***CASE STUDY NO. 4***

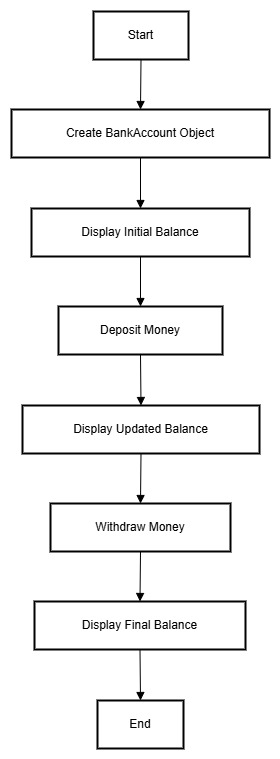
* **Title**
* A bank account system
* **Theory Concepts Of Java/OOP Used in This Case Study**

1. **Class**:   
   A class is a blueprint that defines the properties (fields) and behaviors (methods) of objects. It encapsulates data and functionality into a single unit. In this program, the BankAccount class models a bank account with fields such as accountHolder and balance, and methods like deposit, withdraw, and displayBalance to manage account operations.
2. **Object**:   
   An object is a specific instance of a class that contains real data. It is created using the new keyword and represents an entity of the class in memory. For example, the account object in this program is an instance of the BankAccount class and holds information specific to a particular bank account.
3. **Encapsulation**:   
   Encapsulation is the practice of hiding the internal state of an object and exposing only what is necessary through methods. It improves security and simplifies interaction with an object. In this program, accountHolder and balance are private fields accessed through methods like deposit and withdraw to ensure controlled and valid modifications.
4. **Constructor:**   
   A constructor is a special method used to initialize an object's state when it is created. It has the same name as the class and does not have a return type. In the BankAccount class, the constructor initializes the accountHolder and balance fields, setting up the account's initial state.
5. **Method:**   
   A method is a function defined inside a class that specifies the behavior of objects. Methods operate on the object's fields and can take input parameters. In this program, deposit adds funds to the account, withdraw deducts funds, and displayBalance shows the current balance.
6. **Conditionals:**Conditionals are control structures that execute different blocks of code based on specified conditions. They enable decision-making in a program. For example, the if statements in the deposit and withdraw methods check whether the transactions are valid, such as ensuring that the withdrawal amount does not exceed the account balance.

* **Algorithm:**

1. **Start**
2. **Define the BankAccount class**:
   * Declare private fields accountHolder (String) and balance (double).
   * Create a constructor to initialize accountHolder and balance with provided values.
   * Define the deposit method:
     + If amount > 0, add amount to balance.
   * Define the withdraw method:
     + If amount > 0 and amount <= balance, subtract amount from balance.
   * Define the displayBalance method:
     + Print the accountHolder and current balance.
3. **Define the BankSystem class**:
   * In the main method:
     + Create a BankAccount object account with "John Doe" as accountHolder and 500.00 as the initial balance.
     + Call displayBalance to print the initial balance.
     + Call the deposit method with 200.00 and then displayBalance to show the updated balance.
     + Call the withdraw method with 100.00 and then displayBalance to show the updated balance.
4. **End**

* **Flowchart:**



* **Program/Code:**
* class BankAccount

{

    private String accountHolder;

    private double balance;

    public BankAccount(String accountHolder, double initialBalance)

    {

        this.accountHolder = accountHolder;

        this.balance = initialBalance;

    }

    public void deposit(double amount)

    {

        if (amount > 0)

        {

            balance += amount;

        }

    }

    public void withdraw(double amount)

    {

        if (amount > 0 && amount <= balance) {

            balance -= amount;

        }

    }

    public void displayBalance()

    {

        System.out.println("Account Holder: " + accountHolder + ", Balance: $" + balance);

    }

}

public class BankSystem

{

    public static void main(String[] args)

    {

        BankAccount account = new BankAccount("John Doe", 500.00);

        account.displayBalance();

        account.deposit(200.00);

        account.displayBalance();

        account.withdraw(100.00);

        account.displayBalance();

    }

}

* **Output:**
* Account Holder: John Doe, Balance: $500.0

Account Holder: John Doe, Balance: $700.0

Account Holder: John Doe, Balance: $600.0

* **Conclusion**
* This Bank Account System demonstrates a simple, real-world application of Object-Oriented Programming in Java. Using **classes** and **objects**, the program models essential banking functionalities like deposits, withdrawals, and balance inquiries. By leveraging **encapsulation**, **constructors**, and **methods**, the system ensures a modular and maintainable design. The integration of conditionals guarantees valid transactions, enhancing program reliability.