

**(Affiliated to Tribhuvan University)**

**Advanced Java Programming**

**Lab 003**

**Object Oriented Programming in Java**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Submitted by:**

Abhinna Ojha, 20788/075

BSc. CSIT - VII

**Submitted to:**

Mr. Krishna Pandey

Department of CSIT

# Write classes to hold Account, SB-Account and Current-Account details. [Here, implement the concept of inheritance.]

# The common properties of the account are Account number, name and amount.

# Specifics of SB account is 4% interest to be paid per month.

# Implement the run-time polymorphism by creating base class variable and derived class object.

# Ask the user for which type of account to be created then create the corresponding account (Note: Use scanner class).

# Implement function overriding by having deposit and withdraw functions and perform the required action accordingly.

# Ensure base class can’t be instantiated. (Note: Abstract keyword can be used).

# Define the minimum balance for the both the type of accounts. Use final keyword to create constants.

# Ensure sb account class and current account class will cannot be used as base classes (Note:You can use final classes).

/\*

\* Title:

\* 1. Write classes to hold Account, SB-Account and Current-Account details. [Here implement

\* the concept of inheritance.]

\* The common properties of the account are Account number, name and amount.

\* Specifics of SB account is 4% interest to be paid per month.

\*

\* ð Implement the run-time polymorphism by creating base class variable and derived class object.

\* ð Ask the user for which type of account to be created then create the corresponding account

\* (Note: Use scanner class).

\* ð Implement function overriding by having deposit and withdraw functions and perform the

\* required action accordingly.

\*

\* Ensure base class can’t be instantiated. (Note: Abstract keyword can be used).

\* 2. Define the minimum balance for the both the type of accounts. Use final keyword to

\* create constants.

\* Ensure sb account class and current account class will cannot be used as base classes

\* (Note:You can use final classes).

\*

\*

\* Date modified; Author(s); Modification details

\* 2022-12-19; abhinna; Created the program

\* 2022-12-22; abhinna; Added the classes, finals, and abstract methods

\* 2022-12-23; abhinna; Implemented deposit and withdraw methods through override

\* \*/

import java.util.Scanner;

public class Main

{

public static void main(String[] args)

{

int accountType = 0;

boolean infiniteFlag = true;

Account account = null;

// infiniteFlag for allowance of only 3 choices

while (infiniteFlag)

{

System.out.println("Enter the account type to open new account");

System.out.println("1. SB Account");

System.out.println("2. Current Account");

System.out.println("0. Exit");

System.out.println("Choice: ");

Scanner scanner = new Scanner(System.in);

accountType = Integer.parseInt(scanner.nextLine());

// when 0 is hit, terminate

if (accountType == 0)

{

infiniteFlag = false;

}

else

{

String x = "";

switch (accountType) {

// when 1 is hit, create SB account

case 1 ->

{

x = "SB";

account = new SBAccount();

}

// when 2 is hit, create current account

case 2 ->

{

x = "Current";

account = new CurrentAccount();

}

}

System.out.print("Creating " + x + " Account");

System.out.print("\nEnter your name: ");

account.name = scanner.nextLine();

System.out.print("Minimum balance to open " + x + " Account is " + account.getMinimumBalance());

System.out.print("\nEnter your balance: ");

double amt = Double.parseDouble(scanner.nextLine());

if(account.amount + amt < account.getMinimumBalance())

{

System.out.println("Minimum balance to open " + x + " Account is " + account.getMinimumBalance());

}

else

{

account.amount += amt;

System.out.println( x + " account created");

infiniteFlag = false;

}

}

}

Scanner scanner = new Scanner(System.in);

infiniteFlag = true;

while (infiniteFlag)

{

System.out.println("Enter the choice");

System.out.println("1. Deposit");

System.out.println("2. Withdraw");

System.out.println("0. Exit");

System.out.println("Choice: ");

int choice = Integer.parseInt(scanner.nextLine());

switch (choice)

{

case 1-> account.deposit();

case 2-> account.withdraw();

case 0-> infiniteFlag = false;

default -> System.out.println("Invalid choice");

}

}

}

}

// base class to be inherited

// base class made abstract so that it cannot be instantiated

abstract class Account

{

int accountNumber;

String name;

double amount;

// constructor to initialise amount = 0

Account()

{

amount = 0;

}

// abstract methods so that it can be overridden later

public abstract void deposit();

public abstract void withdraw();

public abstract double getMinimumBalance();

}

// derived classes made final so they cannot be inherited

final class SBAccount extends Account

{

double interest;

final double minimumBalance = 10000;

// constructor to instantiate 4% interest per month and minimum balance

SBAccount()

{

interest = 4.0;

}

// overriding abstract deposit()

@Override

public void deposit()

{

// using try catch as there may be runtime error where double value may not be inputted

try

{

System.out.println("Enter amount to deposit to your SB account: ");

Scanner scanner = new Scanner(System.in);

double amt = Double.parseDouble(scanner.nextLine());

amount += amt;

System.out.println("Deposit successful, your new balance is " + amount);

}

catch (Exception e)

{

System.out.println("Deposit unsuccessful, some error occurred, please try again");

}

}

// overriding abstract withdraw()

@Override

public void withdraw()

{

// using try catch as there may be runtime error where double value may not be inputted

try

{

System.out.println("Enter amount to withdraw from your SB account: ");

Scanner scanner = new Scanner(System.in);

double amt = Double.parseDouble(scanner.nextLine());

// minimumBalance is 10000, and thus the minimum amount of withdrawal is amount - minimumBalance

if (amt > (amount - getMinimumBalance()))

{

System.out.println("Withdraw unsuccessful, insufficient balance");

}

else

{

amount -= amt;

System.out.println("Withdraw successful, your new balance is " + amount);

}

}

catch (Exception e)

{

System.out.println("Withdraw unsuccessful, some error occurred, please try again");

}

}

// method to get the value of constant minimumBalance

@Override

public double getMinimumBalance()

{

return minimumBalance;

}

}

final class CurrentAccount extends Account

{

final double minimumBalance = 7000;

// overriding abstract deposit()

@Override

public void deposit()

{

// using try catch as there may be runtime error where double value may not be inputted

try

{

boolean flag = true;

while (flag)

{

System.out.println("Enter amount to deposit to your Current account (min 1000): ");

Scanner scanner = new Scanner(System.in);

double amt = Double.parseDouble(scanner.nextLine());

if (amt < 1000)

{

System.out.println("Please deposit at least 1000");

}

else

{

amount += amt;

System.out.println("Deposit successful, your new balance is " + amount);

flag = false;

}

}

}

catch (Exception e)

{

System.out.println("Deposit unsuccessful, some error occurred, please try again");

}

}

// overriding abstract withdraw()

@Override

public void withdraw()

{

// using try catch as there may be runtime error where double value may not be inputted

try

{

System.out.println("Enter amount to withdraw from your Current account, minimum 1000: ");

Scanner scanner = new Scanner(System.in);

double amt = Double.parseDouble(scanner.nextLine());

if (amt < 1000)

{

System.out.println("Minimum withdrawable amount is 1000");

}

else

{

// minimumBalance is 7000, and thus the minimum amount of withdrawal is amount - minimumBalance

if (amt > (amount - getMinimumBalance()))

{

System.out.println("Withdraw unsuccessful, insufficient balance");

}

else

{

amount -= amt;

System.out.println("Withdraw successful, your new balance is " + amount);

}

}

}

catch (Exception e)

{

System.out.println("Withdraw unsuccessful, some error occurred, please try again");

}

}

// method to get the value of constant minimumBalance

@Override

public double getMinimumBalance()

{

return minimumBalance;

}

}















