

CHAPTER 1

INTRODUCTION

1.1. PROBLEM DEFINITION

The management of pension funds is currently beset by inefficiencies, a lack of transparency, and susceptibility to risks like fraud and noncompliance with regulations. The aforementioned obstacles, when combined with changing demographics and uncertain economic conditions, highlight the pressing necessity for an updated pension fund administration system. A system of this kind should simplify administrative procedures, improve openness, maximize investment plans, reduce risks, guarantee legal compliance, and strengthen security protocols. The goal is to develop a robust and effective pension fund management system that protects retirees' financial security and builds stakeholder confidence in the integrity of the pension system by tackling these issues head-on.

1.2. OBJECTIVES

The aim is to create a contemporary pension fund management system that tackles inadequacies, amplifies transparency, maximizes investment approaches, minimizes hazards, guarantees adherence to regulations, and reinforces security protocols.

Through the accomplishment of these goals, the system hopes to protect retirees' financial security, optimize administrative procedures, and build stakeholder confidence in the integrity of the pension system, all of which will contribute to the development of a strong and effective framework for pension fund management.

1.3. METHODOLOGY TO BE FOLLOWED

Modern techniques and best practices are integrated into a comprehensive approach that was used in the development of the updated pension fund management system. This entails carrying out in-depth research to pinpoint important issues and specifications, working with stakeholders and industry experts to acquire insights, and utilizing cutting-

edge technologies like blockchain and artificial intelligence to improve investment strategies, expedite administrative procedures, and boost transparency. To reduce risks and guarantee compliance with regulations, strong risk management frameworks and compliance procedures will also be put in place. The system will be iteratively refined through ongoing monitoring and assessment to guarantee that retirees' financial security and the public's confidence in the pension system are protected.

1.4. EXPECTED OUTCOMES

There are numerous and extensive anticipated benefits from putting in place a modernized pension fund management system. First, by automating repetitive operations, cutting down on paperwork, and raising the general effectiveness of pension fund management operations, the system is expected to simplify administrative procedures. For pension fund administrators, this efficiency gain will result in cost savings and resource optimization.

1. Enhanced Efficiency: Streamlined administrative processes leading to cost savings and resource optimization.
2. Improved Transparency: Real-time access to fund performance data, enhancing trust and accountability.
3. Optimized Investment Strategies: Utilization of advanced technologies for predictive analysis, resulting in improved returns.
4. Effective Risk Management: Identification, assessment, and mitigation of various risks, safeguarding fund sustainability.
5. Regulatory Compliance: Automated reporting and monitoring to ensure adherence to legal requirements, mitigating legal risks.
6. Strengthened Security: Implementation of robust security measures to prevent fraud and cybersecurity threats, safeguarding fund assets and data.

These outcomes collectively contribute to the overall goal of preserving the financial well-being of retirees and instilling confidence in the integrity of the pension system.

1.5. HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE:

- An Intel Pentium processor to guarantee seamless operation.
- 8 GB of RAM at minimum; however, for better multitasking, more RAM is advised.
- A minimum 128GB storage space to hold the data and related files for the system.

SOFTWARE:

- Java Spring Boot is the backend framework used to develop the backend application.
- Front-end architectures:
- The JavaScript library React.js, which is used to create the user interface.
- Tailwind CSS, which styles the elements of the front end.
- Material-UI for extra design elements and UI components.

PostgreSQL is the principal Relational Database Management System (RDBMS) utilized for data storage.

- Tool Development:
- An Integrated Development Environment (IDE) such as Eclipse or IntelliJ IDEA for Java development.
- A code editor (like Visual Studio Code) for developing JavaScript.
- Package managers (like Maven and npm) for managing dependencies.

CHAPTER 2

FUNDAMENTALS OF JAVA

2.1. INTRODUCTION

Java is a popular and flexible object-oriented programming language that can be used in a wide range of fields, including enterprise systems, scientific computing, mobile applications, and web development. One of Java's built-in benefits is its "Write Once, Run Anywhere" feature, which enables programmers to write code that runs smoothly on any platform that has a Java Virtual Machine (JVM). Because of this unique quality, Java is positioned as the go-to option for cross-platform applications, guaranteeing code portability across various operating systems.

A vital part of giving developers access to a wealth of reusable parts and APIs is the Java Development Kit (JDK), which houses a sizable class library. This gives developers the ability to quickly and effectively create complex applications by utilizing pre-built features. Java remains a fundamental language for a wide range of software development projects because of its strong ecosystem and platform independence.

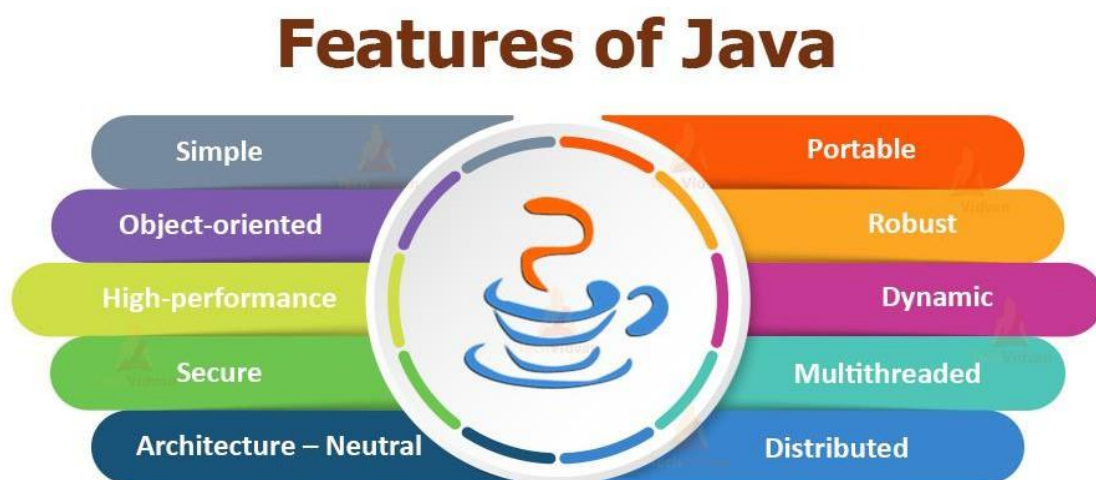


Fig 1.1 Different features of java

2.2. ADVANTAGES IN JAVA

1. Platform Independence: Java programs are extremely portable and versatile because they can run on any platform that supports the Java Virtual Machine (JVM).
2. Object-Oriented: Java is an object-oriented programming language that makes it possible to write modular and reusable code, which makes applications easier to maintain and scale.
3. Robust Standard Library: Java comes with a strong standard library (Java API) that includes pre-built classes and functions for common programming tasks, saving developers from having to start from scratch when writing code.
4. Automatic Memory Management: Memory leaks are less likely and memory management is made easier for developers by Java's automatic garbage collection feature, which controls memory allocation and deallocation.
5. Support for Multiple Threading: Java has the ability to support multiple threads, which is essential for creating applications that are responsive and scalable, particularly in server-side and multi-user settings.
6. Security: Java comes with built-in security features like sandboxing and bytecode verification that help guard against malicious code and guarantee the integrity of programs that run on the JVM.
7. Community and Ecosystem: Java offers a large developer community as well as a robust ecosystem of libraries, frameworks, and tools, giving programmers all they need to create creative and high-caliber applications.



Fig 1.2 Advantages of java

2.3. DATA TYPES

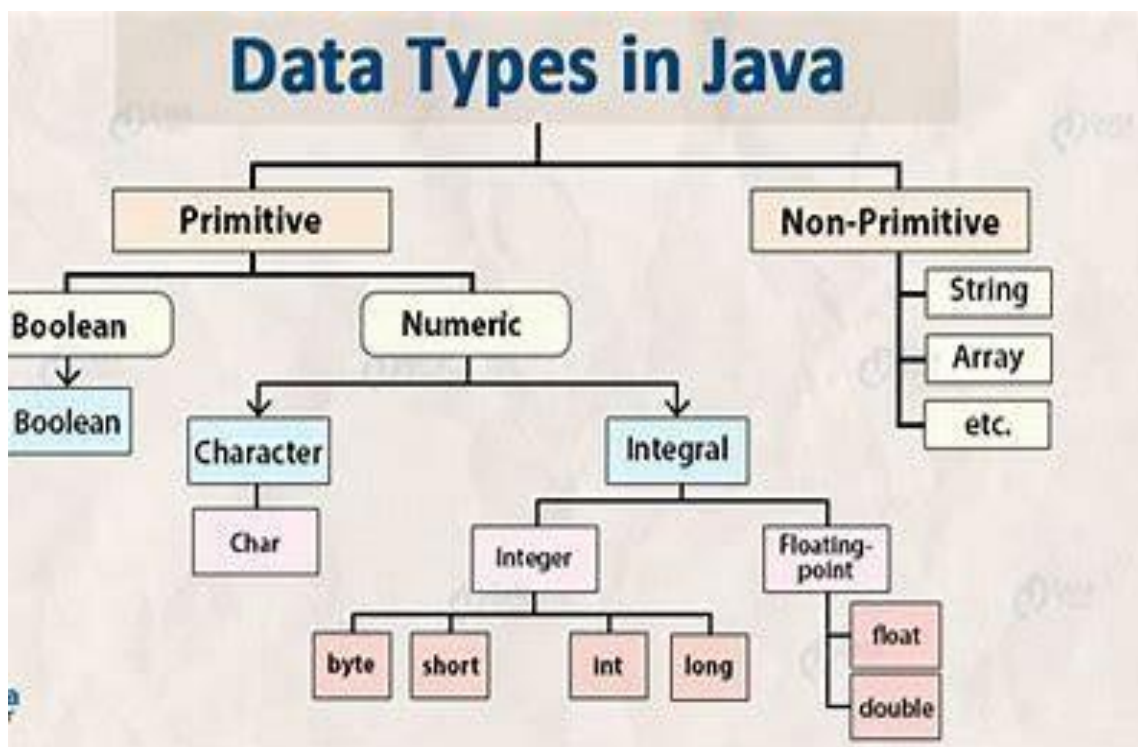


Fig 1.3 Data types in java

2.4. CONTROL FLOW

In Java, control flow describes the order in which statements are carried out within a program. Java provides a number of control flow mechanisms to regulate the execution flow, such as:

1. **Conditional Statements:** To execute distinct code blocks depending on predefined conditions, Java offers conditional statements like if, else if, and else. Developers can regulate the execution flow by using these statements to evaluate boolean expressions.
2. **Looping Statements:** To repeatedly run a block of code, Java allows looping statements like for, while, and do-while. With the help of these statements, developers can execute commands until a predetermined condition is satisfied or iterate over arrays and collections.
3. **Java provides branching statements, such as break, continue, and return, which can be used to change the direction in which loops or switch statements execute. These statements let programmers return values from methods, skip iterations, and prematurely end loops.**
4. **Switch Statement:** Depending on the value of a variable or expression, Java's switch statement enables programmers to run distinct code blocks. In situations where several conditional branches need to be evaluated, it offers an alternative to nested if-else statements.
5. **Exception Handling:** Runtime errors and other exceptional conditions that may arise during program execution can be gracefully handled by developers thanks to Java's exception handling mechanism. It has the keywords try, catch, finally, and throw to handle exceptions and keep the program stable.

All things considered, Java's control flow mechanisms give programmers strong tools to control how their programs execute, enabling the development of reliable, adaptable, and maintainable software solutions.

2.5. METHODS

1. **Static Methods:** Typically used for utility functions, static methods in Java belong to the class rather than any instance of the class, enabling direct calling without the need to create an object.
2. **Void Methods:** In Java, void methods are used to perform actions or operations without yielding a result. They do not return any value after execution.

3. Passing Parameters: Java methods with parameters enable the passing of values into the method for processing, which improves flexibility and reusability by allowing the behavior of the method to be customized.
4. Return Statement: To enable data retrieval or method output, the return statement in Java methods is used to end the method and return a value or result to the calling code.
5. Method overriding: It feature encourages polymorphic behavior and makes code customization easier by allowing a subclass to offer a particular implementation of a method that is already defined in its superclass.
6. Method overloading: It allows for the coexistence of multiple methods with the same name but different parameters within a class. This feature, known as method overloading, improves code readability and flexibility by offering multiple ways to invoke a single method.
7. Access modifiers: which include public, private, protected, and default (package-private), regulate the accessibility of variables, classes, and methods. They also maintain code visibility and security by guaranteeing encapsulation.
8. Instance methods: It can operate on object-specific data because they are attached to specific objects within a class and have access to instance variables and other instance methods.

2.6. OBJECT ORIENTED CONCEPTS

The paradigm of object-oriented programming (OOP) is founded on the idea of "objects," which can hold code in the form of procedures (methods or functions) and data in the form of fields (attributes or properties). Modularity, encapsulation, inheritance, and polymorphism are emphasized by OOP as the main tenets for organizing and creating software systems.

1. Objects and Classes: Classes serve as templates for constructing objects, specifying their characteristics and methods. Classes are represented by objects, which are particular instances with distinct data and behavior.
2. Encapsulation: Bundling data (attributes) and methods that work with that data into a single unit (class) is known as encapsulation. By preventing external access to an object's internal state and guaranteeing that data can only be accessed and altered via approved procedures, it improves data security and integrity.
3. Inheritance: A class (subclass or child class) can inherit characteristics and behaviors from another class (superclass or parent class) through the mechanism of inheritance. It

encourages the reuse of code by enabling subclasses to inherit from and specialize on the functionality of their parent classes.

4. Polymorphism: Through method overloading and overriding, polymorphism enables objects of different classes to be treated as objects of a common superclass. It makes code design more adaptable and extensible by enabling runtime binding based on the actual type of objects and dynamic method invocation.

5. Abstraction: Abstraction is the process of modeling the pertinent parts of complex systems and hiding the unneeded details in order to simplify them. Abstract classes and interfaces, which define a common interface for related classes while concealing implementation details, are how OOP achieves abstraction.

6. Modularity : This promotes code reusability, maintainability, and scalability by emphasizing the division of code into more manageable, smaller units (classes and modules). It makes code organization easier, lowers complexity, and fosters better developer collaboration.

2.7. EXCEPTION HANDLING

1. TRY-CATCH BLOCKS: Try-catch blocks are used in Java to elegantly handle exceptions. Try blocks include code that may throw an exception; if this happens, the catch block handles the exception and implements any necessary error handling or recovery procedures.

2. MULTIPLE CATCH BLOCKS: Java allows distinct kinds of exceptions to be handled independently by allowing numerous catch blocks to be linked to a single try block. Every catch block identifies a specific exception type that it is capable of handling, giving error management flexibility and enabling customized responses to various error types.

3. FINALLY BLOCK: Try-catch blocks are used in Java to elegantly handle exceptions. Try blocks include code that may throw an exception; if this happens, the catch block handles the exception and implements any necessary error handling or recovery procedures.

4. THROW: Java allows distinct kinds of exceptions to be handled independently by allowing numerous catch blocks to be linked to a single try block. Every catch block identifies a specific exception type that it is capable of handling, giving error management flexibility and enabling customized responses to various error types.

5. **THROWS:** In Java, the throws keyword is used in method declarations to specify that the method may throw specific kinds of errors while it's running. It provides a mechanism to transfer exception handling responsibilities to the caller by identifying the exceptions that are not handled within the method itself but are instead propagated to the calling code for processing.

2.8. FILE HANDLING

1. **FILE CLASS:** The File class in Java is used to represent directory and file pathnames. It offers ways to add, remove, rename, and find out details about files and folders on the file system.

2. **FILE INPUT/OUTPUT STREAMS:** Writing to files is done via the FileOutputStream and FileInputStream Java classes, respectively. They enable the transport of data as a series of bytes between the application and external files.

3. **BYTE VS CHARACTER STREAMS:** Bit streams, which are appropriate for binary data, manage I/O operations at the byte level and are represented by the InputStream and OutputStream classes. Character streams—represented by the Reader and Writer classes—manage character-level input/output (I/O) operations that are appropriate for text data and take care of character encoding and decoding automatically.

4. **READING AND WRITING TEXT FILES:** The FileReader and FileWriter classes in Java allow you to read and write text files at the character level. By automatically translating characters to bytes using the platform's default character encoding, these classes make managing text data easier.

5. **FILE AND DIRECTORY OPERATIONS:** The methods for creating, deleting, renaming, and listing files in a directory are among the operations that may be done on files and directories using Java's File class. Developers can use these actions to programmatically handle files and directories within the file system.

6. **RandomAccessFile:** The Java class RandomAccessFile enables reading and writing to a file at any point, instead of in sequential order. Because of this, the file can have effective random access to its contents, which makes it appropriate for applications that need frequent random access operations.

7. **SERIAL VERSUS DESERIALIZATION:** An object is serialized when it is made into a byte stream, usually for transmission or storage. The process of reconstructing the object from the byte stream in reverse is called deserialization. For network connection or permanent storage, Java objects can be deserialized using ObjectInputStream and serialized using ObjectOutputStream.

8. HANDLING EXCEPTIONS IN FILE I/O: When executing file I/O operations in Java, a number of causes, including file not found, permission problems, or I/O faults, may result in exceptions such `FileNotFoundException`, `IOException`, and `SecurityException`. In order to provide robustness and dependability in file I/O operations, these exceptions are gracefully handled by means of exception handling methods like try-catch blocks.

2.9. PACKAGES AND IMPORT

1. PACKAGES DECLARATION: The "package" keyword and the package name are used to declare packages in Java at the start of a source file.
2. PACKAGE STRUCTURE: Java packages offer a hierarchical framework for class and interface management, preventing naming conflicts and preserving code modularity.
3. IMPORT STATEMENTS: Java import statements facilitate code reuse and improve readability by allowing classes from other packages to be accessed in the current source file.
4. NAMING CONFLICTS: When classes or interfaces with the same name exist in separate packages, it might lead to naming conflicts in Java. To address ambiguity, explicit qualification or aliasing are required.
5. DEFAULT PACKAGE: If no package declaration is given, classes are placed in the package without a stated name in Java. However, because of potential naming conflicts and a lack of encapsulation, using this package is discouraged.
6. STATIC IMPORT: Java's static import feature enables access without class qualification to static members (fields and methods) of one class to be imported directly into another, hence decreasing verbosity in code.
7. CLASSPATH AND PACKAGES: The Java classpath is an environment variable that tells the Java compiler and runtime where to look for classes and packages so that applications can use them.
8. LIBRARY PACKAGES: Pre-written classes and interfaces grouped together to provide specific functionalities are called library packages in Java. For example, `java.util`

2.10 INTERFACES

1. INTERFACE STATEMENT: When establishing abstract methods that implementing classes must supply, an interface in Java is specified using the "interface" keyword, which is followed by the interface name and its body.

2. **DEFAULT METHODS:** Java interfaces with default methods enable the inclusion of new methods without affecting backward compatibility. These default implementations can be modified by classes that implement them if necessary.
3. **STATIC METHODS:** Java 8 brought about the introduction of static methods in interfaces, which allowed the definition of utility methods connected with the interface itself and did not require an instance of the implementing class to be used.
4. **Multiple inheritance of interfaces** is supported by Java, which spares classes the complications of multiple class inheritance by enabling a class to implement several interfaces and inherit abstract methods from each interface.
5. **IMPLEMENTING INTERFACES:** In Java, an interface is implemented when a class uses the "implements" keyword with the interface name or interfaces' names after it. This gives all of the abstract methods defined in the interface(s) concrete implementations.
6. **EXTENDING INTERFACES:** Java interfaces have the ability to extend other interfaces by using the "extends" keyword. This permits the sub-interface to add new methods or constants and inherit abstract methods from the parent interface or interfaces.

2.11. CONCURRENCY

1. **THREADS AND MULTITHREADING:** Java threads are small processes that facilitate concurrent execution, allowing several operations to be carried out at once within a single program, improving responsiveness and performance.
2. **THREAD LIFECYCLE:** A thread travels through several states from creation to termination, which are represented by the Java thread lifecycle: new, runnable, blocked, waiting, timed waiting, and terminated.
3. **RUNNABLE INTERFACE:** In Java, an action or task that can be carried out by a thread is represented by the Runnable interface, which makes it possible to detach the task's implementation from the thread.
4. **SYNCHRONIZATION:** Java's synchronization system regulates how many threads can access shared resources at once, preventing inconsistent and corrupt data. Only one thread is able to access the resource at a time.
5. **LOCKS AND MUTEX :** Java offers low-level synchronization methods called locks and mutexes, which give threads exclusive access to shared resources while blocking concurrent access by other threads until the lock is released.

6. DEADLOCKS: In multithreaded programs, deadlocks happen when two or more threads are stuck together waiting for one another to release a resource, leaving no thread able to move forward.

7. THREAD SAFETY: In Java, a class's or method's ability to be utilized by many threads concurrently without resulting in data corruption or unexpected behavior is known as thread safety.

8. EXECUTORY FRAMEWORK: Java's executor framework offers a high-level abstraction for controlling thread execution, allowing for the division of work submission and execution. It also facilitates asynchronous execution, thread pooling, and scheduling.

CHAPTER 3

JAVA COLLECTIONS GUIDE

3.1. OVERVIEW OF JAVA COLLECTIONS

For the purpose of storing, managing, and processing collections of objects in Java programs, the Java Collections Framework offers an extensive collection of classes and interfaces. It provides a large selection of data formats and algorithms, making collection management easy and effective.

The framework comes with fundamental interfaces like Map, Set, and List that provide several approaches to data organization and access. ArrayList, LinkedList, HashSet, and TreeMap are a few examples of implementations of these interfaces that meet different requirements and performance requirements.

Generics are supported by the Java Collections Framework, which makes type-safe collections possible, improves code readability, and eliminates the need for explicit type casting.

It encourages code reuse and boosts efficiency by offering algorithms for sorting, finding, and modifying collections through utility methods in the Collections class.

The focus of the framework is interoperability, which makes it possible to integrate collections with other Java APIs like serialization, concurrency tools, and stream API with ease.

All things considered, the Java Collections Framework is a fundamental component of Java programming, providing extensive capability for managing object collections in an effective and efficient manner.

3.2. IMPORTANTS IN JAVA DEVELOPMENT

Robust, effective, and maintainable software solutions are the result of numerous key factors in Java development. The following are some essential components:

1. Object-Oriented Programming (OOP): Encapsulation, inheritance, and polymorphism—three OOP tenets that support modular, scalable, and reusable code—are the foundation of Java.

2. JDK (Java Development Kit): It is necessary to be familiar with JDK, including the libraries, tools, and APIs for creating, troubleshooting, and implementing Java applications.
3. Integrated Development Environment (IDE): Using IDEs such as IntelliJ IDEA, Eclipse, or NetBeans effectively can help you be more productive by streamlining the coding, debugging, and project management processes.
4. Java SE (Java Standard Edition): Building foundational knowledge and creating simple Java applications require a firm grasp of fundamental Java ideas, syntax, and functionality.
5. Java Enterprise Edition (Java EE): To create scalable, enterprise-grade applications, one must be familiar with Java EE technologies including Servlets, JSP, JDBC, JPA, and EJBs.
6. Frameworks and Libraries: Mastery of well-known Java frameworks and libraries, such as Hibernate, Spring, Apache Maven, and Apache Tomcat, speeds up development, makes standard jobs easier, and encourages best practices.
7. Concurrency: Building successful, responsive apps that make good use of contemporary hardware requires a solid understanding of Java's multithreading, synchronization, and concurrent programming features.
8. Unit Testing: Developers may ensure software quality and dependability by automating testing processes and writing robust, dependable code by mastering unit testing frameworks like JUnit and TestNG.
9. Design Patterns: Knowledge of design patterns like MVC, Factory, Singleton, and Observer encourages scalable, modular, and maintainable programming, which makes code reuse and improving readable content.
10. Version Control Systems (VCS): Working with VCS tools such as Git allows you to track changes, version code, and collaborate with others. It also ensures code integrity.
11. Continuous Integration and Continuous Deployment (CI/CD): The software development lifecycle is streamlined, build and deployment processes are automated, and code quality is ensured by understanding CI/CD pipelines and tools like Travis CI.
12. Security: Building secure applications and defending against potential threats and assaults is made easier by being aware of common vulnerabilities, secure coding techniques, and best practices for Java security.
13. Documentation and Commenting: Using tools like Javadoc to write simple, understandable code comments and documentation improves code readability, maintainability, and fosters developer cooperation.
14. Optimization of Performance Knowledge of JVM tuning, optimization methods, and performance profiling tools facilitates the identification and resolution of performance

bottlenecks, guaranteeing peak performance from the application.

Developers can produce software solutions that are dependable, scalable, and suit business needs by grasping four key facets of Java development.

3.3. BENEFITS AND USECASES

BENEFITS:

1. **PLATFORM INDEPENDENCE (WORA):** Java's "Write Once, Run Anywhere" (WORA) concept enables the compilation of Java programs into bytecode, which is compatible with any platform by allowing them to execute on a Java Virtual Machine (JVM).
2. **OBJECT ORIENTED:** By utilizing the concepts of encapsulation, inheritance, and polymorphism, Java's object-oriented programming (OOP) paradigm encourages modular, reusable, and maintainable code.
3. **RICH STANDARD LIBRARY** (set of APIs for multiple functionality): Java has a rich standard library that includes extensive APIs for a range of behaviors, making it easier to design applications that are both scalable and efficient. These functionalities include networking, collections, concurrency, and I/O operations.
4. **MULTITHREADING SUPPORT:** Java comes with built-in support for multithreading, which enables programmers to design concurrent apps that can carry out several activities at once, improving responsiveness and performance.
5. **DYNAMIC AND EXTENSIBLE:** Java's extensibility and dynamic nature allow programmers to add new features and capabilities to the language by utilizing frameworks, libraries, and outside tools, which promotes creativity and adaptability.
6. **COMMUNITY SUPPORT:** Java has a sizable and vibrant developer, enthusiast, and contributor community that offers forums, resources, and assistance, encouraging cooperation, education, and information sharing.
7. **Robustness:** Java's robustness originates from its powerful memory management, compile-time error checking, and exception handling features, which enable programmers to create dependable and stable software systems.

USECASES:

1. WEB DEVELOPMENT
2. ENTERPRISE APPLICATION (SCALABLE AND ROBUST)
3. MOBILE APPLICATIONS (JAVA OR KOTLIN)
4. CLOUD COMPUTING
5. DESKTOP APPLICATIONS
6. EMBEDDED SYSTEMS
7. GAME DEVELOPMENT (WITH Lib GDX AND ENGINES LIKE UNITY)

3.4. CORE COLLECTIONS INTERFACES

1. **COLLECTION INTERFACE:** The Java Collection interface offers the fundamental operations for manipulating collections, including insertion, deletion, and iteration over members. It also represents a set of objects known as elements.
2. **LIST INTERFACES:** Java's list interfaces, such List and ArrayList, extend Collection and give ways to access elements by index as well as an ordered collection of elements where duplicates are permitted.
3. **SET INTERFACES:** Set interfaces in Java, such as Set and HashSet, extend Collection and provide methods for adding and removing members while maintaining uniqueness. They represent a collection of unique components where duplicates are prohibited.
4. **QUEUE INTERFACES:** In Java, queue interfaces like LinkedList and Queue describe a collection that retains elements before processing and adheres to the First-In-First-Out principle.
5. **MAP INTERFACES:** Map interfaces, such as HashMap and Map in Java, enable for the efficient retrieval, insertion, and deletion of components based on keys. They represent a collection of key-value pairs where each key is unique.
6. **DEQUE INTERFACES:** Java's Deque and ArrayDeque interfaces extend Queue and offer a double-ended queue with methods for both stack- and queue-like behavior. They also handle the insertion and removal of elements at both ends.

7. **SORTEDSET INTERFACE:** The Java SortedSet interface extends Set and provides methods for range operations and position-based element access. It represents a set of elements sorted in ascending order based on either a provided comparator or their natural ordering. their place.

8. **SORTEDMAP INTERFACE:** The Java SortedMap interface is an extension of Map that depicts a map with keys kept in ascending order. It provides access to key-value mappings based on position and range operations.

3.5. COMMON COLLECTION IMPLEMENTATIONS

1. **Priority Queue:** A Java implementation of the Queue interface, Priority Queue places elements in order either by a predetermined comparator or by their natural ordering, guaranteeing that the element with the highest priority is always at the front of the queue.

2. **LINKED HASHSET:** LinkedHashSet is a Java implementation of the Set interface that uses a linked list and a hash table to ensure uniqueness while maintaining the insertion order of members.

3. **HASH TABLE:** HashMap is a Java implementation of the Map interface that stores key-value pairs in a hash table data structure for quick retrieval and insertion operations with an average complexity of constant time.

4. **VECTOR:** A synchronized dynamic array implementation of the List interface in Java, Vector may result in performance overhead but guarantees thread safety during concurrent access.

5. **STACK:** A Last-In-First-Out (LIFO) data structure that enables elements to be placed onto and removed from the stack in a sequential order, Stack is a Java subclass of Vector.

6. **TREE MAP:** TreeMap is a Java implementation of the SortedMap interface that offers effective range operations and key-based navigation by storing key-value pairs in a sorted order based on the keys' inherent ordering or a supplied comparator.

7. **ARRAY DEQUE:** ArrayDeque is a Java implementation of the Deque interface that provides effective insertion and removal operations at both ends of the deque by storing elements in a resizable array.

8. **LINKED LIST:** The Java implementation of the List interface, LinkedList, stores elements in a doubly linked list data structure, facilitating quick insertions and deletions at any point in the list.

3.6. ITERATORS AND COLLECTIONS API

ITERATORS: Java iterators offer ways to iterate a collection such that each element can be accessed in turn. The Java collection framework's "hasNext()" and "next()" iterator interfaces are used to determine whether there are any more elements in the iteration and to retrieve the subsequent element.

COLLECTIONS API:

1. **COLLECTION:** The Java Collection interface offers a uniform interface for working with collections, including methods for adding, removing, and iterating over items. Collections are groups of objects that are referred to as elements.
2. **LIST:** The Java List interface extends Collection and provides methods for adding, removing, and accessing elements by index. It represents an ordered collection of elements where duplicates are permitted and elements can be retrieved by their index position.
3. **SET:** The Java Set interface is an extension of Collection that denotes a set of distinct elements in which duplicates are prohibited. It offers methods for both addition and deletion of members while maintaining uniqueness.
4. **QUEUE:** The Java Queue interface is an extension of Collection, which provides methods for inserting, removing, and inspecting elements. It represents a collection that retains elements before processing and is arranged in the First-In-First-Out (FIFO) order.
5. **MAP:** The Java Map interface defines a set of key-value pairs in which every key is distinct. This enables effective element retrieval, insertion, and deletion based on keys and offers key-based navigation and manipulation techniques.

3.7. CUSTOM COLLECTIONS AND GENERICS

CUSTOM COLLECTIONS:

1. **CUSTOM LIST IMPLEMENTATION:** To implement a custom list in Java, one must create a class that resembles the behavior of the List interface, assign methods for adding,

removing, and accessing elements, and manage an underlying array or linked structure that stores elements.

2. CUSTOM SET APPLICATION: To implement a custom set application in Java, you need to create a class that keeps track of a set of unique components. To guarantee uniqueness, you can frequently use a balanced tree structure or a hash table. You also need to provide methods for querying, adding, and removing entries from the collection.

3. CUSTOM MAP IMPLEMENTATION: To create a custom map implementation in Java, a class representing a collection of key-value pairs must be created. To offer effective key-based retrieval, insertion, and deletion, a balanced tree structure or hash table are usually used.

4. CUSTOM QUEUE IMPLEMENTATION: Developing a Java class that oversees a group of elements in the First-In-First-Out (FIFO) order entails building a custom queue. To make insertion and removal of elements at both ends of the queue easier, an array or linked structure is frequently used.

GENERICs:

1. CLASSES (class Box<T>)

2. INTERFACES (interface List<T>)

3. METHODS (public <T> T getElement(T[] array,int index))

4. BOUNDED TYPE PARAMETERS (<T extends Number>)

5. WILDCARDS (List<?>)

6. GENERIC CONSTRUCTORS (public <U> MyClass(U input))

CHAPTER 4

FUNDAMENTALS OF DBMS

4.1. INTRODUCTION

A Course on Database Management Systems (DBMS) includes an overview of software systems that are made to effectively store, handle, and retrieve data. A key element of contemporary information systems, database management systems (DBMS) allow businesses to access and manage massive amounts of data in an organized way. It offers tools for data manipulation and querying and helps with concurrency management, security, and data integrity. Data modeling, schema design, query languages, and transaction management are important ideas that are meant to optimize data storage and retrieval procedures in order to satisfy the requirements of various users and applications.

4.2. CHARACTERISTICS OF A DBMS

1. **Data Independence:** Database management systems (DBMS) offer abstraction layers that divide a piece of data's logical representation from its physical storage, allowing modifications to one without impacting the other.
2. **Data Integrity:** To guarantee the correctness and consistency of data stored in the database, DBMS implements data integrity constraints, which prohibit the entry of erroneous or incomplete data.
3. **Concurrency Control:** A DBMS controls how many users or applications can access the database at once, protecting the consistency and integrity of the data while enabling effective concurrent transaction execution.
4. **Data Security:** To prevent unauthorized access and manipulation of sensitive data, database management systems (DBMS) employ security techniques such as user authentication, authorization, and encryption to regulate database access.
5. **Data Recovery:** Data availability and durability are ensured by DBMS's backup and recovery methods, which protect against data loss from hardware malfunctions, system failures, or human mistake.

6. Query Language Support: Structured Query Language (SQL) and other query languages are provided by DBMSs to enable users to efficiently obtain, update, and manipulate data.
7. Transaction Management: Database management systems (DBMS) guarantee that transactions have the ACID qualities (Atomicity, Consistency, Isolation, Durability), which assures that database operations are performed accurately and dependably even in the event of errors.
8. Scalability and Performance: Indexing, query optimization, and caching techniques help DBMS optimize performance while supporting scalability by handling growing data volumes and user loads.
9. Data Abstraction: A simplified abstraction layer conceals the complexity of data administration and storage, allowing users and applications to interact with the database through a high-level interface provided by DBMS.
10. Multi-user Support: A database management system (DBMS) offers tools to guarantee data consistency and integrity in a multi-user setting by enabling numerous users or applications to access and modify the database simultaneously.

4.3. DATA MODEL

The conceptual representation of data and its relationships within a database is referred to as a data model in the context of Database Management Systems (DBMS). It acts as a guide for creating the database's structure and specifying the methods for storing, organizing, and accessing data. Important traits of a data model consist of:

1. Entities and Attributes: Determining which concepts, things, or qualities need to be represented in the database, as well as the attributes that go along with them.
2. Relationships: Determining the one-to-one, one-to-many, or many-to-many relationships between entities in order to record associations and dependencies between them.
3. Defining constraints, such as entity integrity, referential integrity, and domain constraints, in order to ensure data consistency and integrity.
4. Normalization: Using normalization techniques to reduce data dependencies and redundancies, assuring effective data upkeep and storage.
5. Data Abstraction: By hiding implementation specifics behind a conceptual abstraction of data, users and applications can interact with databases without being aware of their internal workings.

6. Modeling Language: To graphically and officially define the data model, modeling languages like Entity-Relationship (ER) diagrams, Unified Modeling Language (UML), or relational schema are used.

7. Flexibility: Encouraging adaptability to the gradual evolution of data requirements over time, enabling necessary data model extensions and revisions.

4.4. THREE SCHEMA ARCHITECTURE

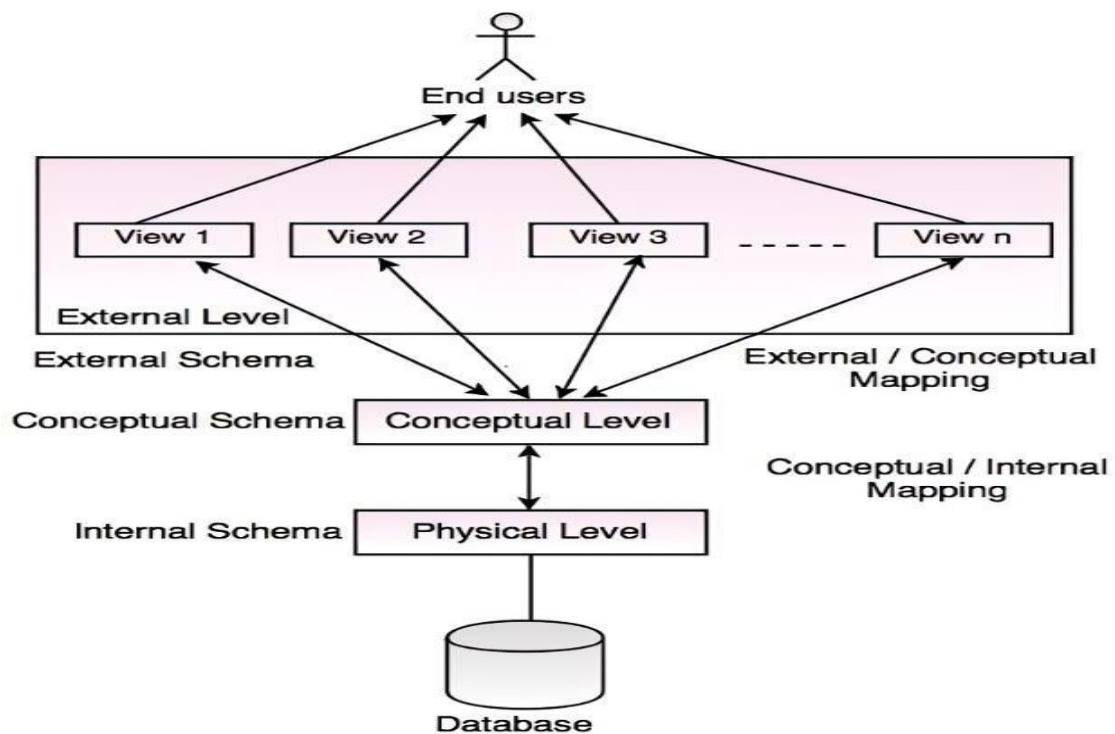


Fig. Three Level Architecture of DBMS

Fig 4.1 Three level Architecture of DBMS

4.5. DBMS COMPONENT MODULES

A database management system, or DBMS, is usually made up of a number of modules or components that cooperate to offer effective data administration, storage, and retrieval. Among these modules are:

1. Data management, retrieval, and storage on physical storage devices are the responsibilities of the storage manager. It manages activities including data organization, data access techniques, and disk space allocation and deallocation.

2. Query Processor: Prepares an execution plan for obtaining or modifying data from the database by processing user requests defined in a query language (like SQL). It consists of parts such as query executor and query optimizer.
3. Database transactions' ACID (Atomicity, Consistency, Isolation, Durability) qualities are ensured by the transaction manager. It controls how concurrent transactions are carried out, guarantees the atomicity and durability of transactions, and upholds data consistency.
4. Concurrency Control Manager: Oversees many users' or transactions' simultaneous access to the database. It uses strategies like locking, protocols based on timestamps, or optimistic concurrency management to guarantee transaction isolation and avoid data inconsistencies.
5. The database buffer manager is responsible for overseeing the memory buffer pool that holds frequently retrieved data pages from the disk. By using caching and buffering techniques, it reduces disk I/O operations and improves data retrieval efficiency.
6. By implementing authentication, authorization, and encryption procedures, the Security and Authorization Module protects data and manages database access. To keep sensitive information safe from unwanted access, it establishes user roles, privileges, and access control guidelines.
7. In the case of system failures, crashes, or disasters, the Database Recovery Manager manages database backup, restore, and recovery activities to guarantee data availability and integrity. It keeps track of transactions and uses recovery-oriented strategies like checkpointing and logging.
8. Data Dictionary/Metadata Repository: Holds metadata describing the structure, constraints, and schema of the database. To help with data management and schema evolution, it offers a centralized repository for storing data definitions, data relationships, and other database-related information.
9. Tools for Database Administrators (DBAs): Offers instruments and applications for managing databases, including construction, setup, tracking, and optimization of performance. It enables effective management of user accounts, rights, backups, and other administrative tasks by administrators.

4.6. ENTITY-RELATIONSHIP MODEL

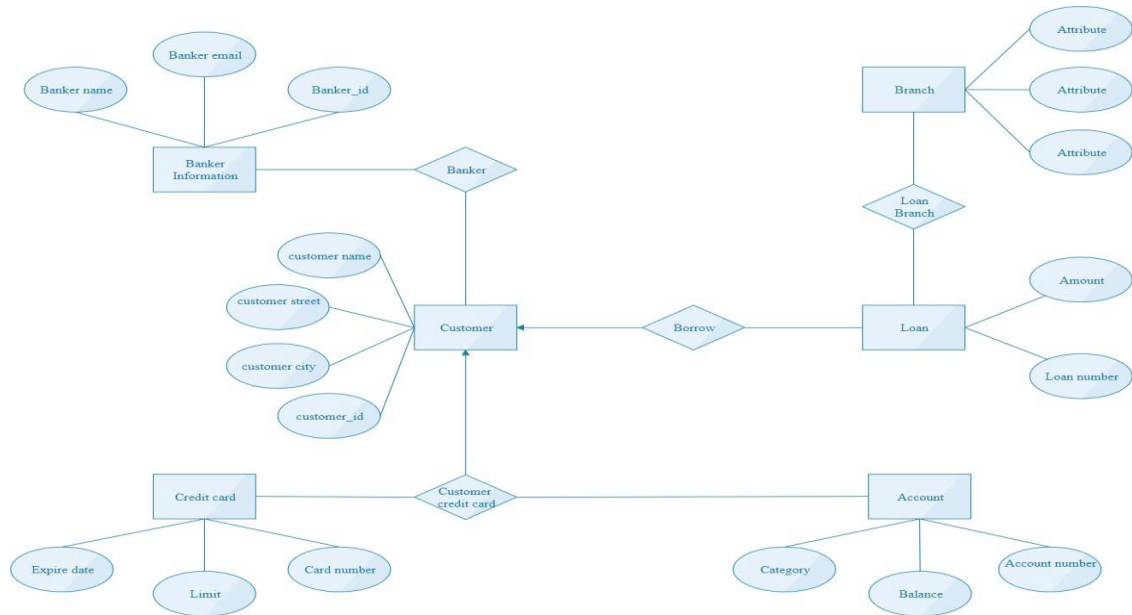
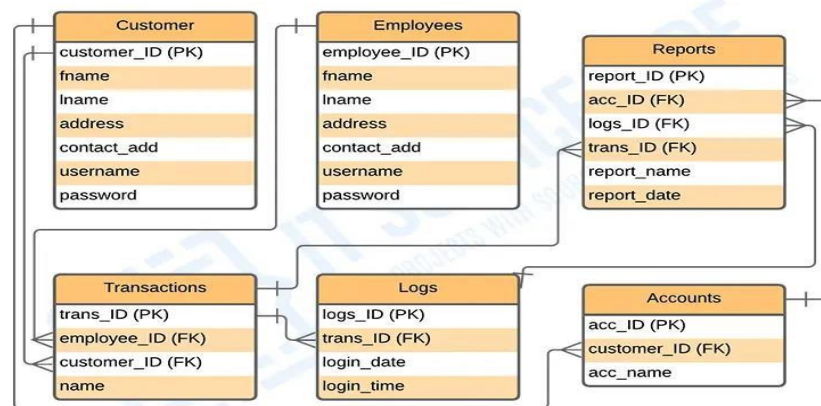


Fig.4.2. BANKING SYSTEM BMS ER MODEL

4.7. RELATIONAL SCHEMA

BANK MANAGEMENT SYSTEM



ENTITY RELATIONSHIP DIAGRAM

Fig.4.3. Relational schema

CHAPTER 5

FUNDAMENTALS OF SQL

5.1. INTRODUCTION

Understanding SQL's function as a standard language for relational database management is part of learning about SQL (Structured Query Language). SQL allows users to work with databases in a variety of ways, including entering, updating, removing, and querying data. It offers a strong and user-friendly interface for working with databases, which makes it an essential ability for database managers, data analysts, and developers alike. Furthermore, SQL is widely applicable across various platforms and industries due to its support by the majority of relational database management systems (RDBMS).

5.2. SQL COMMANDS

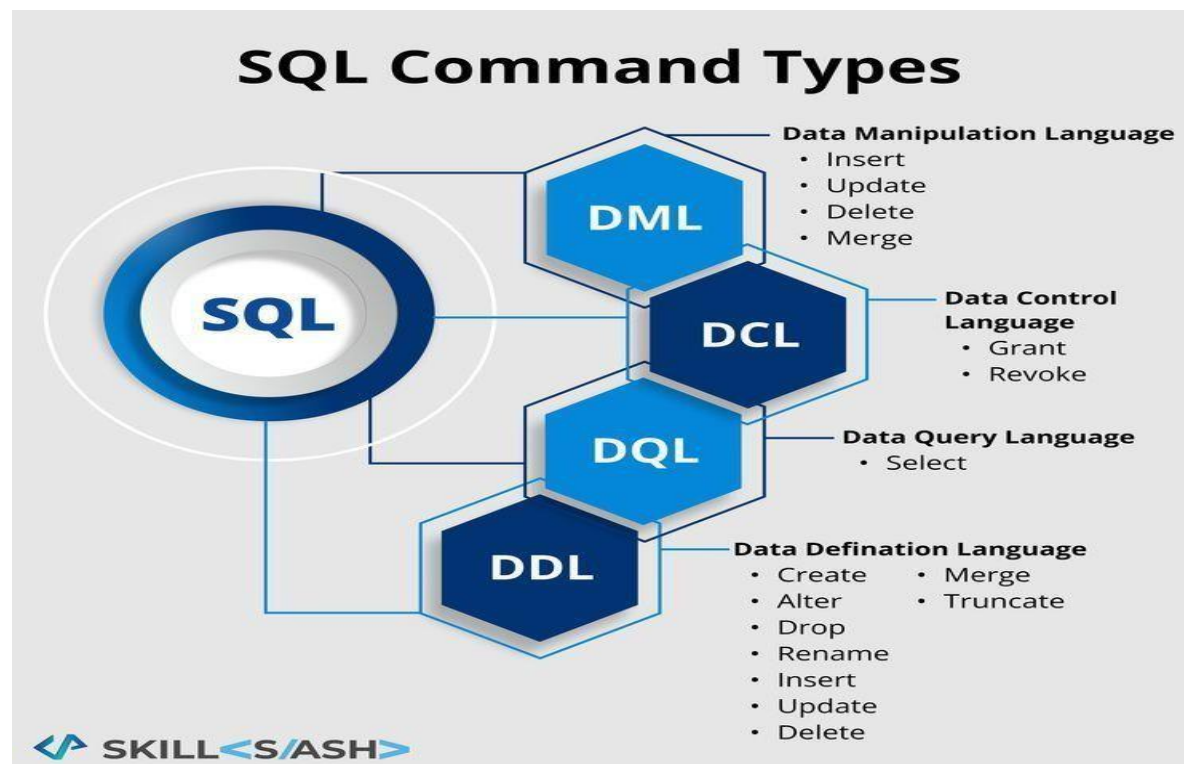


Fig.5.1. SQL COMMANDS

5.3. DATA DEFINITION LANGUAGE

DDL: It is a category of commands and managing the structure of a relational database.

DDL statements enables the users to create, modify, and delete database objects such as tables, indexes and views.

1.CREATE: Used to create the new database objects like tables, indexes, views and more.

Syntax: create table table_name;

2.ALTER: Enables the modification of the structure of the table.

Syntax: alter table table_name modify column column_name;

3.DROP: Used to deletes the indexes or databases objects.

Syntax: drop table table_name;

4.TRUNCATE: Removes all records which are saved before while preserving the table structure. It is faster than the delete process.

Syntax: truncate table table_name;

5.RENAME: It renames the table name or database name to a new one.

Syntax: rename old_table_tablename new_table_tablename;

5.4. DATA MANIPULATION LANGUAGE

DML: It uses for comprises the commands that allow users to interact with and manipulate data stored in a relational database.

1.UPDATE: Modifies existing records in a table based on specified condition.

Syntax: update table_name set col1=val1 where condition;

2.INSERT: Add new records into a table.

Syntax: insert into table(col1,col2) values(value1,value2);

3.DELETE: Removes the records from a table on specified conditions only.

Syntax: delete from table where condition;

5.5. DATA CONTROL LANGUAGE

DCL: It is a category of SQL that deals with the permissions and access control within a relational DBMS.

1.GRANT: Granting the permissions to user like select, insert, update and delete.

Syntax: grant select table_name to user-name;

2.REVOKE: Revokes all the permissions back from user.

Syntax: revoke select table_name from user_name;

5.6. TRANSACTION CONTROL LANGUAGE

TCL: It consists of a set of commands that manage transactions within a database. Transactions are units of work that can consist of one or more that can consist of one or more SQL statements and are typically used to ensure data integrity.

1.COMMIT: Commits the current transaction, making all changes made during the transaction permanent.

Syntax: commit;

2.ROLLBACK: Undoes the changes made during the current transaction, reverting the database to its state before the transaction began.

Syntax: rollback;

3.SAVEPOINT: Establishes the point within a transaction to which you can later roll back.

Syntax: savepoint to rollback;

5.7. DATA QUERY LANGUAGE

DQL: It is a subset of SQL that specially focuses on the retrieval of data from a database. The DQL is the “select” Statement, which allows the users to specify the columns in the table.

1.SELECT: Retrieves the table for displaying from the database.

Syntax: select * from table_name;

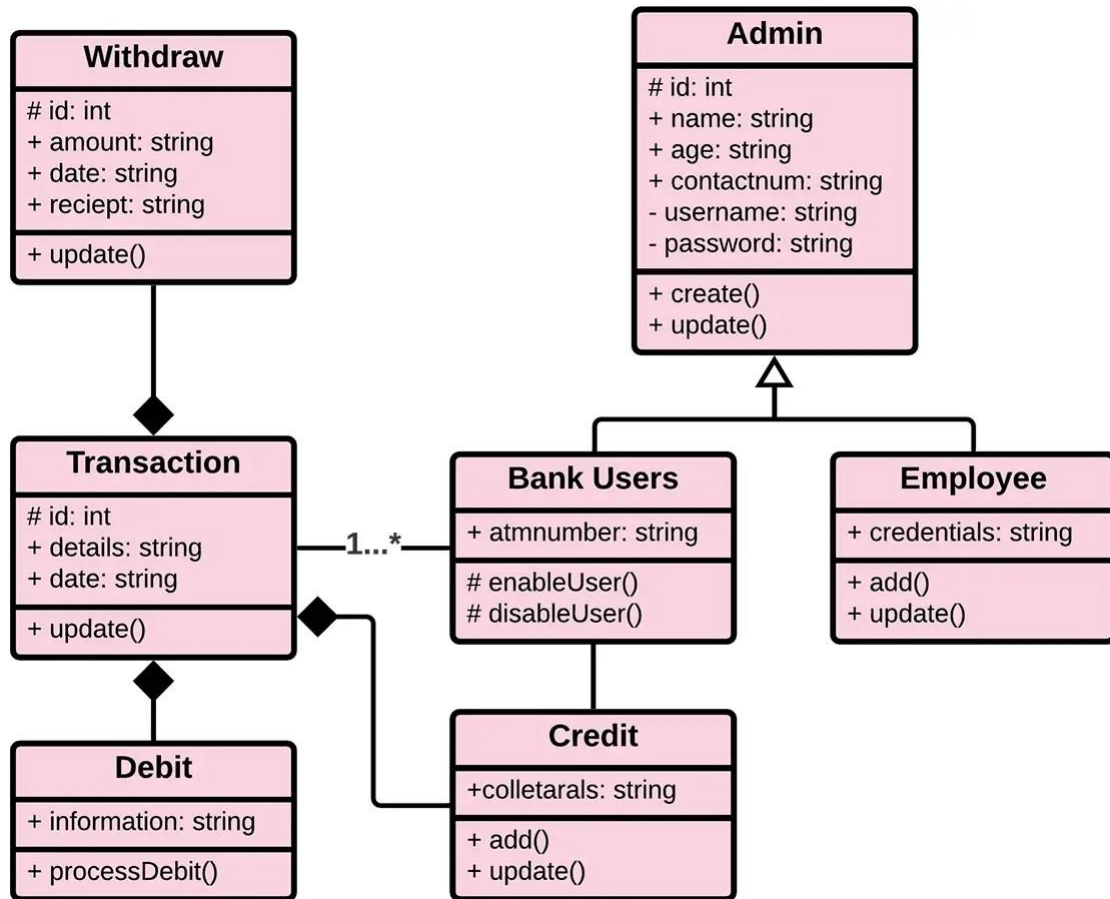
CHAPTER 6

DESIGN AND ARCHITECTURE

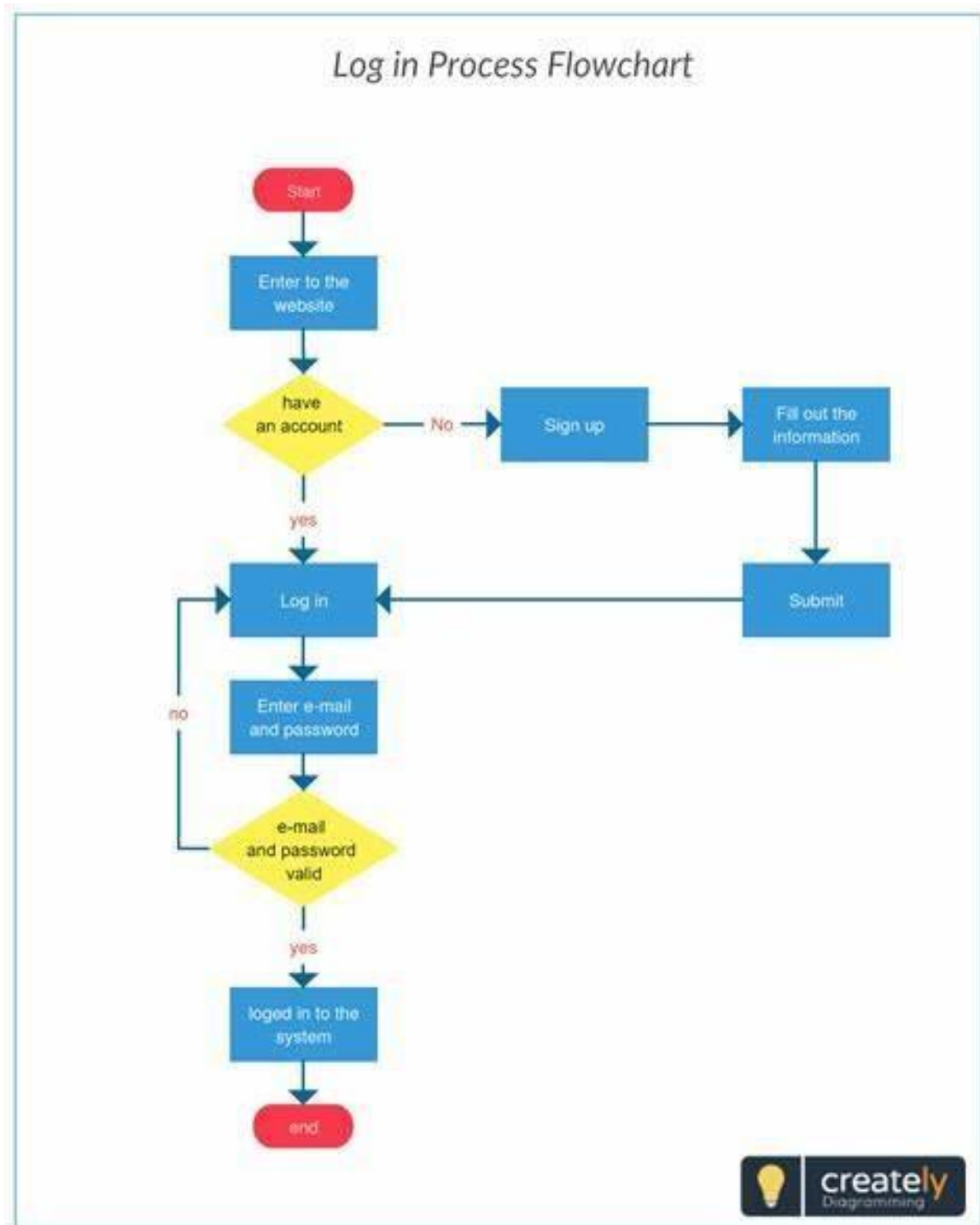
6.1. DESIGN GOALS

- 1.**SCALABILITY:** Ensures the system can handle a large number of concurrent users and exams without compromising performance.
- 2.**USER-FRIENDLY INTERFACE:** Design an intuitive and user-friendly interface for both exam takers and admins to enhance.
- 3.**SECURITY:** Implement robust security measures to prevent cheating, unauthorized access, and ensure the integrity of exam content.
- 4.**FLEXIBILITY:** Allow customization of exam parameters, question types and grading criteria to accommodate various educational needs and assessments formats.
- 5.**RELIABILITY:** Ensure high system availability and reliability to minimize disruptions during critical exam periods.

6.2. DATABASE STRUCTURES



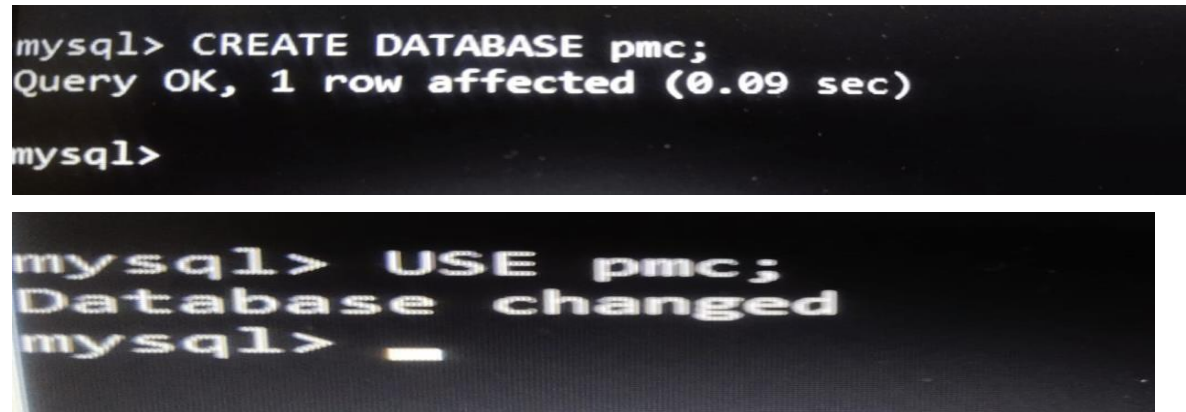
6.3. FLOW CHART



CHAPTER 7

IMPLEMENTATION

7.1. CREATING THE DATABASE



```
mysql> CREATE DATABASE pmc;
Query OK, 1 row affected (0.09 sec)

mysql>

mysql> USE pmc;
Database changed
mysql> _
```

7.2. CONNECTING THE DATABASE TO THE APPLICATION

```
Connection connection = null;
try {
    Class.forName(className:"com.mysql.cj.jdbc.Driver");
    connection = DriverManager.getConnection(
        url:"jdbc:mysql://localhost:3306/pmc",
        user:"ROOT",
        password:"ROOT123"
    );
}
```

7.3. PROCESSING QUERIES

1. User Password
- 2.create database pmc
- 3.use pmc
- 4.create table users
- 5.create table transactions

6.trigger

```
mysql> CREATE TABLE users(  
  -> user_id INT AUTO_INCREMENT PRIMARY KEY,  
  -> username VARCHAR(50) NOT NULL,  
  -> password VARCHAR(50) NOT NULL,  
  -> full_name VARCHAR(50) NOT NULL,  
  -> email VARCHAR(50) NOT NULL,  
  -> balance DECIMAL(10,2) DEFAULT 0.00,  
  -> active BOOLEAN DEFAULT TRUE  
  -> );  
Query OK, 0 rows affected (0.13 sec)
```

```
mysql> CREATE TABLE transactions(  
  -> transaction_id INT AUTO_INCREMENT PRIMARY KEY,  
  -> user_id INT NOT NULL,  
  -> transaction_type ENUM('DEPOSIT', 'WITHDRAWAL') NOT NULL,  
  -> amount DECIMAL(10,2) NOT NULL,  
  -> transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  -> FOREIGN KEY(user_id) REFERENCES users(user_id)  
  -> );  
Query OK, 0 rows affected (0.08 sec)  
  
mysql> _
```

```
mysql> delimiter ;  
mysql> create event if not exists MonthlyAddMoneyEvent  
  -> on schedule  
  -> every 1 month  
  -> starts timestamp(current_date) + interval 1 month + interval 11 day  
  -> comment 'Add money to account on the 5th day of every month'  
  -> Do  
  -> call AddMoneyToAccount();  
Query OK, 0 rows affected (0.02 sec)  
  
mysql> _
```

```
mysql> delimiter //
mysql> create procedure AddMoneyToAccount()
  -> Begin
  -> update users set balance = balance + 1000 where active = true;
  -> end //
Query OK, 0 rows affected (0.02 sec)
```

7.4. IMPLEMENTATION

```
1 public class App {
2     public static void main(String[] args) {
3         // Create an instance of the LoginSystem class to launch the login system
4         LogInGUI loginSystem = new LogInGUI();
5     }
6 }
7
```

MAIN

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.sql.*;
5
6  public class CheckBalanceGUI extends JFrame implements ActionListener {
7
8      JButton homeButton;
9      JLabel balanceLabel;
10     int userId; // User ID field to store the user ID
11
12     public CheckBalanceGUI(int userId) {
13         this.userId = userId; // Store the user ID
14         setTitle(title:"Check Balance");
15         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16         setSize(width:400, height:300);
17
18         JPanel panel = new JPanel(new GridLayout(rows:2, cols:1));
19
20         balanceLabel = new JLabel(text:"Your Balance: ");
21         homeButton = new JButton(text:"Home");
22
23         homeButton.addActionListener(this);
24
25         panel.add(balanceLabel);
26         panel.add(homeButton);
27
28         add(panel);
29         setVisible(b:true);
30
31         // Fetch balance from database
32         fetchBalanceFromDatabase();
33     }
34
35     public void fetchBalanceFromDatabase() {
36         // Connect to the database
37         Connection connection = null;
```

```
37     Connection connection = null;
38     try {
39         Class.forName(className:"com.mysql.cj.jdbc.Driver");
40         connection = DriverManager.getConnection(
41             url:"jdbc:mysql://localhost:3306/pmc",
42             user:"root",
43             password:"root"
44         );
45
46         // Query to fetch balance from database
47         String query = "SELECT balance FROM users WHERE user_id = ?";
48         PreparedStatement preparedStatement = connection.prepareStatement(query);
49         preparedStatement.setInt(parameterIndex:1, userId);
50
51         ResultSet resultSet = preparedStatement.executeQuery();
52
53         if (resultSet.next()) {
54             double balance = resultSet.getDouble(columnLabel:"balance");
55             balanceLabel.setText("Your Balance: $" + balance);
56         } else {
57             balanceLabel.setText(text:"Error: Balance not found");
58         }
59
60         resultSet.close();
61         preparedStatement.close();
62         connection.close();
63     } catch (Exception e) {
64         e.printStackTrace();
65         balanceLabel.setText(text:"Error: Unable to fetch balance");
66     }
67 }
68
69 public void actionPerformed(ActionEvent e) {
70     if (e.getSource() == homeButton) {
71         HomeGUI hg= new HomeGUI(userId);
72         dispose(); // Close current window
73     }
```

CHECK BALANCE


```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.sql.*;
5
6  public class DepositGUI extends JFrame implements ActionListener {
7      private int userId; // User ID field to store the user ID
8
9      JTextField amountField;
10     JButton depositButton, homeButton;
11
12     public DepositGUI(int userId) {
13         this.userId = userId; // Store the user ID
14         setTitle(title:"Deposit");
15         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16         setSize(width:400, height:300);
17
18         JPanel panel = new JPanel(new GridLayout(rows:3, cols:1));
19
20         JLabel amountLabel = new JLabel(text:"Enter Amount to Deposit:");
21         amountField = new JTextField();
22         depositButton = new JButton(text:"Deposit");
23         homeButton = new JButton(text:"Home");
24
25         depositButton.addActionListener(this);
26         homeButton.addActionListener(this);
27
28         panel.add(amountLabel);
29         panel.add(amountField);
30         panel.add(depositButton);
31         panel.add(homeButton);
32
33         add(panel);
34         setVisible(b:true);
35     }
36
37     public void actionPerformed(ActionEvent e) {
```

```
37 public void actionPerformed(ActionEvent e) {
38     if (e.getSource() == depositButton) {
39         depositAmount();
40     } else if (e.getSource() == homeButton) {
41         new HomeGUI(userId);
42         dispose();
43     }
44 }
45
46 public void depositAmount() {
47     try {
48         double amount = Double.parseDouble(amountField.getText());
49
50         Connection connection = DriverManager.getConnection(url:"jdbc:mysql://localhost:3306/pmc", user:"root", password:"root");
51
52         String updateBalanceQuery = "UPDATE balance SET amount = ? WHERE user_id = ?";
53         PreparedStatement updateBalanceStatement = connection.prepareStatement(updateBalanceQuery);
54         updateBalanceStatement.setDouble(parameterIndex:1, amount);
55         updateBalanceStatement.setInt(parameterIndex:2, userId);
56         updateBalanceStatement.executeUpdate();
57
58         String recordTransactionQuery = "INSERT INTO transactions (user_id, transaction_type, amount) VALUES (?, 'DEPOSIT', ?)";
59         PreparedStatement recordTransactionStatement = connection.prepareStatement(recordTransactionQuery);
60         recordTransactionStatement.setInt(parameterIndex:1, userId);
61         recordTransactionStatement.setDouble(parameterIndex:2, amount);
62         recordTransactionStatement.executeUpdate();
63
64         JOptionPane.showMessageDialog(this, message:"Deposit successful");
65
66         connection.close();
67     } catch (NumberFormatException | SQLException ex) {
68         ex.printStackTrace();
69         JOptionPane.showMessageDialog(this, message:"Error occurred");
70     }
71 }
```

DEPOSIT

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  public class HomeGUI extends JFrame implements ActionListener {
6      private int userId; // User ID field to store the user ID
7
8      JButton depositButton, withdrawButton, balanceButton, signOutButton;
9
10     // Default constructor
11     public HomeGUI() {
12         setTitle(title:"Home");
13         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14         setSize(width:400, height:300);
15
16         JPanel panel = new JPanel(new GridLayout(rows:4, cols:1));
17
18         depositButton = new JButton(text:"Deposit");
19         withdrawButton = new JButton(text:"Withdraw");
20         balanceButton = new JButton(text:"Check Balance");
21         signOutButton = new JButton(text:"Sign Out");
22
23         depositButton.addActionListener(this);
24         withdrawButton.addActionListener(this);
25         balanceButton.addActionListener(this);
26         signOutButton.addActionListener(this);
27
28         panel.add(depositButton);
29         panel.add(withdrawButton);
30         panel.add(balanceButton);
31         panel.add(signOutButton);
32
33         add(panel);
34         setVisible(b:true);
35     }
36 }
```



```
34         setVisible(b:true);
35     }
36
37     // Constructor with user ID parameter
38     public HomeGUI(int userId) {
39         this(); // Call the default constructor to initialize JFrame
40         this.userId = userId; // Store the user ID
41     }
42
43     public void actionPerformed(ActionEvent e) {
44         if (e.getSource() == depositButton) {
45             DepositGUI dh = new DepositGUI(userId);
46             dispose();
47         } else if (e.getSource() == withdrawButton) {
48             WithdrawGUI wg = new WithdrawGUI(userId);
49             dispose();
50         } else if (e.getSource() == balanceButton) {
51             CheckBalanceGUI cb = new CheckBalanceGUI(userId);
52             dispose();
53         } else if (e.getSource() == signOutButton) {
54             LogInGUI lg = new LogInGUI();
55             dispose();
56         }
57     }
58 }
59
```

```
1  import javax.swing.*;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import java.sql.Connection;
5  import java.sql.DriverManager;
6  import java.sql.PreparedStatement;
7  import java.sql.ResultSet;
8
9  public class LogInGUI {
10     JFrame frame = new JFrame();
11     private JLabel usernameLabel, passwordLabel, messageLabel;
12     private JTextField usernameField;
13     private JPasswordField passwordField;
14     private JButton logInButton, signUpButton;
15
16     private static int userId; // Static variable to store the user ID
17
18     public static int getUserId() {
19         return userId;
20     }
21
22     public static void setUserId(int id) {
23         userId = id;
24     }
25
26     LogInGUI(){
27         usernameLabel = new JLabel(text:"Username");
28         usernameLabel.setBounds(x:50,y:100,width:75,height:25);
29         passwordLabel = new JLabel(text:"Password");
30         passwordLabel.setBounds(x:50,y:150,width:75,height:25);
31         messageLabel = new JLabel(text:"");
32         messageLabel.setBounds(x:125,y:250,width:250,height:35);
33
34         usernameField = new JTextField();
35         usernameField.setBounds(x:125,y:100,width:200,height:25);
36         passwordField = new JPasswordField();
```

PENSION MANAGEMENT SYSTEM

```
37 passwordField.setBounds(x:125,y:150,width:200,height:25);
38
39 loginButton = new JButton(text:"Log In");
40 loginButton.setBounds(x:150,y:200,width:100,height:25);
41 loginButton.addActionListener(new ActionListener() {
42     @Override
43     public void actionPerformed(ActionEvent e) {
44         try {
45             Connection connection = DriverManager.getConnection(url:"jdbc:mysql://localhost:3306/pmc", user:"root", password:"root123");
46             PreparedStatement ps = connection.prepareStatement(sql:"Select user_id, username, password from users where username=? and pa");
47             ps.setString(1, usernameField.getText());
48             ps.setString(2, passwordField.getText());
49             ResultSet rs = ps.executeQuery();
50             if (rs.next()) {
51                 // Set the user ID globally
52                 int userId = rs.getInt(columnLabel:"user_id");
53                 loginGUI.setUserId(userId);
54
55                 // Open HomeGUI
56                 HomeGUI HGUI = new HomeGUI(loginGUI.getUserId());
57                 frame.dispose();
58             } else {
59                 messageLabel.setText(text:"Wrong username or password");
60             }
61             connection.close();
62         } catch (Exception exception) {
63             exception.printStackTrace();
64         }
65     }
66 }
67
68 signUpButton = new JButton(text:"Sign Up");
69 signUpButton.setBounds(x:150,y:250,width:100,height:25);
70 signUpButton.addActionListener(new ActionListener() {
71     @Override
72     public void actionPerformed(ActionEvent e) {
```

```
72     public void actionPerformed(ActionEvent e) {
73         frame.dispose();
74         signUPGUI sg = new signUPGUI();
75     }
76 ;
77
78 ame.add(usernameLabel);
79 ame.add(usernameField);
80 ame.add(passwordLabel);
81 ame.add(passwordField);
82 ame.add(loginButton);
83 ame.add(messageLabel);
84 ame.add(signUpButton);
85
86 ame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
87 ame.setSize(width:420,height:420);
88 ame.setLayout(manager:null);
89 ame.setVisible(b:true);
90
91
92 Run | Debug
93 static void main(String[] args) {
94     w loginGUI();
95
96 }
```

LOGIN PAGE

```
1 import javax.swing.*;
2 import java.awt.event.ActionEvent;
3 import java.awt.event.ActionListener;
4 import java.sql.Connection;
5 import java.sql.DriverManager;
6 import java.sql.PreparedStatement;
7
8 public class signUPGUI {
9     JFrame frame = new JFrame();
10    private JLabel usernameLabel, passwordLabel, fullnameLabel, emailLabel, messageLabel;
11
12    // Declaring Text Fields
13    private JTextField usernameField, fullnameField, emailField;
14    private JPasswordField passwordField;
15    // Declaring Buttons
16    private JButton signUpButton;
17
18    signUPGUI() {
19        usernameLabel = new JLabel(text:"Username");
20        usernameLabel.setBounds(x:50, y:50, width:75, height:25);
21        fullnameLabel = new JLabel(text:"Full Name");
22        fullnameLabel.setBounds(x:50, y:100, width:75, height:25);
23        emailLabel = new JLabel(text:"Email");
24        emailLabel.setBounds(x:50, y:150, width:75, height:25);
25        passwordLabel = new JLabel(text:"Password");
26        passwordLabel.setBounds(x:50, y:200, width:75, height:25);
27        messageLabel = new JLabel(text:"");
28        messageLabel.setBounds(x:125, y:250, width:250, height:35);
29        // TextFields
30        usernameField = new JTextField();
31        usernameField.setBounds(x:125, y:50, width:200, height:25);
32        fullnameField = new JTextField();
33        fullnameField.setBounds(x:125, y:100, width:200, height:25);
34        emailField = new JTextField();
35        emailField.setBounds(x:125, y:150, width:200, height:25);
36        passwordField = new JPasswordField();
37        passwordField.setBounds(x:125, y:200, width:200, height:25);
```

```
38 // Buttons
39 signUpButton = new JButton(text:"Sign Up");
40 signUpButton.setBounds(x:150, y:300, width:100, height:25);
41 signUpButton.addActionListener(new ActionListener() {
42     @Override
43     public void actionPerformed(ActionEvent e) {
44         try {
45             Connection connection = DriverManager.getConnection(url:"jdbc:mysql://localhost:3306/pmc", user:"root", password:"
46             String query = "INSERT INTO users (username, full_name, email, password) VALUES (?, ?, ?, ?)";
47             PreparedStatement ps = connection.prepareStatement(query);
48             ps.setString(parameterIndex:1, usernameField.getText());
49             ps.setString(parameterIndex:2, fullnameField.getText());
50             ps.setString(parameterIndex:3, emailField.getText());
51             ps.setString(parameterIndex:4, passwordField.getText());
52             ps.executeUpdate();
53             connection.close();
54         } catch (Exception exception) {
55             exception.printStackTrace();
56         }
57         LoginGUI lg = new LoginGUI();
58         frame.dispose();
59     }
60 });
61 // Adding components to frame
62 frame.add(usernameLabel);
63 frame.add(usernameField);
64 frame.add(fullnameLabel);
65 frame.add(fullnameField);
66 frame.add(emailLabel);
67 frame.add(emailField);
68 frame.add(passwordLabel);
69 frame.add(passwordField);
70 frame.add(messageLabel);
71 frame.add(signUpButton);
72 // Setting up the frame
73 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
74
75 frame.add(passwordLabel);
76 frame.add(passwordField);
77 frame.add(messageLabel);
78 frame.add(signUpButton);
79 // Setting up the frame
80 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
81 frame.setSize(width:420, height:420);
82 frame.setLayout(manager:null);
83 frame.setVisible(b:true);
84 }
```

SIGN UP PAGE


```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.sql.*;
5
6  public class WithdrawGUI extends JFrame implements ActionListener {
7      private int userId; // User ID field to store the user ID
8
9      JTextField amountField;
10     JButton withdrawButton, homeButton;
11
12     public WithdrawGUI(int userId) {
13         this.userId = userId; // Store the user ID
14         setTitle(title:"Withdraw");
15         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16         setSize(width:400, height:300);
17
18         JPanel panel = new JPanel(new GridLayout(rows:3, cols:1));
19
20         JLabel amountLabel = new JLabel(text:"Enter Amount to Withdraw:");
21         amountField = new JTextField();
22         JButton homeButton = new JButton(text:"Withdraw");
23         homeButton = new JButton(text:"Home");
24
25         withdrawButton.addActionListener(this);
26         homeButton.addActionListener(this);
27
28         panel.add(amountLabel);
29         panel.add(amountField);
30         panel.add(withdrawButton);
31         panel.add(homeButton);
32
33         add(panel);
34         setVisible(b:true);
35     }
36
37     public void actionPerformed(ActionEvent e) {
```

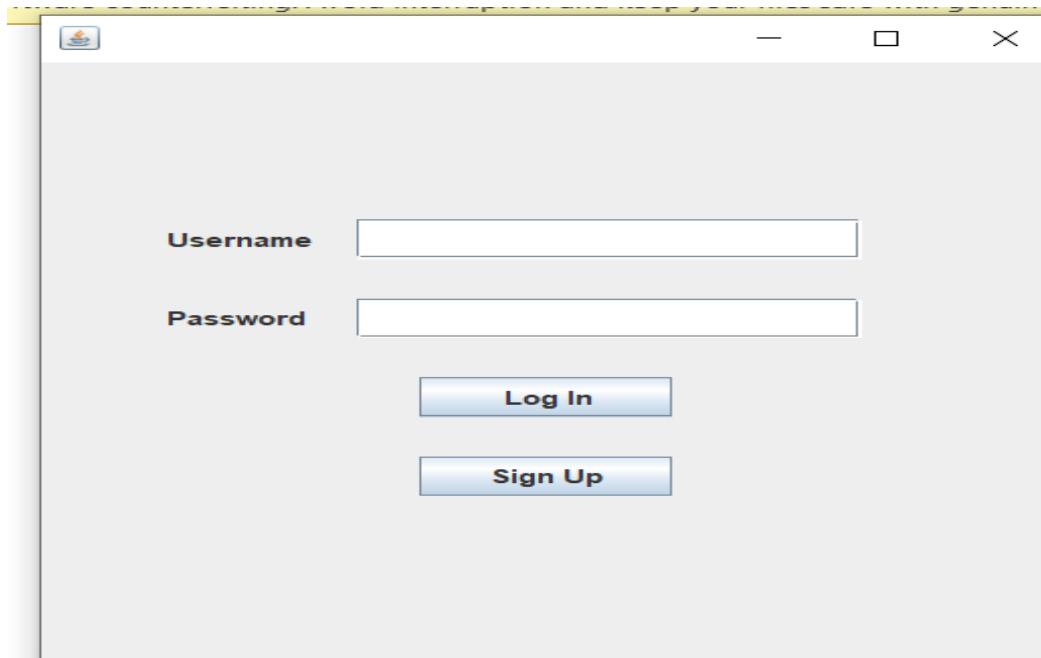
PENSION MANAGEMENT SYSTEM

```
37 public void actionPerformed(ActionEvent e) {
38     if (e.getSource() == withdrawButton) {
39         withdrawAmount();
40     } else if (e.getSource() == homeButton) {
41         HomeGUI gh= new HomeGUI(userId); // Pass user ID to HomeGUI
42         dispose(); // Close current window
43     }
44 }
45
46 public void withdrawAmount() {
47     try {
48         double amount = Double.parseDouble(amountField.getText());
49
50         // Connect to the database
51         Connection connection = DriverManager.getConnection(
52             url:"jdbc:mysql://localhost:3306/pmc",
53             user:"root",
54             password:"root"
55         );
56
57         // Check user's balance
58         String checkBalanceQuery = "SELECT balance FROM users WHERE user_id = ?";
59         PreparedStatement checkBalanceStatement = connection.prepareStatement(checkBalanceQuery);
60         checkBalanceStatement.setInt(parameterIndex:1, userId); // Use the stored user ID
61         ResultSet balanceResult = checkBalanceStatement.executeQuery();
62         if (balanceResult.next()) {
63             double currentBalance = balanceResult.getDouble(columnLabel:"balance");
64             if (amount > currentBalance) {
65                 JOptionPane.showMessageDialog(this, message:"Insufficient balance");
66             } else {
67                 // Update user's balance
68                 String updateBalanceQuery = "UPDATE users SET balance = balance - ? WHERE user_id = ?";
69                 PreparedStatement updateBalanceStatement = connection.prepareStatement(updateBalanceQuery);
70                 updateBalanceStatement.setDouble(parameterIndex:1, amount);
71                 updateBalanceStatement.setInt(parameterIndex:2, userId);
72                 updateBalanceStatement.executeUpdate();
73
74                 // Record transaction
75                 String recordTransactionQuery = "INSERT INTO transactions (user_id, transaction_type, amount) VALUES (?, 'WITHDRAWAL'";
76                 PreparedStatement recordTransactionStatement = connection.prepareStatement(recordTransactionQuery);
77                 recordTransactionStatement.setInt(parameterIndex:1, userId);
78                 recordTransactionStatement.setDouble(parameterIndex:2, amount);
79                 recordTransactionStatement.executeUpdate();
80
81                 JOptionPane.showMessageDialog(this, message:"Withdrawal successful");
82             }
83         } else {
84             JOptionPane.showMessageDialog(this, message:"User not found");
85         }
86
87         connection.close();
88     } catch (NumberFormatException ex) {
89         JOptionPane.showMessageDialog(this, message:"Invalid amount");
90     } catch (SQLException ex) {
91         ex.printStackTrace();
92         JOptionPane.showMessageDialog(this, message:"Error occurred");
93     }
94 }
95
96
97 }
98
```

WITHDRAWAL

CHAPTER 8

RESULTS



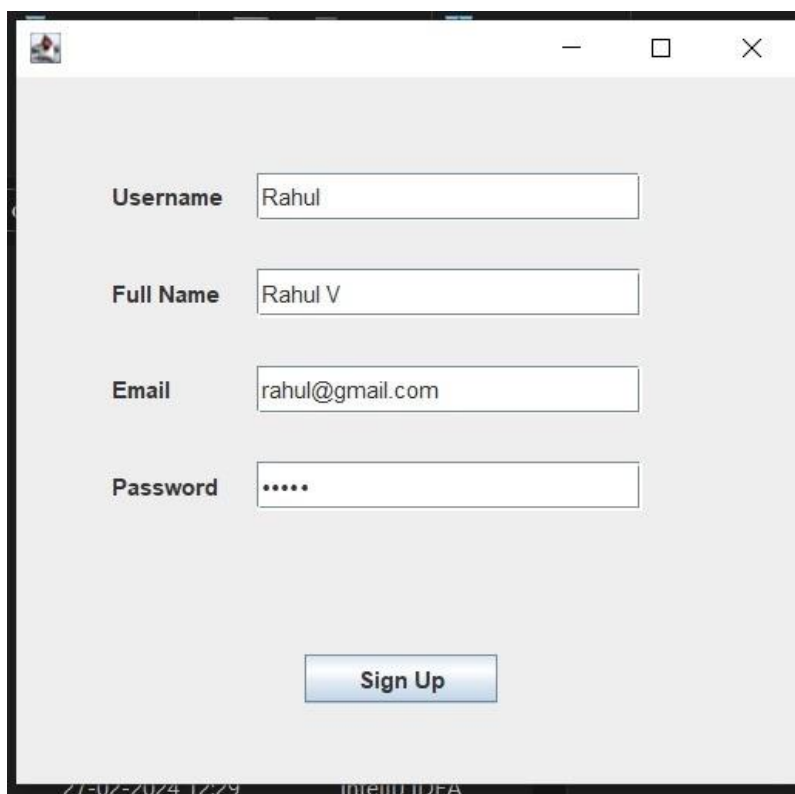
A screenshot of a web application window titled "Pension Management System". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. The main content area is light gray and contains two input fields: "Username" and "Password". Below these fields are two buttons: "Log In" and "Sign Up".

Username

Password

Log In

Sign Up



A screenshot of a web application window titled "Pension Management System". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. The main content area is light gray and contains four input fields: "Username", "Full Name", "Email", and "Password". Below these fields is a "Sign Up" button. The "Username" field contains "Rahul", the "Full Name" field contains "Rahul V", the "Email" field contains "rahul@gmail.com", and the "Password" field contains ".....".

Username

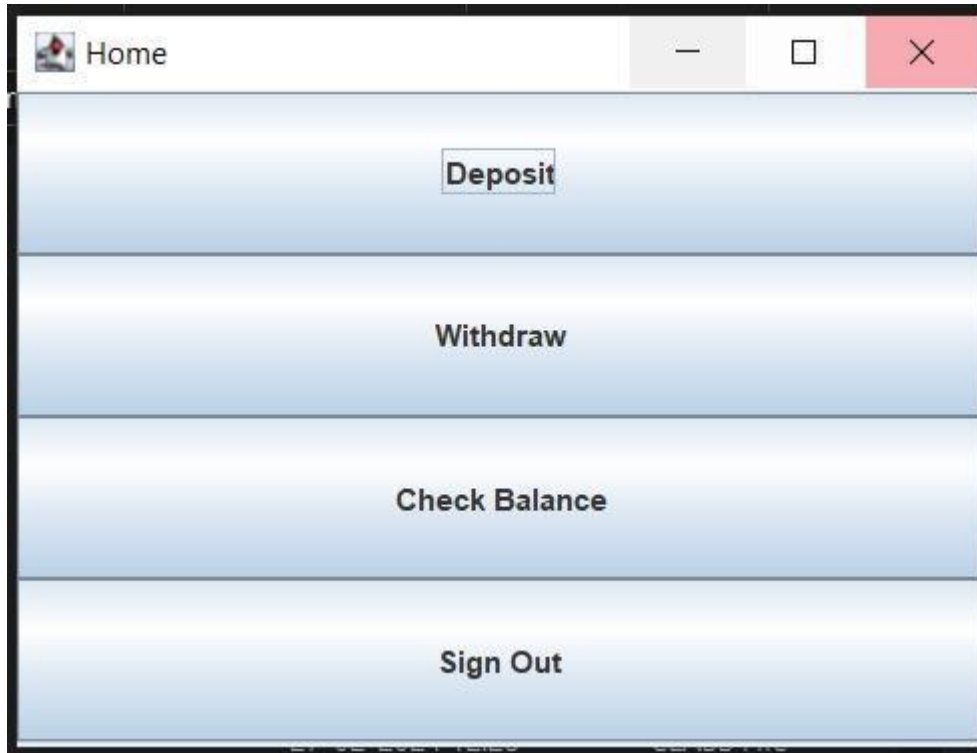
Full Name

Email

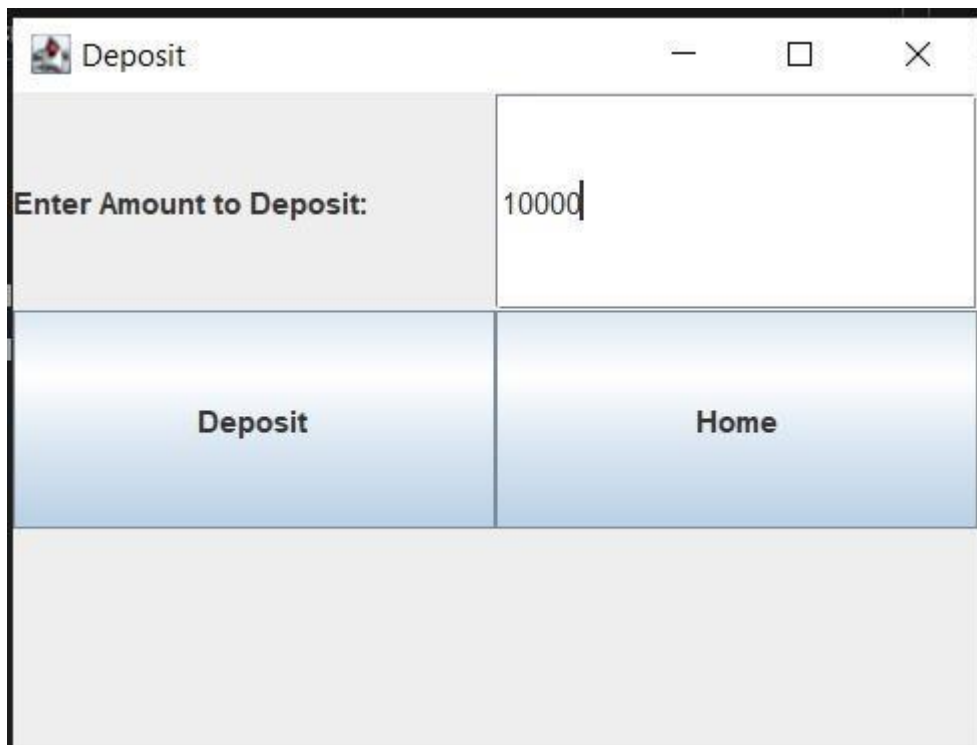
Password

Sign Up

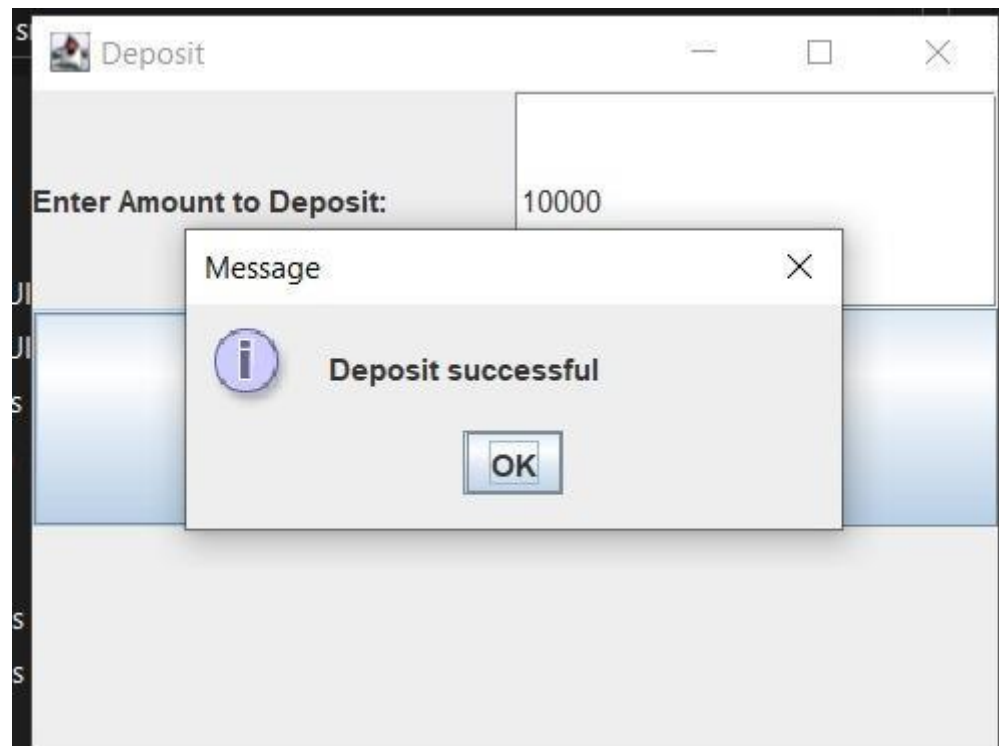
27-02-2024 12:29 IntelliJ IDEA



A screenshot of a web application window titled "Home". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. The main content area is a light blue gradient with four buttons stacked vertically: "Deposit", "Withdraw", "Check Balance", and "Sign Out".



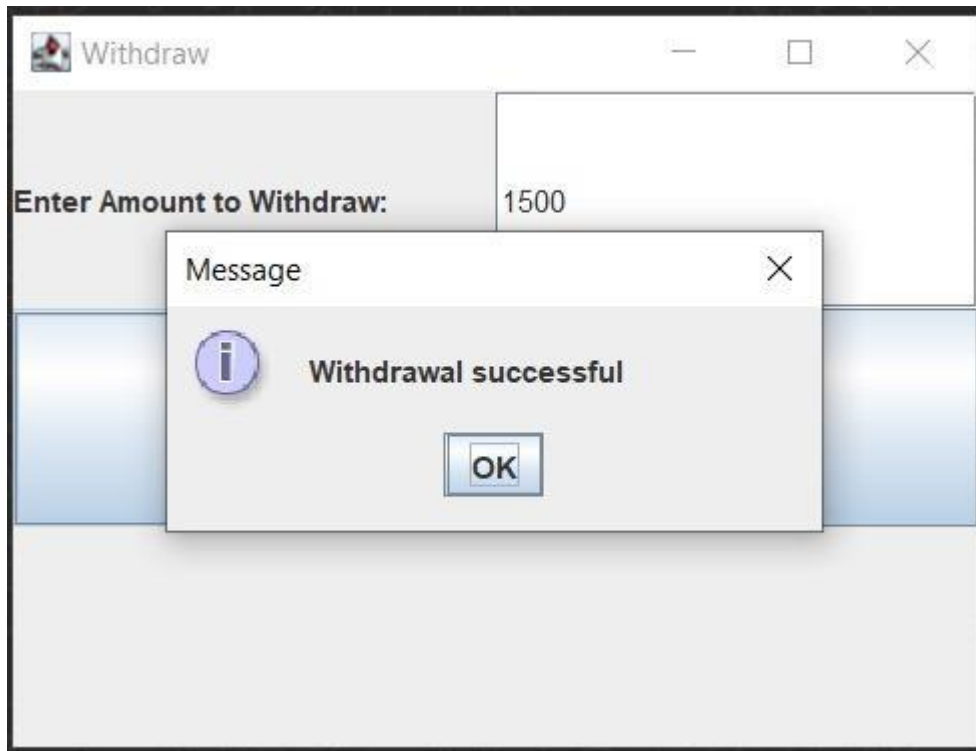
A screenshot of a web application window titled "Deposit". The window has a standard Windows-style title bar with a minimize button, a maximize button, and a close button. The main content area is divided into three sections. The top section has a label "Enter Amount to Deposit:" on the left and a text input field on the right containing the value "10000". The bottom section is divided into two buttons: "Deposit" on the left and "Home" on the right. The bottom-most section of the window is a solid light gray area.



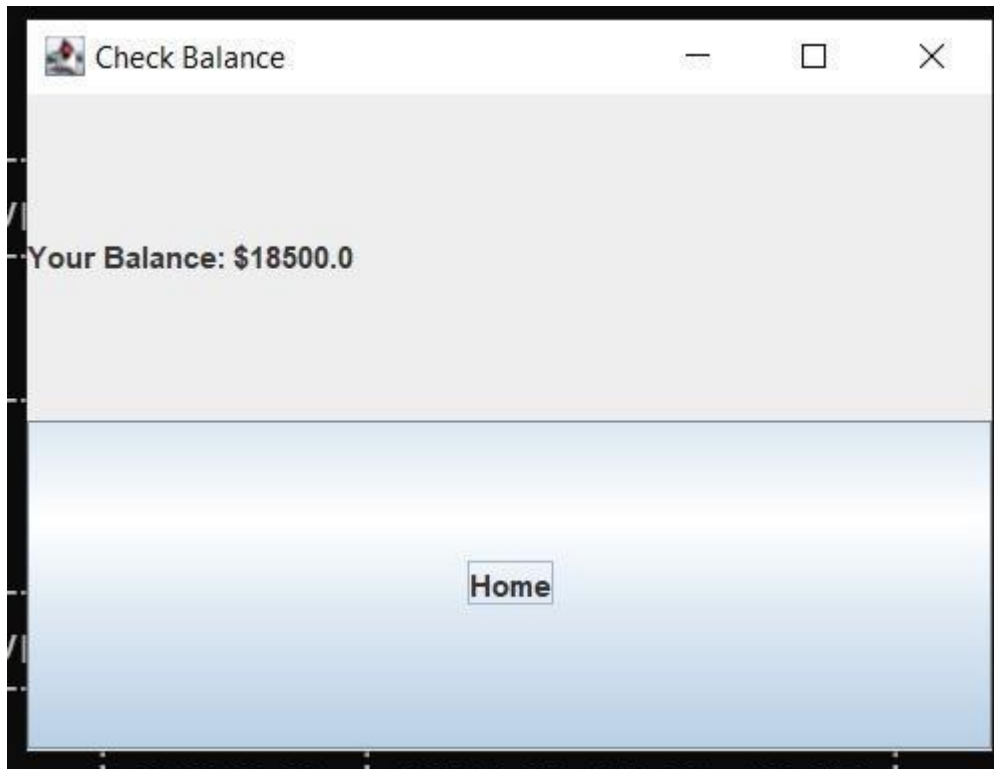
```
mysql> select * from transactions;
+-----+-----+-----+-----+-----+
| transaction_id | user_id | transaction_type | amount | transaction_date |
+-----+-----+-----+-----+-----+
| 3 | 2 | DEPOSIT | 10000.00 | 2024-02-27 12:59:40 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> |
```

```
mysql> select * from transactions;
+-----+-----+-----+-----+-----+
| transaction_id | user_id | transaction_type | amount | transaction_date |
+-----+-----+-----+-----+-----+
| 3 | 2 | DEPOSIT | 10000.00 | 2024-02-27 12:59:40 |
| 4 | 2 | DEPOSIT | 10000.00 | 2024-02-27 13:00:07 |
+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```



```
mysql> select * from transactions;
+-----+-----+-----+-----+-----+
| transaction_id | user_id | transaction_type | amount | transaction_date |
+-----+-----+-----+-----+-----+
| 3 | 2 | DEPOSIT | 10000.00 | 2024-02-27 12:59:40 |
| 4 | 2 | DEPOSIT | 10000.00 | 2024-02-27 13:00:07 |
| 5 | 2 | WITHDRAWAL | 1500.00 | 2024-02-27 13:00:34 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```



```
mysql> select * from users;
+-----+-----+-----+-----+-----+-----+-----+
| user_id | username | password | full_name | email | balance | active |
+-----+-----+-----+-----+-----+-----+-----+
| 2 | Rahul | rahul | Rahul V | rahul@gmail.com | 18500.00 | 1 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

CHAPTER 9

CONCLUSION

"In summary, the creation of the Pension Fund Management System marks a critical turning point in the modernization and simplification of pension fund operations. During the project, we have effectively recognized and tackled the essential requirements for managing pension funds, such as effective documentation, precise computations, and a smooth user interface.

Robust security measures are incorporated into our complete system to protect pensioners' interests and ensure regulatory compliance while safeguarding sensitive data. Furthermore, the addition of automated elements has greatly decreased human mistake and increased overall process efficiency in fund administration.

Through the utilization of state-of-the-art technologies like blockchain and artificial intelligence, we have established the basis for a system that is both scalable and flexible enough to fulfill the changing needs of pension fund management. Furthermore, our intuitive design and user-friendly interface enable administrators to confidently carry out their tasks and manage the system with ease.

In the future, the Pension Fund Management System could have a positive influence on millions of retirees' lives globally in addition to improving the operational efficacy of pension funds. We are prepared to lead innovation and provide long-term solutions that advance retirement well-being and financial stability with ongoing support and cooperation."

❖ REFERENCES:

- <https://ui.shadcn.com/docs>
- <https://react.dev/>
- <https://www.baeldung.com/building-a-restful-web-service-with-spring-and-java-based-configuration>
- Google
- Youtube
- Geeks for Geeks