
| | |
|---|---|
| **Name: Collet Kanyera** | **Registration Number: H240219F** |
| **Name: Kumbirai Dunira** | **Registration Number: H240165R** |
| **Part** | **2.1** |
| **Course Code** | **ICS2101** |
| **Course Name** | **Object Oriented Programming** |
| **Department** | **Computer Science** |
| **Assignment** | **Practical Assigment 1** |
| | |

**Write a java program to solve the banking problem. Create an abstract class Bank that declare account**

**name and balance as state, abstract methods deposit, withdraw and getBalance as behavior. Create a**

**class Account that extends the abstract Bank.**

**i. withdraw(), withdraws money from an Account. Ensure that the withdrawal method does not**

**exceed the Account's balance. If it does, the balance should be left unchanged and the method**

**should print a message indicating “Withdrawal amount exceeded account balance”.**

**ii. deposit() adds only valid (amount greater than 0) amount to the balance**

**iii. getBalance() returns the current balance.**

**iv. All bank transactions should be recorded in a file named Bank.txt located in the root director**

**Create another class AccountTest to test the withdraw, deposit and getBalance methods. [25]**

```java
package bank_system;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * Abstract class Bank:
 * --------------------
 * - Acts as a blueprint for all types of bank accounts.
 * - Holds common "state" (data: account name, balance).
 * - Declares abstract methods (deposit, withdraw, getBalance)
 *   which subclasses must implement.
 */
abstract class Bank {
    // Fields (State of the account)
    protected String accountName;  // Stores the account holder's name
    protected double balance;      // Stores the account balance

    // Constructor to initialize state when an object is created
    public Bank(String accountName, double balance) {
        this.accountName = accountName;
        this.balance = balance;
    }

    // Abstract methods (Behaviors)
    // These must be implemented in subclasses
    public abstract void deposit(double amount);   // Add money to account
    public abstract void withdraw(double amount);  // Take money out of account
    public abstract double getBalance();           // Check balance
}

/**
 * Account class:
```

```java
 * ------------------
 * - Extends (inherits from) Bank.
 * - Provides actual implementation of deposit, withdraw, and getBalance.
 * - Adds functionality to log transactions into a file (Bank.txt).
 */
class Account extends Bank {

    // Constructor: calls the parent (Bank) constructor
    public Account(String accountName, double balance) {
        super(accountName, balance);
    }

    /**
     * Deposit Method:
     * -----------------
     * - Only accepts positive amounts.
     * - Increases the balance by the deposit amount.
     * - Logs the successful deposit into Bank.txt.
     * - If deposit is invalid (negative or zero), prints error.
     */
    @Override
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount; // Increase balance
            logTransaction("Deposited: " + amount + " | Balance: " + balance);
        } else {
            System.out.println("Invalid deposit amount!");
        }
    }

    /**
     * Withdraw Method:
     * ------------------
     * - Allows withdrawal only if:
     *      1. The amount is positive
     *      2. The amount is <= balance
     * - Deducts the withdrawal amount from balance if successful.
     * - Logs both successful and failed attempts in Bank.txt.
     */
    @Override
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount; // Deduct amount
            logTransaction("Withdrew: " + amount + " | Balance: " + balance);
        } else {
            System.out.println("Withdrawal amount exceeded account balance");
            logTransaction("Failed Withdrawal Attempt: " + amount + " | Balance
unchanged: " + balance);
        }
    }

    /**
     * getBalance Method:
     * -------------------
     * - Returns the current balance of the account.
     */
    @Override
    public double getBalance() {
        return balance;
```

```java
    }

    /**
     * logTransaction Method:
     * ----------------------
     * - Private helper method.
     * - Appends transaction details into a text file named "Bank.txt".
     * - Each line has the account holder's name and the transaction performed.
     */
    private void logTransaction(String message) {
        try (FileWriter fw = new FileWriter("Bank.txt", true);
             PrintWriter pw = new PrintWriter(fw)) {
            pw.println(accountName + ": " + message);
        } catch (IOException e) {
            System.out.println("Error writing to file: " + e.getMessage());
        }
    }
}

/**
 * AccountTest Class:
 * ------------------
 * - Contains the main() method.
 * - Used to test the banking functionality by:
 *      1. Creating an Account object.
 *      2. Performing deposits and withdrawals.
 *      3. Displaying final account balance.
 */
public class AccountTest {
    public static void main(String[] args) {
        // Create an account with initial balance of 1000
        Account acc1 = new Account("Alice Moyo", 1000);

        // Test deposit (Valid case)
        acc1.deposit(500);   // Adds 500, balance becomes 1500

        // Test deposit (Invalid case)
        acc1.deposit(-200);  // Invalid, ignored

        // Test withdraw (Valid case)
        acc1.withdraw(700);  // Deducts 700, balance becomes 800

        // Test withdraw (Invalid case - too much)
        acc1.withdraw(2000); // Fails, balance stays 800

        // Display final balance
        System.out.println("Final Balance: " + acc1.getBalance());
    }
}
```

**QUESTION 2**

**a. Write a java program using a stream to filter the numbers that are divisible by 5 from the following ArayList and print them out. [10]**

**(1, 4, 5, 20, 30, 6)**

```
package stream_filter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class DivisibleByFive {
    public static void main(String[] args) {

        // Create a list of numbers (ArrayList) using Arrays.asList()
        List<Integer> numbers = new ArrayList<>(Arrays.asList(1, 4, 5, 20, 30,
6));

        // Use Java Streams to filter numbers divisible by 5 and print them
        numbers.stream() // Convert the list into a stream (like a pipeline for
data)
                .filter(n -> n % 5 == 0) // Keep only numbers divisible by 5
                .forEach(System.out::println); // Print each number that passed the
filter
    }
}
```

**b. Write a java program creating your own custom exception. Throw and catch the exception displaying proper message to the user [15]**

```
// Custom exceptions extend Exception (checked exception)
// or RuntimeException (unchecked exception).
package customer_exception;

class InvalidAgeException extends Exception {
    // Constructor that takes a message to describe the error
    public InvalidAgeException(String message) {
        super(message); // Call the parent Exception constructor
    }
}

public class Custom_Exception{
    // Method to check if a person is old enough to vote
    public static void checkAge(int age) throws InvalidAgeException {
        if (age < 18) {
            // If age < 18, throw our custom exception
            throw new InvalidAgeException("Age " + age + " is too young to vote.
Minimum age is 18.");
        } else {
            System.out.println("Age " + age + " is valid. You can vote!");
        }
```

```
    }

    public static void main(String[] args) {
        try {
            // Step 2: Call method with invalid age to trigger exception
            checkAge(14);

            // Step 3: Call method with valid age (this won't throw exception)
            checkAge(25);

        } catch (InvalidAgeException e) {
            // Step 4: Catch the custom exception and display message
            System.out.println("Custom Exception Caught: " + e.getMessage());
        }
    }
}
```