Object Oriented Programming Lecture

IS-202 D-EN

Object Oriented Programming Final Exam

Erikson Tandanu 00000109972

Hans Kristian 00000077457

Matthew Sebastian 00000091095

San Cornelius Carlostio 00000077518

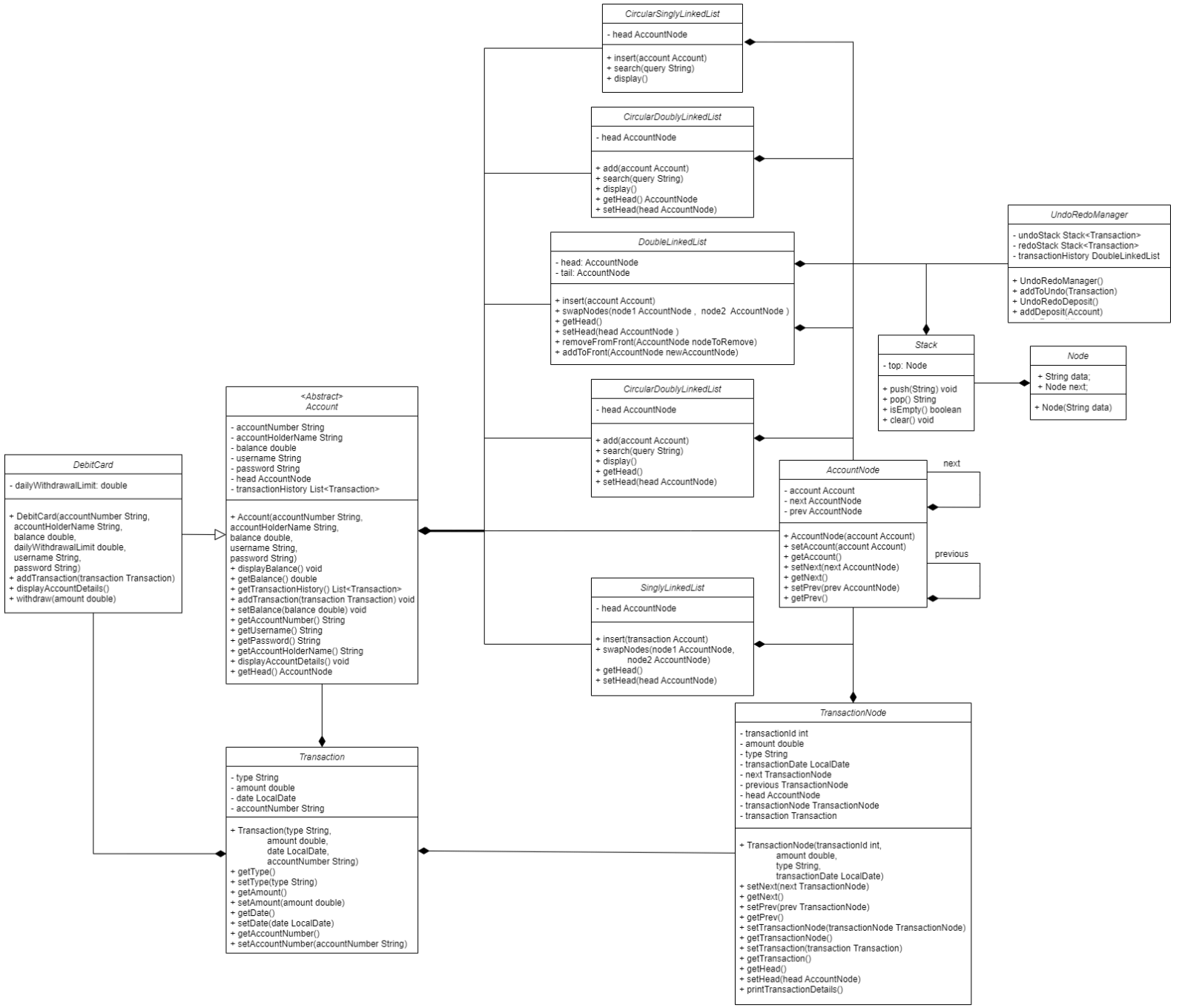
SISTEM INFORMASI

UNIVERSITAS MULTIMEDIA NUSANTARA

TANGERANG

2024

MECH BANK System



The MECH BANK

Banks play an important role in everyday life. The banking system is a network of financial institutions that provide various financial services to individuals and businesses so that banks

act as facilitators of the flow of funds, providers of financial services, and guardians of economic stability.

The reason we chose bank as our project is because banks consist of various classes and have a lot of interesting and complex data so it is very interesting to use as a project and we can also learn and develop various skills, as well as make useful contributions to society. We created a bank consisting of 12 classes, namely

Why is OOP significant to the Bank System?

Modern banking systems rely heavily on complex and reliable software. Object-Oriented Programming (OOP) offers an effective approach to building banking systems that are efficient, maintainable, and easy to develop. Several methods in OOP are useful for making Bank Systems more efficient and more effective, such as inheritance, polymorphism, encapsulation, abstraction, singly linked lists, doubly linked lists, circular doubly linked lists, stack, and queue. Using various OOP techniques, a code in the Bank System can be more structured and make the code easier to understand, change, and maintain. Programmers are also helped by using the OOP method in developing a complex system because code using the OOP method is easier to read and understand so programmers can increase the efficiency, scalability, security, and maintainability of their code.

The OOP method that we use in our bank system

A. Inheritance

One of the main pillars of OOP is inheritance, which allows inheriting methods and data from a class (superclass or parent class) to a new class (subclass or child class).

There are several advantages to using inheritance in the Account class:

- Code Duplication Reduction: Common properties and functions such as transaction recording, interest calculation, and customer information can be transferred from the Account class to subclasses. This reduces code duplication and increases development efficiency.
- Easier Code Maintenance: The code update and maintenance process becomes easier because property and function changes in the Account class will automatically be carried over to the subclass.

- Better Code Readability: The use of inheritance makes the code structure more organized and easy to understand because subclasses only need to define specific properties and functions for their account type.

B. Polymorphism

Polymorphism is also one of the main pillars of OOP that allows flexibility and ease in software development. If inheritance focuses on the inheritance of data and methods, polymorphism focuses on the ability of an object to respond to messages in different ways. We use encapsulation to display the displayAccountDetails function.

There are several advantages to using polymorphism:

- Flexibility: Polymorphism increases flexibility and ease of development because it allows developers to write generic code and work with different types of objects without needing to explicitly change the code.
- Ease of Maintenance: Polymorphism makes code easier to maintain and update because changes to the implementation of subclass methods do not require changes to the code that uses them.
- Extensibility: Polymorphism increases system scalability and expandability because it allows adding new object types easily without changing existing code.

C. Encapsulation

Encapsulation is one of the main methods in OOP which allows developers to wrap data and methods in a unit class.

There are several advantages to using encapsulation:

- Improve data security: Encapsulation allows us to hide sensitive data from unauthorized users.
- Increases code modularity: Encapsulation helps us to create more structured and easy-to-understand modules.
- Improves code maintainability: Encapsulation helps us to create code that is easier to change and maintain.

Because all data in the bank is confidential, therefore we use encapsulation in each class to maintain the confidentiality of each attribute in the class. The way to use it is to declare attributes as 'protected' or 'private' so that they can only be accessed directly by the class itself or subclasses inherited from the superclass.

D. Abstraction

One of the important concepts in object-oriented programming (OOP) languages is abstraction. The main goal is to overcome complexity by removing unimportant information from the user. We use abstraction for the Account class, where the Account class is the parent class of the classes we use.

Abstraction benefit:

- Increases code reusability: Abstraction allows us to create modules that can be reused in various situations.
- Improve code security: Abstractions allow us to hide sensitive implementation details from users.

E. Linked List

A linked list is a data structure commonly used in computer science to store a collection of elements. Unlike arrays, which store elements in contiguous memory locations, linked lists store elements in nodes where each node contains a data element and a reference (or pointer) to the next node in the sequence.

There are different types of linked lists, including:

1. **Singly Linked List**

   In this type, each node contains a data element and a pointer to the next node in the sequence. The last node points to null, indicating the end of the list.

   In our project, we use a singly linked list to keep track of deposit and withdrawal information. This helps us keep all the transactions organized.

2. **Doubly Linked List**

   In addition to a pointer to the next node, each node in a doubly linked list also contains a pointer to the previous node, allowing traversal in both directions.

   We use Doubly Linked List to hold all of the transaction data, so user can check each transaction one by one, and if the user accidentally skip some of the data they don’t need to pressing next until it repeats again from the start, user can turn back to the previous data since each node have 2 pointers to go to the next node and previous node.

3. **Circular Doubly Linked List**

   Doubly linked list, but the last node points back to the first node, forming a circle.

   In our project, we use a circular doubly linked list to hold all the user's data. It's like having a circle of information where you can move both forward and backward

through the data, and it never ends. It helps us manage and access user data efficiently.

4. **Stack**

   A stack is a linear data structure that follows the Last In, First Out (LIFO) principle, meaning the element that is pushed last onto the stack is the one that gets popped first. You can implement a stack using all linked list types

5. **Queue**

   A queue is a linear data structure that follows the First In, First Out (FIFO) principle, meaning the element that is enqueued first is the one that gets dequeued first. You can implement a queue using all linked list type

The unique Method that we use in each class

1. **Account**

The Account class has attributes and a constructor that can later be used for its subclasses, the attributes and the constructor in the account are as follows:

- public static final String TransactionNode = null;
- protected String accountNumber;
- protected String accountHolderName;
- protected double balance;
- protected String username;
- protected String password;
- private AccountNode head;
- private List<Transaction> transactionHistory;

public Account (String accountNumber, String accountHolderName, double balance, String username, String password)

The Account class also have a several method, such as:

- displayBalance()
- getBalance()
- getTransactionHistory()
- addTransaction(Transaction transaction)
- setBalance(double balance)
- getAccountNumber()
- getUsername()

- getPassword()
- getAccountHolderName()
- displayAccountDetails()
- getHead()

2. **AccountNode**

The AccountNode class also has attributes and a constructor that can be called and used in different classes, the attributes and the constructor in the account are as follows:

- private Account account;
- private AccountNode next;
- private AccountNode prev;

public AccountNode (Account transaction)

The AccountNode class also has a several methods, such as:

- setAccount(Account account)
- getAccount()
- setNext(AccountNode next)
- getNext()
- setPrev(AccountNode prev)
- getPrev()

3. **DebitCard**

The DebitCard class uses the inheritance method, this DebitCard class inherit from the account class and this class can inherit the attributes from the superclass, so the constructor in the DebitCard class is as follows:

- double dailyWithdrawalLimit;

public DebitCard (String accountNumber, String accountHolderName, double balance, double dailyWithdrawalLimit,String username, String password)

- addTransaction(Transaction transaction)
- displayAccountDetails()
- withdraw(double amount)

4. **SinglyLinkedList**

The SinglyLinkedList class has Node and several methods to insert and swapNode the data into the Node. Here are some of the attributes and methods used:

- private AccountNode head;
- swapNodes(AccountNode node1, AccountNode node2)
- getHead()
- setHead(AccountNode head)

5. **DoubleLinkedList**

The DoubleLinkedList class also has Node and several method to insert and swap the data into the node. Here are some of the attributes and methods used:

- private AccountNode head;
- private AccountNode tail;
- insert(Account account)
- swapNodes(AccountNode node1, AccountNode node2)
- getHead()
- setHead(AccountNode head)

6. **CircularSinglyLinkedList**

Similar to the CircularDoublyLinkedList class, it has Node and several methods to run it. Here are some of the attributes and methods used:

- private AccountNode head;
- insert(Account account)
- search(String query)
- display()

7. **CircularDoublyLinkedList**

The CircularDoublyLinkedList class has Node and several methods to run it. Here are some of the attributes and methods used:

- private AccountNode head;
- add(Account account)
- search(String query)
- display()
- getHead()
- setHead(AccountNode head)

8. **Stack**

The Stack class also has a Node for inserting the data into the node, there a several methods such as:

- private Node top;
- public void push(String data)
- String pop()
- isEmpty()
- clear()

9. Transaction

The Transaction class has several attributes, methods, and constructors for the transaction in MECH Bank. including:

- private String type;
- private double amount;
- private LocalDate date;
- private String accountNumber;

public Transaction (String type, double amount, LocalDate date, String accountNumber)

- getType()
- setType(String type)
- getAmount()
- setAmount(double amount)
- getDate()
- setDate(LocalDate date)
- getAccountNumber()
- setAccountNumber(String accountNumber)

10. TransactionNode

This TransactionNode class is a Node for Transaction class. It have attributes, methods, and also constructor in this class, such as:

- private int transactionId;
- private double amount;
- private String type;
- private LocalDate transactionDate;
- private TransactionNode next;
- private TransactionNode previous;

- private AccountNode head;
- private TransactionNode transactionNode;
- private Transaction transaction;

public TransactionNode (int transactionId, double amount, String type, LocalDate transactionDate)

- setNext(TransactionNode next)
- getNext()
- setPrev(TransactionNode prev)
- getPrev()
- setTransactionNode(TransactionNode transactionNode)
- getTransactionNode()
- setTransaction(Transaction transaction)
- getTransaction()
- getHead()
- setHead(AccountNode head)
- printTransactionDetails()

11. UndoRedoManager

This class has attributes and a several methods for undo/redo the transaction:

- private Stack<Transaction> undoStack;
- private Stack<Transaction> redoStack;
- private DoubleLinkedList transactionHistory;
- UndoRedoManager()
- addToUndo(Transaction transaction)
- UndoRedoDeposit()
- addDeposit(Account transaction)
- undoDepositTransaction()
- undoWithdrawalTransaction
- getAccountByNumber(String accountNumber)

12. Main

In the main class, there are several attributes and main methods for running each system/program in MECH bank, including:

- private static Account currentAccount;

- boolean login()
- Account login(CircularDoublyLinkedList accountList, String username, String password)
- Signup()
- ValidatePassword(String password)
- ViewAccount(Account account)
- Deposit(Account account, DoubleLinkedList depositList)
- Withdrawal(Account account, SinglyLinkedList withdrawalList)
- ViewTransactionHistory(Account account1)
- FindAccount()
- DepositTransactions(Account account)
- WithdrawalTransactions(Account account)
- AskQuestions(CircularSinglyLinkedList questionsList)
- generateTransactionCode(Transaction transaction)
- generateAccountNumber()

In the main method, there is a switch case and looping function to display the MECH Bank menu