OOADJ-Mini project

Bank Management System

Team: A2

Aakash Negi	**PES1UG20CS002**
Akash Agarwal	**PES1UG20CS026**
Advaita Bhargava	**PES1UG20CS023**
Akhil Kumar Elango	**PES1UG20CS029**

Synopsis

The Bank Management System Java project is a software application designed to assist bank staff in efficiently managing various banking operations, such as customer account creation, deposit, withdrawal transactions and employee management.

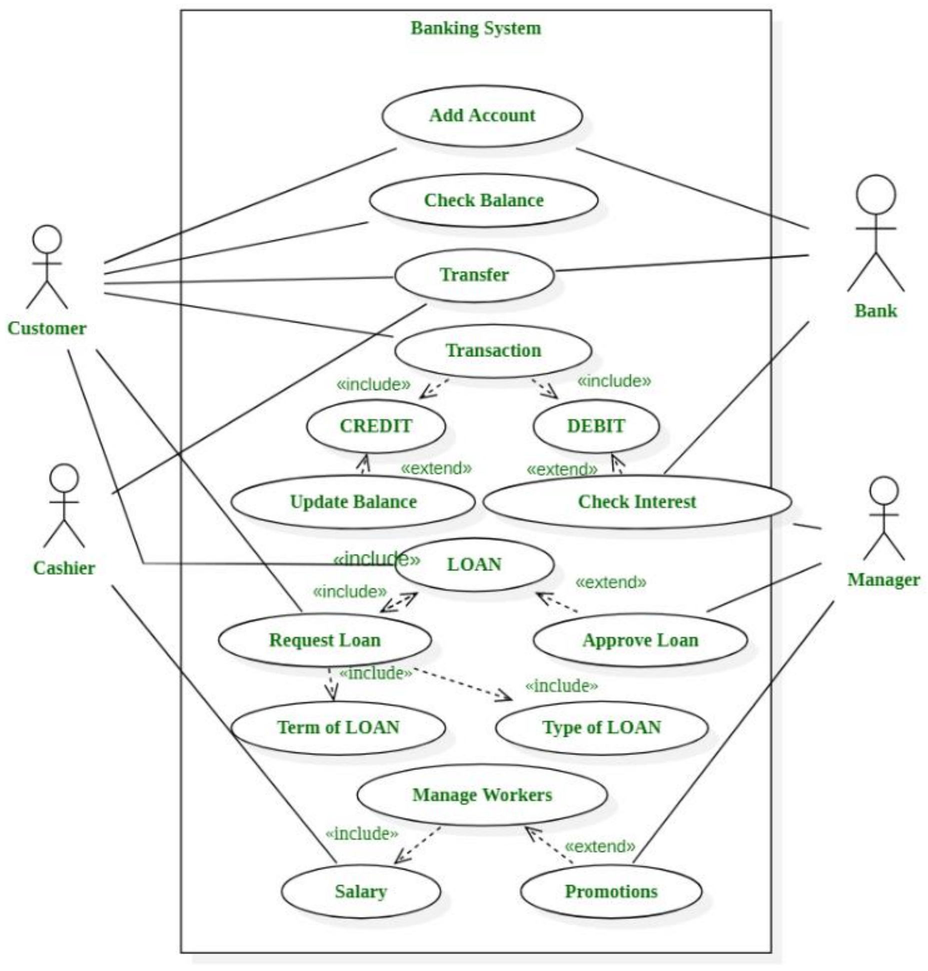
The system is designed with a user-friendly interface and robust functionalities that streamline the banking processes, thereby reducing manual workloads and errors. The project has various modules, including customer module, account module, transaction module and employee module.

In the customer module, customers can create their accounts, view their account details, and update their personal information. In the account module, bank staff can manage customer accounts, such as creating new accounts, deleting existing accounts, and updating account information. The transaction module handles customer transactions such as deposit and withdrawal.

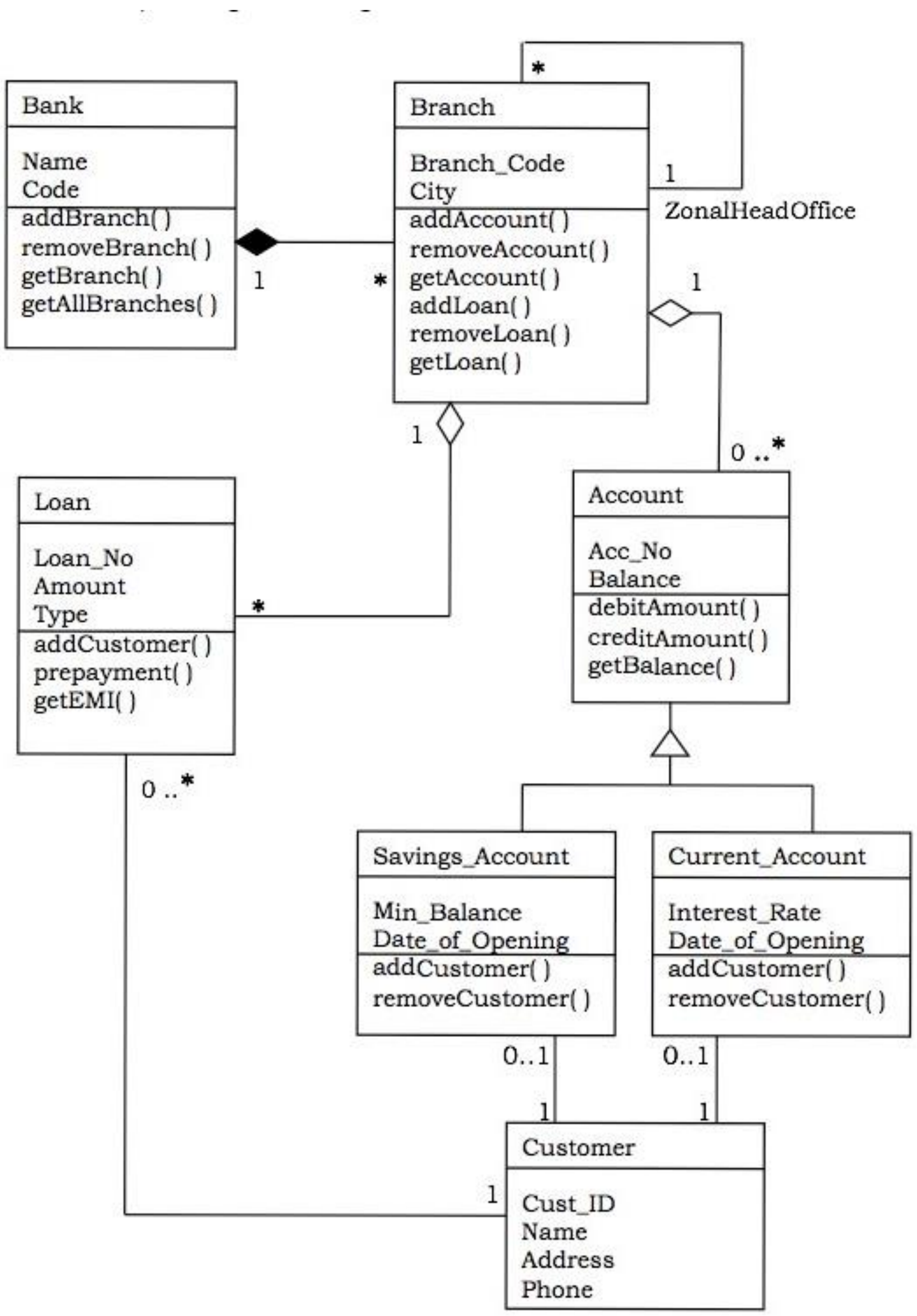
The system is built using Java programming language and MySQL database to store data.

Overall, the Bank Management System Java project provides an efficient and secure solution for bank management and is an ideal choice for banks looking to automate their operations and streamline their services.

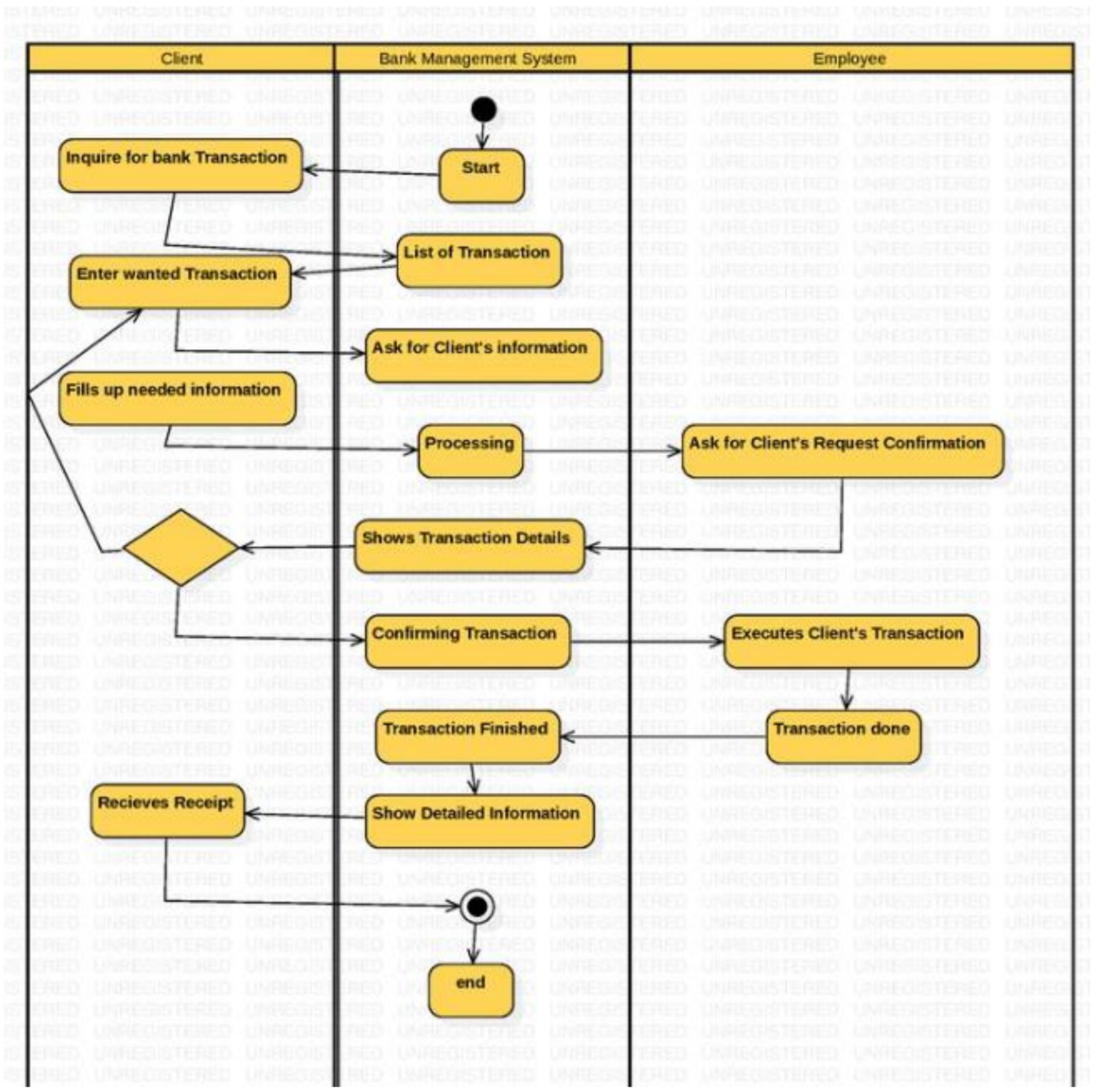
Use Case Diagram



Class Diagram



Activity and State Diagram


Client
Bank Management System
Employee
Inquire for bank Transaction
Start
List of Transaction
Enter wanted Transaction
Ask for Client's information
Fills up needed information
Processing
Ask for Client's Request Confirmation
Shows Transaction Details
Confirming Transaction
Executes Client's Transaction
Transaction Finished
Transaction done
Recieves Receipt
Show Detailed Information
end

Design principles and Design pattern used

SRP

The SRP is used to ensure that each class has a single responsibility and performs only one task. This principle is applied throughout the project by separating the responsibilities of different modules, such as customer, account, transaction, and employee. Each module has its own set of classes that are responsible for handling the related tasks, such as account creation, transaction processing, and employee management. This approach makes the system easier to maintain and modify because changes made to one module do not affect the others.

OCP

The OCP is used to ensure that the code is open for extension but closed for modification. This principle is applied by using abstraction and interfaces to define the behaviors of the different modules. For example, the account module has an Account interface that defines the methods for managing customer accounts, and different account types (such as savings and checking) implement this interface. This allows the system to be easily extended by adding new account types without modifying the existing code. Additionally, the system uses polymorphism to allow different implementations of the same interface to be used interchangeably.

MVC

The Model-View-Controller (MVC) architectural pattern is used in the project to separate the application's responsibilities into three interconnected components: Model, View, and Controller.

Model: The model component represents the data and business logic of the application. In the Bank Management System, the model includes classes such as Account, Customer, Transaction and Employee, which are responsible for managing the application's data and enforcing the business rules.

View: The view component represents the user interface of the application. In the Bank Management System, the view includes the various screens, forms, and menus that the user interacts with to perform various banking operations.

Controller: The controller component acts as an intermediary between the model and the view. In the Bank Management System, the controller includes the various classes that handle user input, interpret user actions, and update the model and view accordingly. For example, when a user creates a new account, the controller handles the input, updates the model with the new account information, and refreshes the view to display the updated account list.