0
3	2.0
4	2.0

MACRO OVERVIEW

S_no	Total_Calories	Total_Protein
1	1750.0	279.0
2	3800.0	546.0

Fig: 8.2. Login Page

Fig: 8.3. Home Page

Smart Macro Tracker

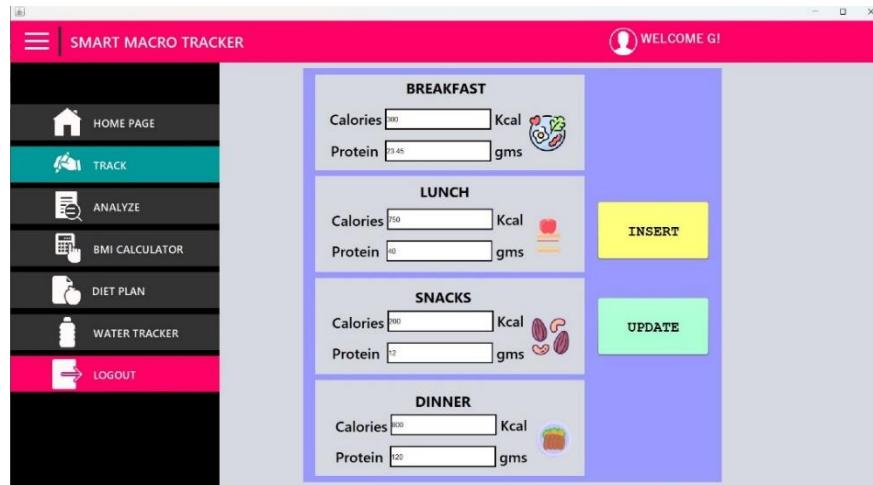


Fig: 8.4. Track Page

The screenshot shows the 'ANALYZE' page. At the top, it says 'ENTER YOUR DAILY CALORIES LIMIT' followed by a text input field containing '2200 kcal' and a 'COMPUTE' button. Below this is a table with four columns: BREAKFAST, LUNCH, SNACKS, and DINNER. The table has 4 rows of data. The last row is highlighted in yellow.

m_id	breakfast_calories	breakfast_protein	lunch_calories	lunch_protein	snack_calories	snack_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

The screenshot shows the 'BMI CALCULATOR' page. On the left is a vertical navigation bar with icons and labels: HOME PAGE, TRACK, ANALYZE, BMI CALCULATOR, DIET PLAN, WATER TRACKER, and LOGOUT. The main area has fields for 'Enter your Height (in Meters)' (1.75) and 'Enter your Weight (in Kgs)' (60), with a 'COMPUTE' button. Below these is a message: 'Your BMI is: 19.591837 Remarks: You are Normal'. There is a 'SAVE' button. At the bottom are four icons representing body types: UNDERWEIGHT (BMI < 18.5), NORMAL (BMI 18.5 - 24.9), OVERWEIGHT (BMI 25 - 29.9), and OBESSE (BMI > 30).

Fig: 8.6. BMI Calculator

Smart Macro Tracker



Fig: 8.7. Diet Plan Page

CALORIC DEFICIT

1. Intermittent Fasting:

- Start with a 12:12 fasting window and gradually increase it as your body adapts.
- Consider progressing to 14:10 or 16:8, allowing your body to enter a deeper fasting state.

2. Fasted Cardio:

- Engage in low-intensity cardio like jogging or brisk walking on an empty stomach to tap into stored fat for energy.
- Aim for 30-60 minutes, depending on your fitness level.

3. Calorie Deficit:

- BEGIN WITH A MODEST CALORIC DEFICIT OF 500 CALORIES PER DAY.
- GRADUALLY INCREASE THE DEFICIT BY 100-200 CALORIES EVERY WEEK UNTIL REACHING A SUSTAINABLE LEVEL.
- ENSURE A BALANCE BETWEEN REDUCING CALORIES AND MAINTAINING NUTRIENT INTAKE.

4. Protein and Fiber-Rich Diet:

- Prioritize lean protein sources like chicken, fish, tofu, legumes, and incorporate high-fiber foods such as fruits, vegetables, and whole grains.
- Protein helps preserve muscle mass during weight loss, while fiber aids in satiety.

5. Hydration:

- Consume plenty of water throughout the day, preferably warm water, to support digestion and metabolic processes.
- Adequate hydration can also help control hunger and prevent overeating.

6. Physical Activity:

- Aim for at least 10,000 steps daily through walking or any form of physical activity.
- Include bodyweight exercises like push-ups, jumping jacks, ab workouts, sit-ups, and squats with稍重 weights to enhance muscle tone and calorie burning.

7. Strength Training:

- Incorporate bodyweight training or resistance exercises to build and tone muscle.
- This can help increase your metabolism and contribute to a more sculpted physique.

8. Sleep:

- Prioritize 7-8 hours of quality sleep each night to support recovery and hormonal balance.
- Lack of sleep can interfere with weight loss efforts and increase stress hormones.

9. Pre-Workout:

- Before evening workouts, consider having black coffee without sugar.
- Caffeine can boost metabolism and improve physical performance.

10. Monitoring and Adjustments:

- Regularly track your progress through measurements, weight, and how your clothes fit.
- Be flexible and adjust the plan based on your body's response and individual needs.

Fig: 8.7.1. Caloric Deficit Page

BODY RECOMPOSITION

1. Fasted Cardio:

- Incorporate fasted cardio sessions, such as jogging or walking, to enhance fat metabolism. This can be done in the morning before breakfast to maximize the use of stored energy.

2. Calorie Surplus:

- Gradually increase caloric intake to be in a surplus, adding 500-700 calories every week.
- Focus on a controlled surplus to support muscle growth without excessive fat gain.

3. Macronutrient Distribution:

- Prioritize protein-rich foods to support muscle protein synthesis. Aim for at least 1.6 to 2.2 grams of protein per kilogram of body weight.
- Include complex carbohydrates to provide sustained energy for workouts & aid in recovery.

4. Calorie-Dense Foods:

- Choose nutrient-dense, high-calorie foods to meet your increased energy needs.
- Include sources of healthy fats such as avocados, nuts, seeds, and olive oil.

5. Weight Training:

- Prioritize weight training exercises 3-4 times a week to stimulate muscle growth. Include compound movements like squats, deadlifts, bench press, and overhead press.
- Focus on progressive overload, gradually increasing weight & intensity to your muscles.

6. Sleep:

- Prioritize 7-8 hours of quality sleep per night. Adequate sleep is crucial for muscle recovery and overall well-being.

7. Pre-Workout Nutrition:

- Consume a pre-workout meal containing carbohydrates and a moderate amount of protein about 1-2 hours before your training session.
- Examples include a banana, whole grain toast with peanut butter, or a protein smoothie with fruit.

8. Post-Workout Nutrition:

- Have a post-workout meal containing protein and carbohydrates to replenish glycogen stores and support muscle recovery.
- Options include a protein shake with a source of fast-digesting carbohydrates or a balanced meal with lean protein and complex carbs.

9. Hydration:

- Stay well-hydrated throughout the day. Proper hydration is essential for overall health.

10. Monitoring and Adjustments:

- Regularly assess your progress through measurements, strength gains, and overall well-

Fig: 8.7.2. Body Re-composition Page

Smart Macro Tracker



Fig: 8.7.3. Caloric Surplus Page

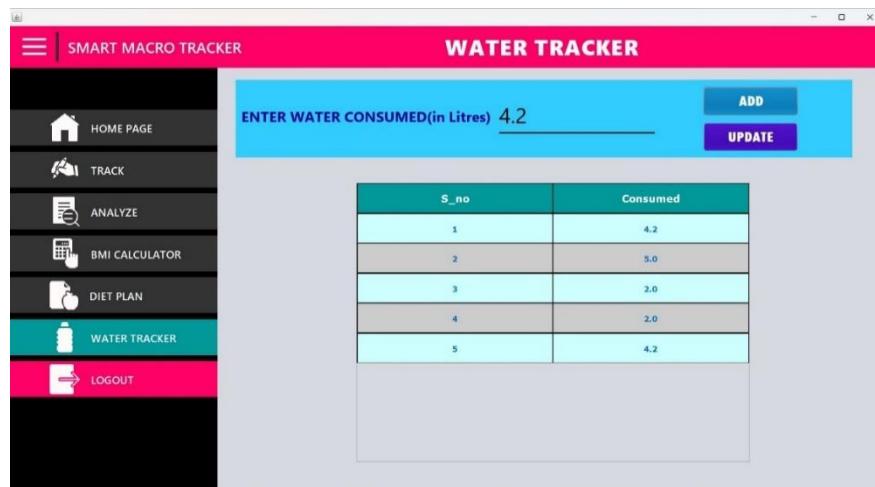


Fig: 8.8. Water Tracker Page

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 physique 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