CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

Most of the bank workers enter people’s account details manually. This manual entering of data will lead to few mistakes therefore we can use this software to avoid those mistake and work seamlessly. Using this software, the data can be stored into the system and can be viewed anytime.

1.2 OBJECTIVES

The main objective of this project is to easily create a new account and do few other operations. This project makes work more and easy to use because of its simplicity. The bank workers have to do a lot work on a daily basis thus this project can make their work much easier and simple without any complication.

1.3 METHODOLOGY TO BE FOLLOWED

This project is used to create a new account like checking account or current account and savings account. This project can also be used to deposit and withdraw certain amount. We can also apply transaction fee for each transaction in the checking account and we can also apply interest for the savings account. In total we have five operations in this project which includes creating a new account (Checking or savings), deposit funds, withdraw funds, apply interest and exit function.

1.4 EXPECTED OUTCOMES

In this project the admin can;

- Can create a new account (Saving or checking account)
- Can deposit funds into any account
- Can withdraw funds from any account
- Can add interest into the savings account

1.5 HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS:

- CPU: Intel or AMD processor with 64-bit support
- Disk Storage: 4 GB of free disk space
- Monitor Resolution: 1280x800; Recommended: 1920x1080
- RAM: 4GB or More; Recommended: 8GB

SOFTWARE REQUIREMENTS:

- OS: Windows 7 or more; Recommended Windows 10
- IDE: Eclipse, Netbeans

CHAPTER 2

FUNDAMENTALS OF JAVA PROGRAMMING

2.1 INTRODUCTION

Also, the creation of Java in itself was deeply rooted in the process of refinement and adaptation occurring in programming languages for the past several decades, driven by the need to solve a fundamental problem that the preceding languages could not solve. Also, the creation of Java in itself was deeply rooted in the process of refinement and adaptation occurring in programming languages for the past several decades, driven by the need to solve a fundamental problem that the preceding languages could not solve. James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan came up with the idea for Java in 1991 while working at Sun Microsystems, Inc. It took 18 months to create the first functional version of the language, which was originally dubbed "Oak" before being renamed "Java" in 1995. More people contributed to the design and development of the language between the initial implementation of Oak in 1992 and the public release of Java in 1995, among whom Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lindholm were significant contributors in maturing the original prototype.

2.2 CLASS

Class is at the core of Java. It can be defined as the logical construct upon which the entire Java language is built. A class defines the shape and nature of an object, and thus, class forms the basis for object-oriented programming in Java. Every idea that will be used in a Java programming must be contained within a class. A user-defined class serves as a template or prototype from which objects are made. We can generate many objects from a class because it represents the collection of attributes or operations shared by all objects of a particular kind. A class is merely a logical construct that takes up no physical or memory resources.

It enables the definition of all the attributes, methods, and APIs that define an object's exterior definition as well as its internal state and behaviour. It also enables us to specify the full syntax for dealing with inheritance, polymorphism, encapsulation, and abstraction. A class is declared by use of the **class** keyword. An example of a class is shown below

```
public class ClassName {
 String name;  int age;
void display () {
//method body;
}
 }
```

The data / variables, defined within a class are called instance variables. The code is contained within methods and collectively, the methods and variables defined within a class are called members of the class. In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access.
2. **class keyword:** class keyword is used to create a class.
3. **Class name:** The name should begin with an initial letter (capitalized by convention).
4. **Superclass (if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces (if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body surrounded by braces, { }

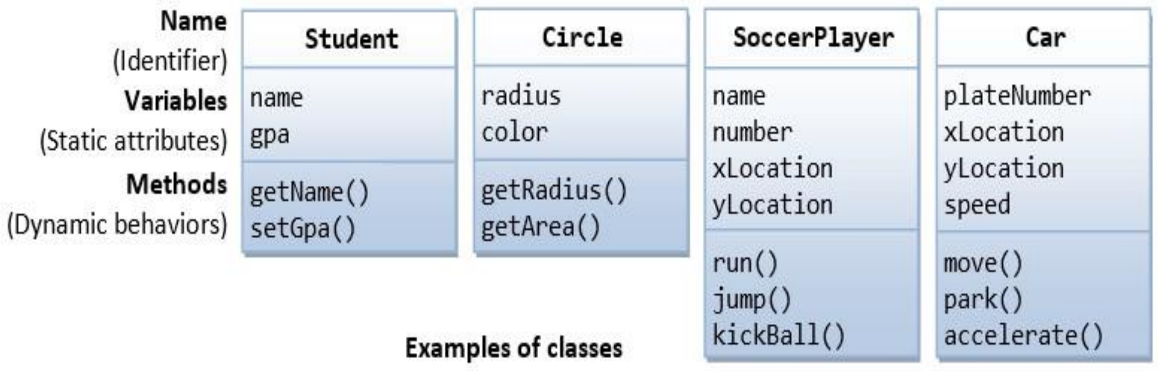


Fig. 2.1.1: Example of classes

2.3 OBJECT

In Java, we actually generate a new data type when we create a class. Then, objects of that type can be declared using this type. However, obtaining objects from a class requires two steps. We must first declare a class type variable. This variable merely serves as a variable that can refer to an object; it does not define an object. The item must then be physically copied and assigned to that variable, which comes second. A new operator can be used to do this. The new operator returns a reference to an object after dynamically allocating memory for it at runtime. The variable then contains this reference. Thus, in Java, all class objects must be dynamically allocated. An object consists of:

State: It is represented by attributes of an object. It also reflects the properties of an object.

Behaviour: It is represented by methods of an object. It also reflects the response of an object with other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

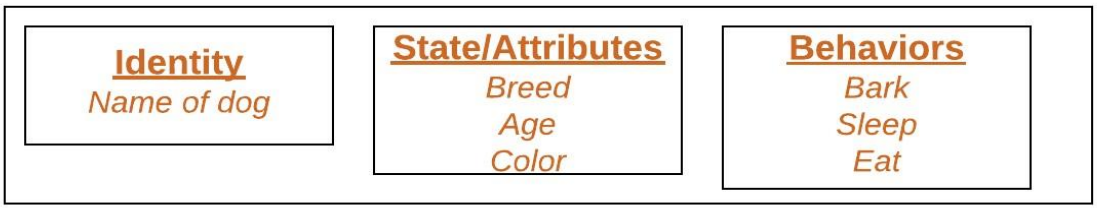


Fig. 2.3.1: Object Attributes

When an object of a class is created, the class is said to be instantiated and all the instances share the attributes and the behaviour of the class. However, the values of those attributes, i.e. the state are unique for each object. A single class may thus have any number of instances.

For example, the following statement is used to declare an object of type Vehicle:

Vehicle myvehicle = new Vehicle();

The new operator dynamically allocates memory for an object. It has this general form:

ClassVar = new classname();

A real-life example shown below can help in better understanding of JAVA Objects & Classes.

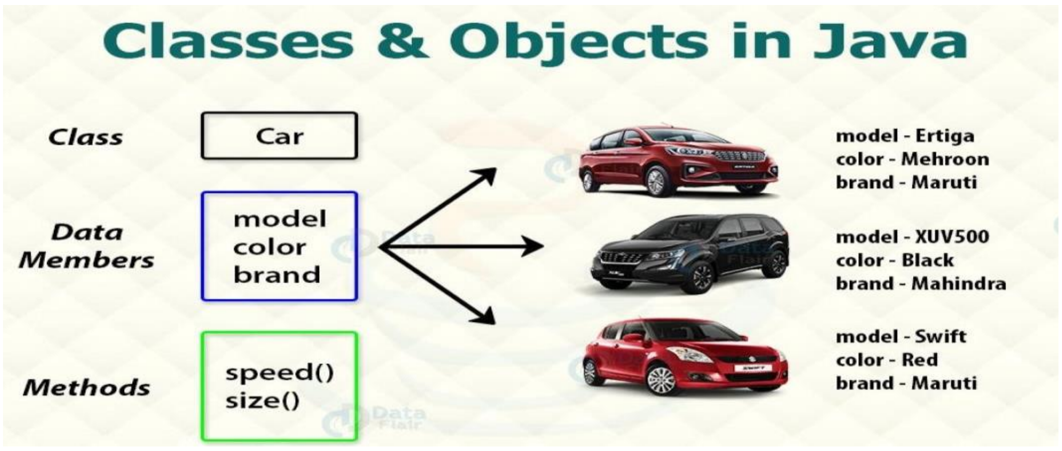


Fig 2.3.2: Classes And Objects

2.4 INHERITANCE

In Java, inheritance is a mechanism that enables one object to obtain/acquire all of another object's properties and behaviours. As it enables the development of hierarchical categories, it is one of the pillars of object-oriented programming. Other, more specialised classes can then inherit this general class and add their own particular characteristics. A class that inherits another class is referred to as a subclass/child class, while the inherited class is referred to as a superclass/parent class. Thus, a subclass is a refined form of a superclass. It adds its own special components and inherits all of the instance variables and methods defined by the superclass.

Terms utilized in Inheritance:

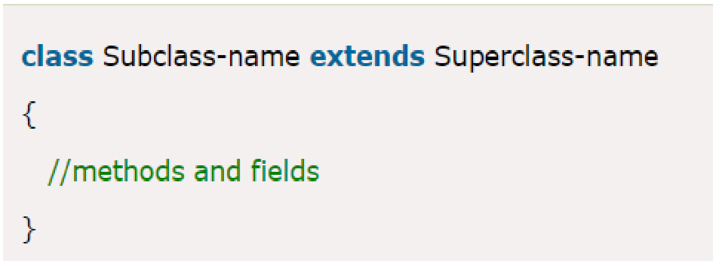
1) Class: A class is a gathering of items which have normal properties.

2) Sub Class: A class which acquires the other class. It is additionally called determined class, broadened class, or kid class.

3) Super Class: Superclass is the class from where a subclass acquires the highlights. Additionally called base class or parent class.

4) Reusability: It is an instrument which works with to reuse the fields and techniques for the current class when you make another class.

Syntax of Java Inheritance:



```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

Fig 2.4.1: Syntax of Inheritance

The extends keyword indicates that you are making a new class that derives from an existing class. On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

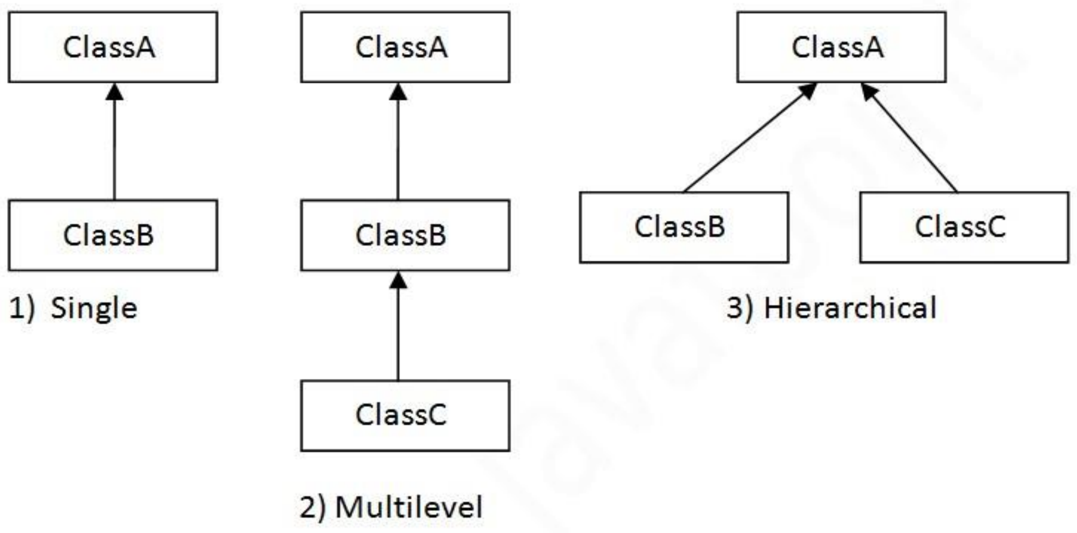


Fig 2.4.2 : Types Of Inheritance

Types of Inheritance:

Single Inheritance: It is the simple inheritance of one class by another.

Multilevel Inheritance: The existence or occurring of a chain of inheritance is known as multilevel inheritance

Hierarchical Inheritance: When multiple classes inherit a single class, the inheritance is called hierarchical inheritance

2.5 POLYMORPHISM

Polymorphism, which means "many forms," is derived from the Greek terms poly, which means "many," and morphs, which means shapes. It is a function that enables the usage of a single interface for a broad category of operations. A child class can incorporate its own functionality while sharing the data and behaviours of its parent class thanks to polymorphism.

In Java polymorphism is mainly divided into two types:

- Compile time Polymorphism:

  Also known as static polymorphism, this type of polymorphism is achieved by method overloading or operator overloading. Java however does not support operator overloading.

  Method Overloading is the existence of multiple functions with same name but different parameters. Such functions are said to be overloaded and can be achieved by a change in number of arguments or/and change in type of arguments.

- Runtime Polymorphism:

  Also known as Dynamic Method Dispatch, it is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

  Method Overriding occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden

METHOD OVERLOADING	METHOD OVERRIDING
Compile time polymorphism	Run time polymorphism
Static binding	Dynamic binding
Implement in a single class	Implement in two class
No need to use inheritance	With inheritance concept we can achieve method overriding
Method name should be same	Method name should be same
Difference in parameter	Same parameter
Return types can be different	Return type should be same
Static methods can be overload	Static method cannot be over ride
Early binding	Late binding

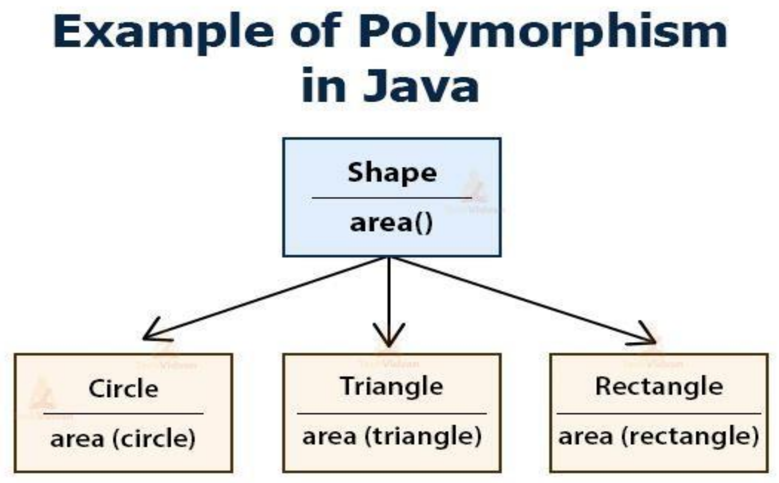


Fig. 2.5.1: **Polymorphism**

2.6 ABSTRACT CLASS

A class which is announced with the theoretical catchphrase is known as a theoretical class in Java. Reflection is a cycle of concealing the execution subtleties and showing just usefulness to the client. Two different ways to accomplish deliberation in java are: Abstract Class and Interface

• An unique class should be proclaimed with a theoretical watchword.

• It can have dynamic and non-unique strategies

• It can't be started up

• It can have constructors and static techniques moreover

• It can have last techniques which will compel the subclass not to change the body of the strategy

Example: abstract void print ();

2.7 JAVA PACKAGES

In Java, similar or related classes, interfaces, and even sub-packages are grouped together in a package. To eliminate name conflicts and provide better, more maintainable code, packages can and are used. They are separated into two groups:

Built-in Packages:

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

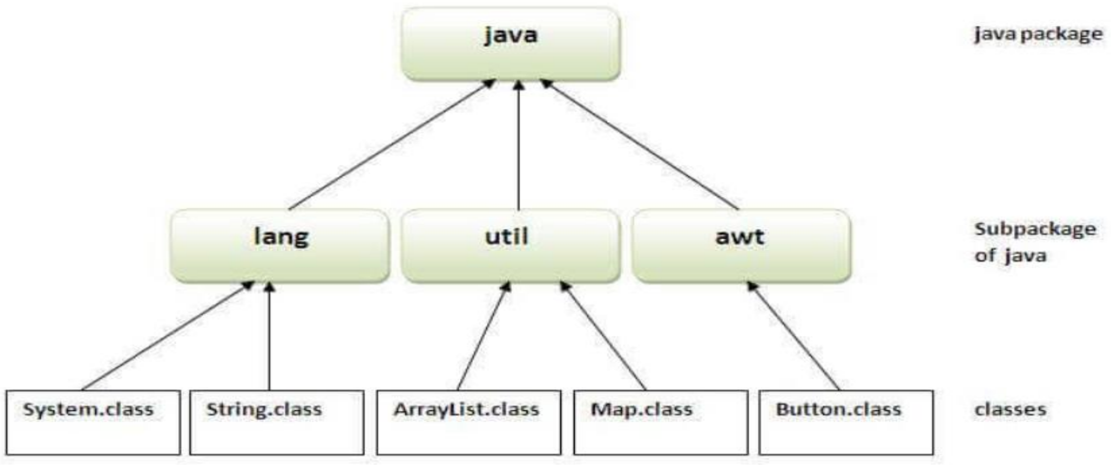


Fig. 2.7.1: Java Package

2.8 EXCEPTION HANDLING

In order to handle runtime faults and keep the application's normal flow, Java uses an exception handling mechanism. An exception in Java is a circumstance that prevents the program's regular flow. It is an object that is hurled during playback.

There are five keywords used for handling exceptions in JAVA –

Try:

The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

Catch:

The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

Finally:

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.

Throw:

The "throw" keyword is used to throw an exception.

Throws:

The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

2.9 THREADS

A thread, with regards to Java, is the way followed when executing a program. All Java Programs have something like one thread, known as the primary thread, which is made by Java Virtual Machine(JVM) at program's beginning, when the principle() technique is summoned with the fundamental thread. In Java, making a thread is refined by carrying out an interface and broadening a class. Each Java Thread is made and constrained by the java.lang.Thread class. A solitary strung application has just one thread and can deal with just each undertaking in turn.

2.10 I/O BASICS

Java I/O (Input and Output) is utilized to handle the info and produce the yield. Java utilizes the idea of a stream to make I/O activity quick. The java.io bundle contains every one of the classes needed for info and yield activities. We can perform document dealing with in Java by Java I/O API. Stream: A stream is a grouping of information. In Java, a stream is made out of bytes.3 streams are created for us automatically. They are:

1. System.out: standard output stream
2. System.in: standard input stream
3. System.err: standard error stream

CHAPTER 3

DESIGN

3.1 DESIGN GOALS

Our design mechanism includes four main things:

1. Stores the new bank accounts.
2. Reduced complexity in accessing the deposit operation.
3. Reduced complexity in accessing the withdraw operation.
4. Can easily apply interest on the savings account.

3.2 ALGORITHM / PSEUDOCODE

```
1. Bank Account Menu
2. If Create new account, select checking account or savings account
   If checking account then enter the details such as name, phone no, aadhaar no, father
      name, mothers name, city, account no and transaction fee
   If savings account then enter the details such as name, phone no, aadhaar no, father
      name, mothers name, city, account no and interest rate
      3.  If deposit funds then enter the account no and amount to be deposited
      4. If withdraw funds then enter the account no and amount to be withdrawn
      5. If Apply Interest then enter the account no to which interest will be applied
```

3.3 FLOWCHART

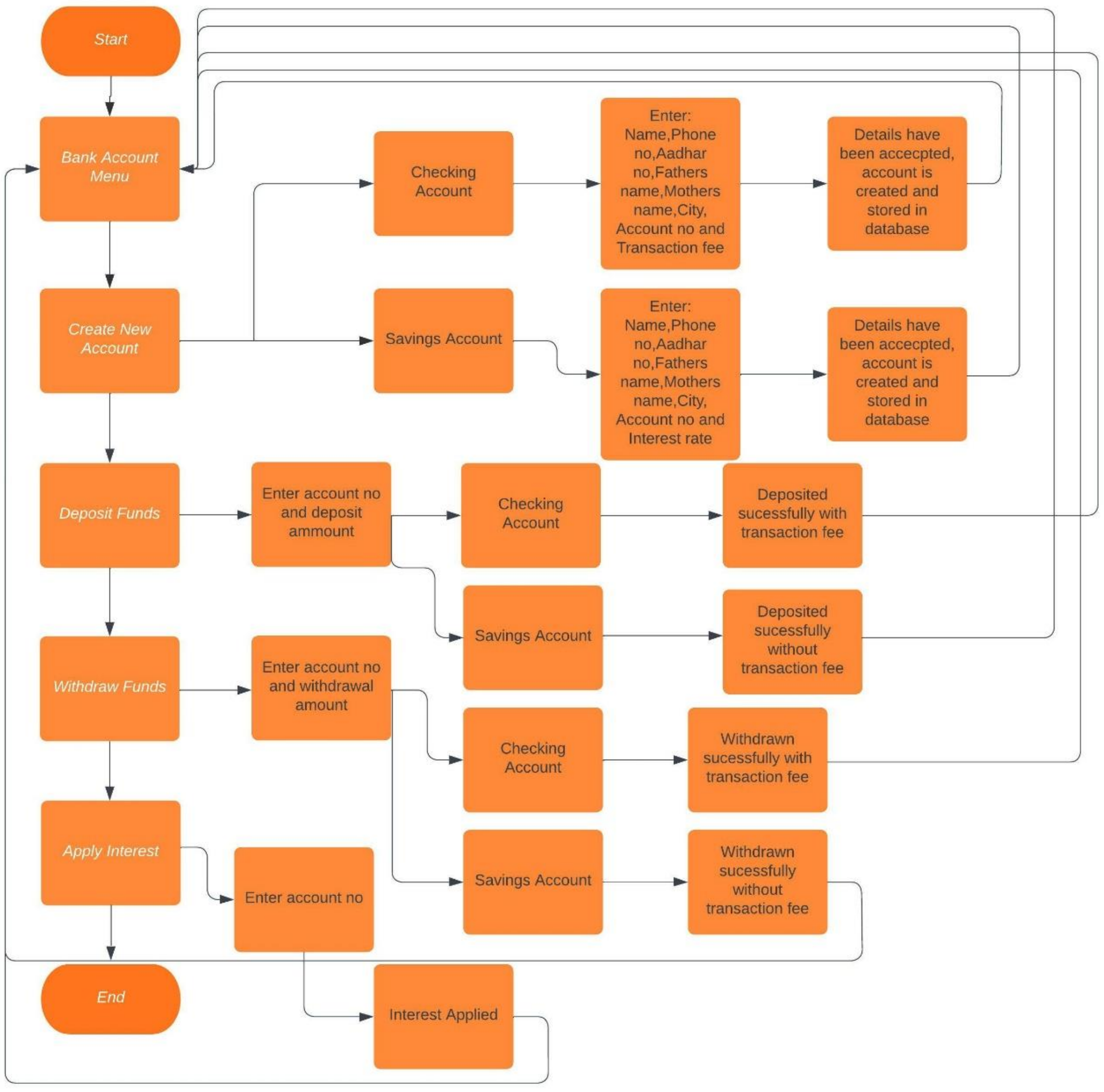

Start
Bank Account Menu
Create New Account
Deposit Funds
Withdraw Funds
Apply Interest
End
Checking Account
Enter: Name,Phone no,Aadhar no,Fathers name,Mothers name,City, Account no and Transaction fee
Details have been accecpted, account is created and stored in database
Savings Account
Enter: Name,Phone no,Aadhar no,Fathers name,Mothers name,City, Account no and Interest rate
Details have been accecpted, account is created and stored in database
Enter account no and deposit ammount
Checking Account
Deposited sucessfully with transaction fee
Savings Account
Deposited sucessfully without transaction fee
Enter account no and withdrawal amount
Checking Account
Withdrawn sucessfully with transaction fee
Savings Account
Withdrawn sucessfully without transaction fee
Enter account no
Interest Applied

Fig 3.3.1: Design flowchart

CHAPTER 4

IMPLEMENTATION

4.1 MODULE 1 FUNCTIONALITY

```
package project;

//Base clase
//Abstract
public abstract class Account {

// account number
private int accountNumber;

// Balance
protected double balance;

// Default constructor
public Account() {

}

public Account(int accountNumber) {
   this.accountNumber = accountNumber;
   balance = 0;
}

// Getter methods
public double getBalance() {
   return this.balance;
}

public int getAccountNumber() {
   return this.accountNumber;
}

// Abstract
/**
* Function to deposit funds into the account as long as
* the amount parameter is > 0
*
* Apply Transaction fee for the CheckingAccount
*
* @param amount value to be deposited
*/
public abstract void deposit(double amount);
/**
* Function to withdraw funds from the Account as long as
*  1. Amount to withdraw must be > 0
*  2. Amount to withdraw must be <= balance
*
* @param amount value to be withdrawn
*/
public abstract void withdraw(double amount);
}
```

This above code is from the class Account, this piece of code creates an abstract class account. Default constructor is created and parameterized constructors are initialized.

4.2 MODULE 2 FUNCTIONALITY

```java
package project;

//Checking Account
public class CheckingAccount extends Account {

// Default Transaction Fee
private static double FEE = 2.5;

// default constructor
public CheckingAccount() {
   super();
}

/**
* Parameter constructor to intialize CheckingAccount
* with a custom Account Number and a Customer Transaction
* Fee.
*/
public CheckingAccount(int accountNumber, double fee) {
   super(accountNumber);
   FEE = fee;
}

/**
* Function to deposit funds into the account as long as the amount parameter is
* > 0
*
* Apply Transaction fee for the CheckingAccount
*
* @param amount value to be deposited
*/
public void deposit(double amount) {

   // First Check amount
   if( amount > 0) {
       balance += amount;
       System.out.printf("Amount %.2f deposited%n", amount);

       // Apply Transaction Fee
       balance -= FEE;
       System.out.printf("Fee %.2f Applied%n", FEE);
       System.out.printf("Current Balance is: %.2f%n", balance);
```

```
    } else {
        System.out.println("A negative amount cannot be deposited");
    }
}

/**
* Function to withdraw funds from the Account as long as 1. Amount to withdraw
* must be > 0 2. Amount to withdraw must be <= balance
*
* @param amount value to be withdrawn
*/
public void withdraw(double amount) {

    // Same check
    if(amount > 0) {
        // Check sufficient balance
        if((amount+FEE) <= balance) {

            System.out.printf("Amount of %.2f withdrawn from Account%n", amount);
            balance -= amount;
            balance -= FEE;
            System.out.printf("Fee of %.2f applied%n", FEE);
            System.out.printf("Current Balance is: %.2f%n", balance);
        }
    } else {
        System.out.println("Negative amount cannot be withdrawn!");
    }
}
}
```

This above code is from the class CheckingAccount, the subclass of Account. It is used to check if the account has sufficient balance and compute the transaction fee for each transaction.

4.3 MODULE 3 FUNCTIONALITY

```
package project;

//Savings Account child class
//has an interest rate
//a method to apply interest - profit

public class SavingAccount extends Account {

// interest rate
private double interestRate;

// default constructor
public SavingAccount() {
   super();
}

/**
* Parameter constructor to intialize Savings account with a customer account
* number and interest rate
*/
public SavingAccount(int accountNumber, double interestRate) {
   super(accountNumber);
   this.interestRate = interestRate;
}

// getter function
public double getInterestRate() {
   return this.interestRate;
}

public void setInterestRate(double interestRate) {
   this.interestRate = interestRate;
}

public double calcInterest() {
   return balance * interestRate /100;
}

public void applyInterest() {
   double interest = calcInterest();
   System.out.printf("Interest amount %.2f added to balance%n", interest);
   deposit(interest);
}

/**
* Function to deposit funds into the account as long as the amount parameter is
* > 0
*
* Apply Transaction fee for the CheckingAccount
*
* @param amount value to be deposited
*/
public void deposit(double amount) {

   // First Check amount
   if (amount > 0) {
       balance += amount;
       System.out.printf("Amount %.2f deposited%n", amount);
       System.out.printf("Current Balance is: %.2f%n", balance);

   } else {
       System.out.println("A negative amount cannot be deposited");
   }
}

/**
* Function to withdraw funds from the Account as long as 1. Amount to withdraw
* must be > 0 2. Amount to withdraw must be <= balance
*
* @param amount value to be withdrawn
*/
public void withdraw(double amount) {

   // Same check
   if (amount > 0) {
       // Check sufficient balance
       if ((amount) <= balance) {
           System.out.printf("Amount of %.2f withdrawn from Account%n", amount);
           balance -= amount;
           System.out.printf("Current Balance is: %.2f%n", balance);
       }
   } else {
       System.out.println("Negative amount cannot be withdrawn!");
   }}}
```

This above code is from the class SavingAccount, the subclass of Account . It is used to compute or calculates the interest for the specified account depending on the balance in that account.

4.1 MODULE 4 FUNCTIONALITY

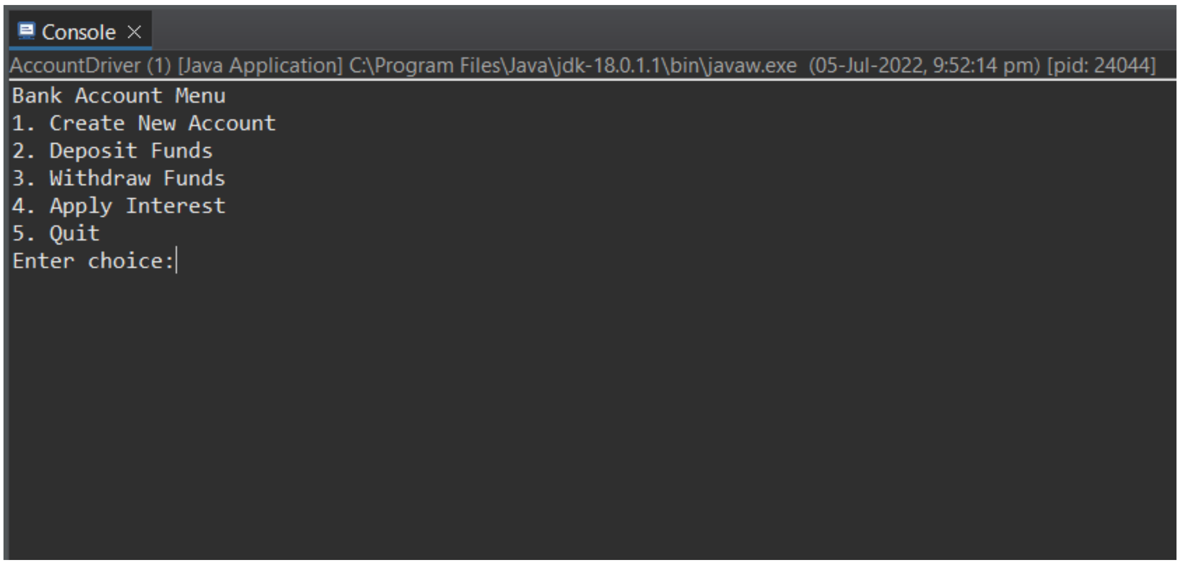
```
package project;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class AccountDriver {

    // Entry point of program
    public static void main(String [] args) {

        Scanner keyboard = new Scanner(System.in);

        // Create array of Accounts
        Account accounts [] = new Account[10];
        int numAccounts = 0;

        int choice;

        do {
            choice = menu(keyboard);
            System.out.println();

            if(choice == 1) {
                accounts[numAccounts++] = createAccount(keyboard);
            } else if(choice == 2) {
                doDeposit(accounts, numAccounts, keyboard);
            } else if(choice == 3) {
                doWithdraw(accounts, numAccounts, keyboard);
            } else if(choice == 4) {
                applyInterest(accounts, numAccounts, keyboard);
            } else {
                System.out.println("GoodBye!");
            }
            System.out.println();
        } while(choice != 5);
    }

    /**
     * Account choice
     *
     * @param keyboard
     * @return choice
     */
    public static int accountMenu(Scanner keyboard) {
        System.out.println("Select Account Type");
        System.out.println("1. Checking Account");
        System.out.println("2. Savings Account");

        int choice;
        do {
            System.out.print("Enter choice: ");
            choice = keyboard.nextInt();
        }while(choice < 1 || choice > 2);

        return choice;
    }

    public static int searchAccount(Account accounts [], int count, int accountNumber) {

        for(int i=0; i<count; i++) {
            if(accounts[i].getAccountNumber() == accountNumber) {
                return i;
            }
        }

        return -1;
    }

    /**
     * Function to perform Deposit on a selected account
     */
    public static void doDeposit(Account accounts [], int count, Scanner keyboard) {
        // Get the account number
        System.out.print("\nPlease enter account number: ");
        int accountNumber = keyboard.nextInt();

        // search for account
        int index = searchAccount(accounts, count, accountNumber);

        if(index >= 0) {
            // Amount
            System.out.print("Please enter Deposit Amount: ");
            double amount = keyboard.nextDouble();

            accounts[index].deposit(amount);
        } else {
            System.out.println("No account exist with AccountNumber: " + accountNumber);
        }
    }

    public static void doWithdraw(Account accounts [], int count, Scanner keyboard) {
        // Get the account number
        System.out.print("\nPlease enter account number: ");
        int accountNumber = keyboard.nextInt();

        // search for account
        int index = searchAccount(accounts, count, accountNumber);

        if(index >= 0) {
            // Amount
            System.out.print("Please enter Withdraw Amount: ");
            double amount = keyboard.nextDouble();
            accounts[index].withdraw(amount);
        } else {
            System.out.println("No account exist with AccountNumber: " + accountNumber);
        }
    }

    public static void applyInterest(Account accounts [], int count, Scanner keyboard) {
        // Get the account number
        System.out.print("\nPlease enter account number: ");
        int accountNumber = keyboard.nextInt();

        // search for account
        int index = searchAccount(accounts, count, accountNumber);

        if(index >= 0) {

            // must be instance of savings account
            if(accounts[index] instanceof SavingAccount) {
                ((SavingAccount)accounts[index]).applyInterest();
            }
        } else {
            System.out.println("No account exist with AccountNumber: " + accountNumber);
        }
    }

    /**
     * Function to create a new Account
     */
    public static Account createAccount(Scanner keyboard) {

        Account account = null;
        int choice = accountMenu(keyboard);

        int accountNumber;
        String phone_no,father_name,mother_name,aadhar_no,address;
        System.out.print("\nEnter your name: ");
        String s = keyboard.next();
        System.out.print("\nEnter your phone number: ");
        phone_no = keyboard.next();
        System.out.print("\nEnter your aadhar number: ");
        aadhar_no = keyboard.next();
        System.out.print("\nEnter your father's name: ");
        father_name = keyboard.next();
        System.out.print("\nEnter your mother's name: ");
        mother_name = keyboard.next();
        System.out.print("\nEnter city: ");
        address = keyboard.next();
        System.out.print("\nEnter Account Number: ");
        accountNumber = keyboard.nextInt();

        if(choice == 1)  { // chekcing account
            System.out.print("Enter Transaction Fee: ");
            double fee = keyboard.nextDouble();
            account = new CheckingAccount(accountNumber, fee);
        } else { // Savings account

            System.out.print("Please enter Interest Rate: ");
            double ir = keyboard.nextDouble();
            account = new SavingAccount(accountNumber, ir);
        }
```

```
            String query = "INSERT INTO account_details values(?,?,?,?,?,?,?)";

            PreparedStatement ps = connection.prepareStatement(query);
            ps.setString(1, s);
            ps.setString(2, phone_no);
            ps.setString(3, aadhar_no);
            ps.setString(4, father_name);
            ps.setString(5, mother_name);
            ps.setString(6, address);
            ps.setString(7, a);
            ps.executeUpdate();
            System.out.println("Your details have been accepted");


        } catch (Exception exception) {
            exception.printStackTrace();
        }




        return account;
    }

    /**
     * Menu to display options and get the user's selection
     *
     * @param keyboard
     * @return choice
     */
    public static int menu(Scanner keyboard) {
        System.out.println("Bank Account Menu");
        System.out.println("1. Create New Account");
        System.out.println("2. Deposit Funds");
        System.out.println("3. Withdraw Funds");
        System.out.println("4. Apply Interest");
        System.out.println("5. Quit");

        int choice;

        do {

            System.out.print("Enter choice: ");
            choice = keyboard.nextInt();
        } while(choice < 1 || choice > 5);

        return choice;
    }     }
```

The above is the code from the class AccountDriver, it provides Menu driven user interaction. User can select an account, deposit, withdraw funds. Apply interest rate on Savings account

CHAPTER 5

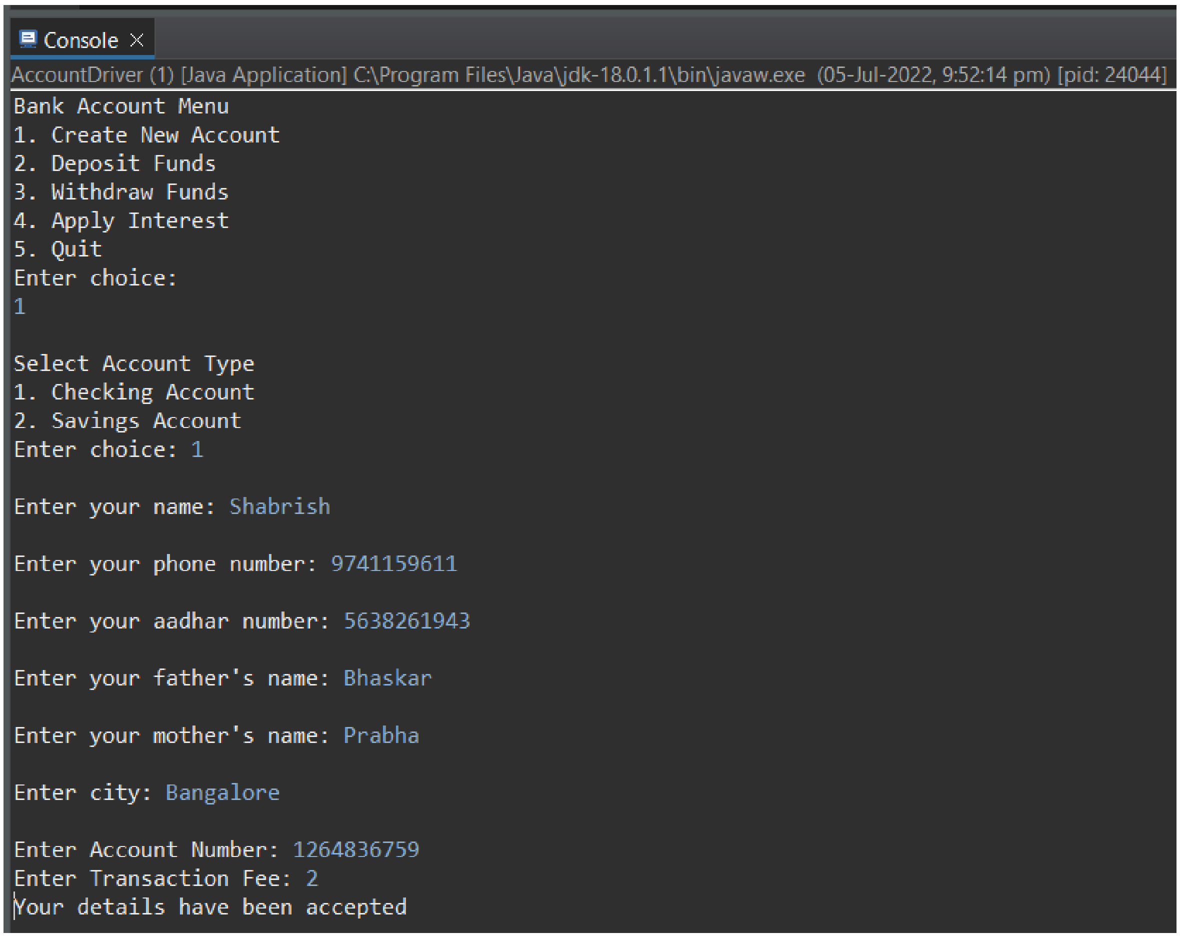
RESULTS



```
Console ×
AccountDriver (1) [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe  (05-Jul-2022, 9:52:14 pm) [pid: 24044]
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice:
```

Fig 5.1 Bank Account Menu

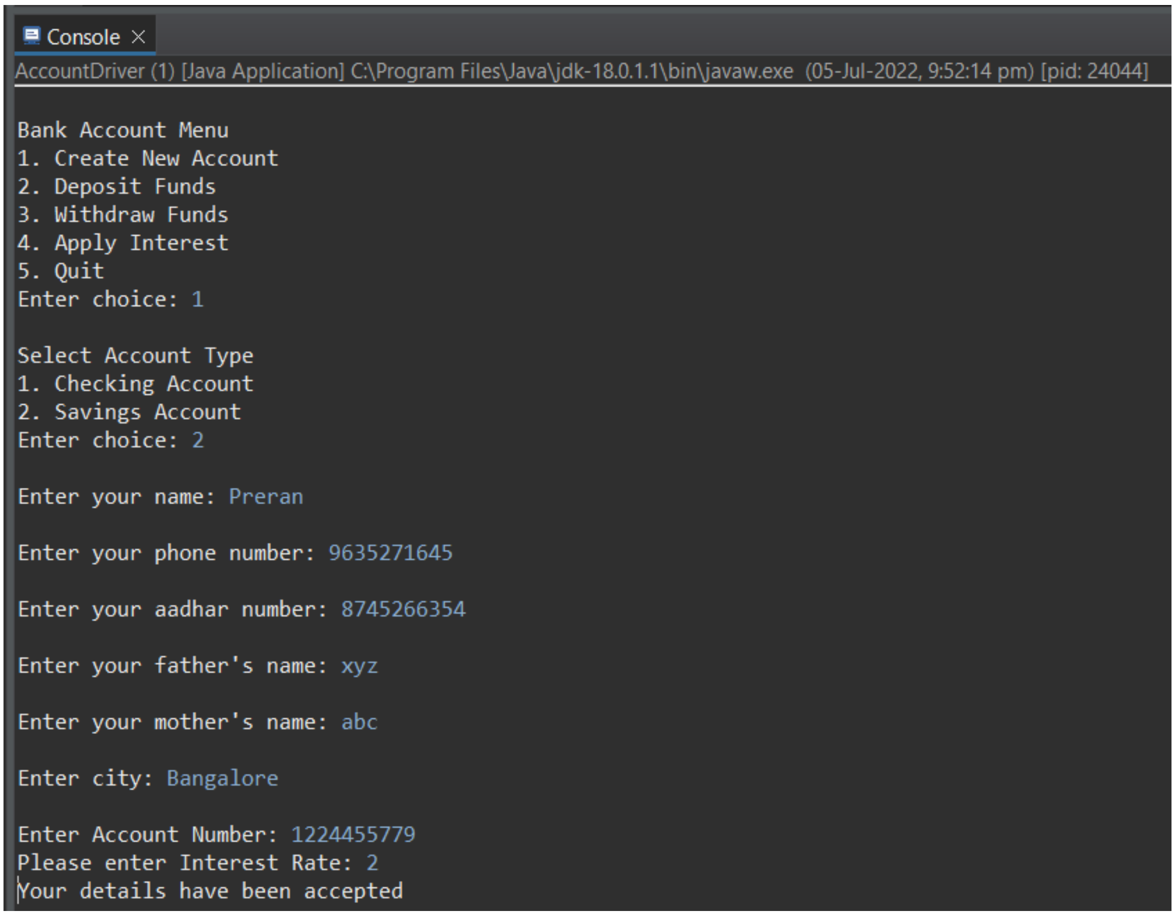
The above screenshot shows the bank account menu. We have 5 operations.



```
Console ×
AccountDriver (1) [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe  (05-Jul-2022, 9:52:14 pm) [pid: 24044]
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice:
1

Select Account Type
1. Checking Account
2. Savings Account
Enter choice: 1

Enter your name: Shabrish

Enter your phone number: 9741159611

Enter your aadhar number: 5638261943

Enter your father's name: Bhaskar

Enter your mother's name: Prabha

Enter city: Bangalore

Enter Account Number: 1264836759
Enter Transaction Fee: 2
Your details have been accepted
```

Fig 5.2 Creating Checking account

The above screenshot shows the process of creation of a new checking account which gets stored in the database. It takes in details like name, phone number, aadhaar number, father's number, mother's name, city, account number and transaction fee.



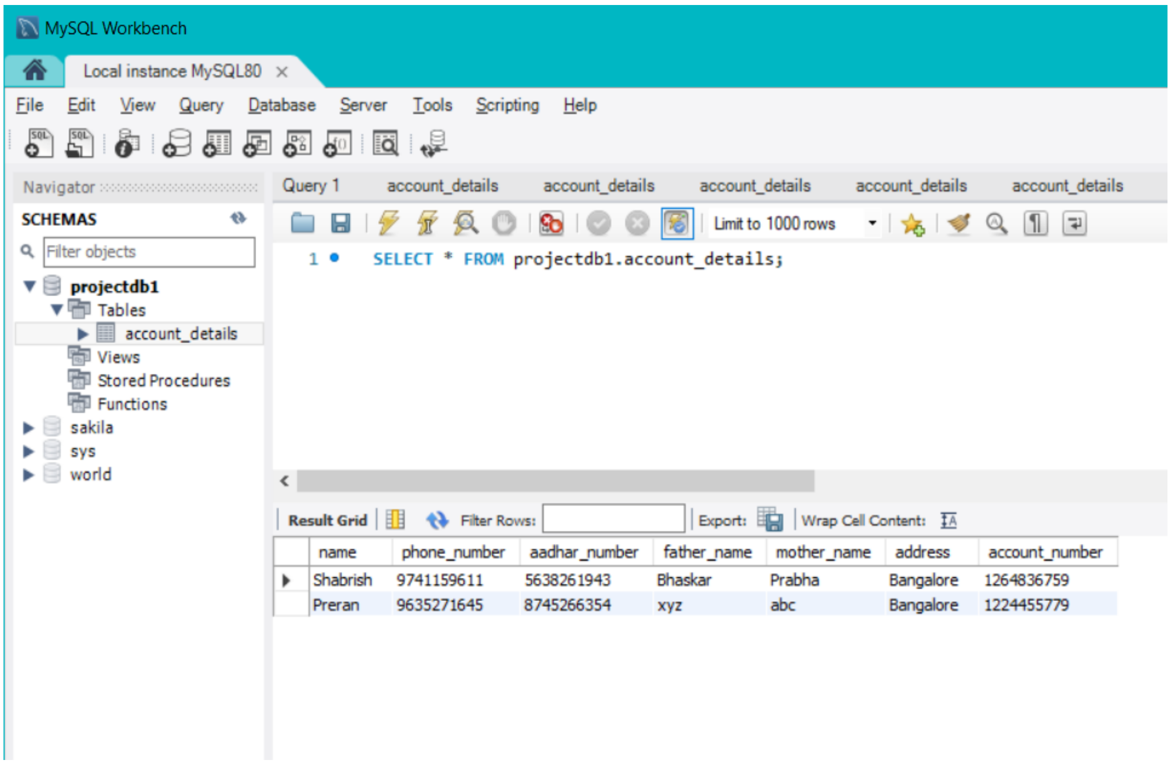
Console ×

AccountDriver (1) [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (05-Jul-2022, 9:52:14 pm) [pid: 24044]

```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 1

Select Account Type
1. Checking Account
2. Savings Account
Enter choice: 2

Enter your name: Preran

Enter your phone number: 9635271645

Enter your aadhar number: 8745266354

Enter your father's name: xyz

Enter your mother's name: abc

Enter city: Bangalore

Enter Account Number: 1224455779
Please enter Interest Rate: 2
Your details have been accepted
```

Fig 5.3 Creating Savings account

The above screenshot shows the process of creation of a new savings account which gets stored in the database. It takes in details like name, phone number, aadhaar number, father's number, mother's name, city, account number and Interest rate.

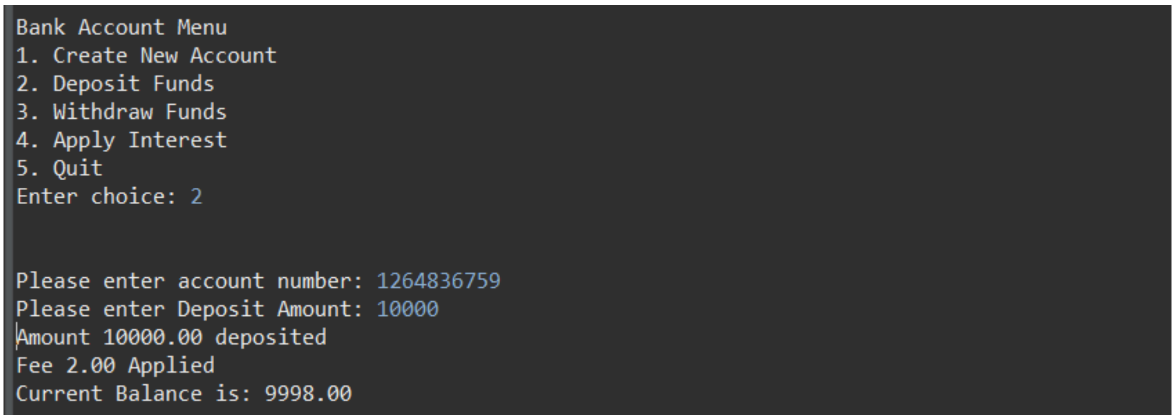


```
SELECT * FROM projectdb1.account_details;
```

name	phone_number	aadhar_number	father_name	mother_name	address	account_number
Shabrish	9741159611	5638261943	Bhaskar	Prabha	Bangalore	1264836759
Preran	9635271645	8745266354	xyz	abc	Bangalore	1224455779

Fig 5.4 Details stored in database

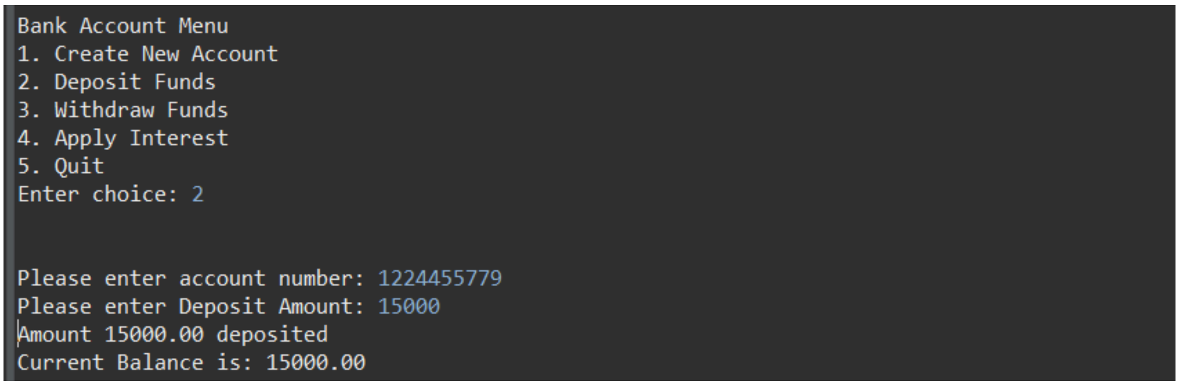
The above screenshot shows the data entries in the MySQL database.



```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 2


Please enter account number: 1264836759
Please enter Deposit Amount: 10000
Amount 10000.00 deposited
Fee 2.00 Applied
Current Balance is: 9998.00
```

Fig 5.5 Deposit operation (Checking Account)

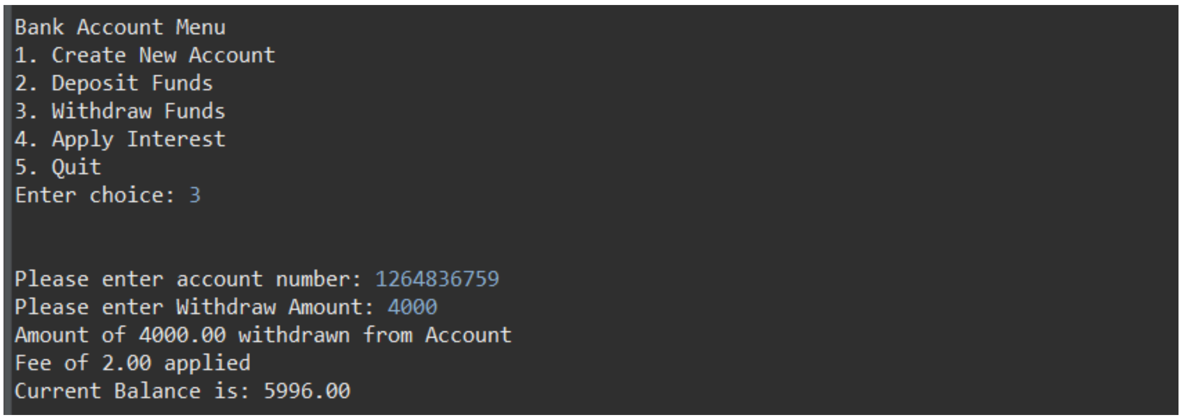
The above screenshot shows the process of depositing funds into a checking bank account with the specified account number. In this there is a transaction fee for every transaction.



```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 2


Please enter account number: 1224455779
Please enter Deposit Amount: 15000
Amount 15000.00 deposited
Current Balance is: 15000.00
```

Fig 5.6 Deposit operation (Saving Account)

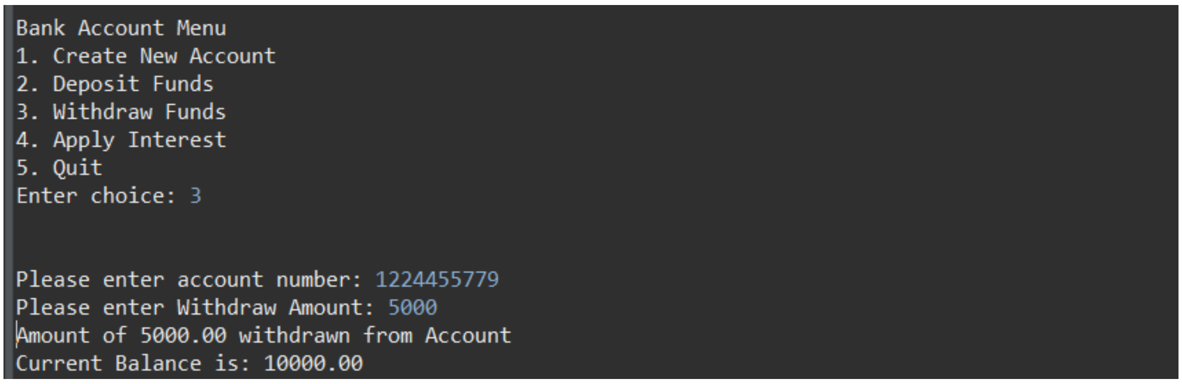
The above screenshot shows the process of depositing funds into a savings bank account with the specified account number. In this there is no transaction fee.



```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 3


Please enter account number: 1264836759
Please enter Withdraw Amount: 4000
Amount of 4000.00 withdrawn from Account
Fee of 2.00 applied
Current Balance is: 5996.00
```

Fig 5.7 Withdraw operation (Checking Account)

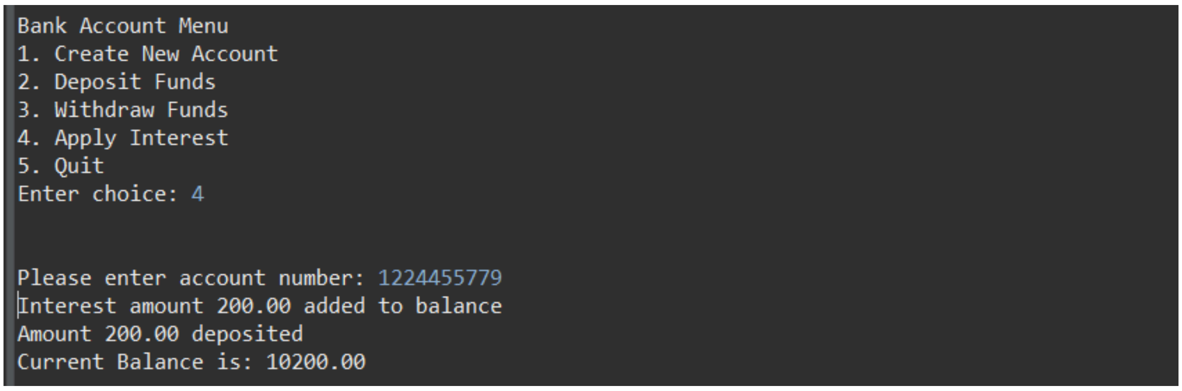
The above screenshot shows the process of withdrawing funds from a checking bank account with the specified account number. There is a transaction fee cut in this.



```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 3


Please enter account number: 1224455779
Please enter Withdraw Amount: 5000
Amount of 5000.00 withdrawn from Account
Current Balance is: 10000.00
```

Fig 5.8 Withdraw operation (Savings Account)

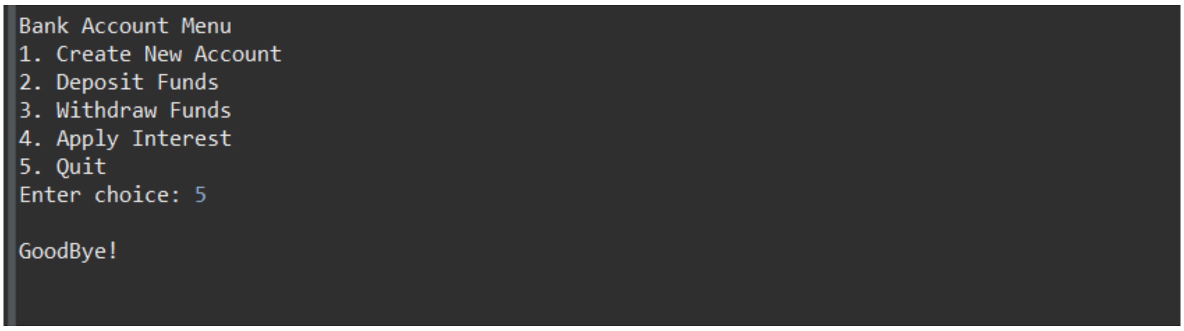
The above screenshot shows the process of withdrawing funds from a savings bank account with the specified account number. There is no transaction fee cut in this.



```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 4


Please enter account number: 1224455779
Interest amount 200.00 added to balance
Amount 200.00 deposited
Current Balance is: 10200.00
```

Fig 5.9 Apply Interest Operation

The above screenshot shows the process of applying interest to the specified savings account.



```
Bank Account Menu
1. Create New Account
2. Deposit Funds
3. Withdraw Funds
4. Apply Interest
5. Quit
Enter choice: 5

GoodBye!
```

Fig 5.10 Quit Operation

The above screenshot shows the process of exiting from the application.

CHAPTER 6

CONCLUSION

The mini project has successfully accomplished the goals it had set out in the objectives and design sections of this report. It is successfully taking in all the details of the costumer and creating a new account and storing the details of that person in the database. Those details can be accessed anytime. We are able to deposit the required funds from any account which is created and stored in the database. We can also withdraw the required funds from any account. We can also work with yearly interest for the savings account. Therefore this project is working successfully and can be used by the officials.

REFERENCES

[1] https://www.javatpoint.com/

[2] https://www.geeksforgeeks.org/

[3] https://www.guru99.com/java-oops-concept.html