RED & WHITE®
Multimedia Education
Shaping "skills" for "scaling" higher...!!!

C++

Project - 4

Banking System

Project Definition: Banking System

Overview:

Develop a simple banking system that demonstrates the concepts of polymorphism, encapsulation, and inheritance. The system should allow users to create different types of bank accounts, perform transactions, and retrieve account information.

Time Allocation:

- Total Duration: 4 Hours
- Total Marks: 10

Instructions:

1. Attempt all assigned tasks.
2. Make suitable assumptions wherever necessary.
3. Upload your exam task by uploading the project to GitHub and submitting the GitHub repository link which must have screenshots of your output in a README.md file.
4. This project is individual-based; copying code from classmates is prohibited.

Remember to follow the instructions provided professionally, make suitable assumptions wherever necessary, and avoid copying code or content from any unauthorized sources. Good luck with your project work!

Project Criteria:

Requirements:

- Class & Object
- Polymorphism
- Inheritance
- Encapsulation

Class Structure:

1. **Base Class:** BankAccount

   Attributes:

   - accountNumber
   - accountHolderName
   - balance

   Methods:

   - deposit(amount: double): Adds funds to the account.
   - withdraw(amount: double): Withdraws funds from the account.

- getBalance(): double: Returns the current balance.
- displayAccountInfo(): Displays account details.

2. **Derived Class:** SavingsAccount (from BankAccount)
   - **Attribute:** interestRate
   - **Method:** calculateInterest(): Calculates interest based on the balance and interest rate.

3. **Derived Class:** CheckingAccount (from BankAccount)
   - **Attribute:** overdraftLimit
   - **Method:** checkOverdraft(): Checks if a withdrawal exceeds the overdraft limit.

4. **Derived Class:** FixedDepositAccount (from BankAccount)
   - **Attribute:** term (duration in months)
   - **Method:** calculateInterest(): Calculates fixed deposit interest.

Requirements:
1. Create instances of different account types (savings, checking, fixed deposit).
2. Demonstrate polymorphism by calling methods (e.g., calculateInterest()) through the base class pointer.
3. Ensure encapsulation by making account balances private and providing appropriate accessors.
4. Implement transaction methods (deposit, withdraw) for each account type.
5. Create a user-friendly menu-driven interface for account management.

Marking Criteria: (Total 10 Marks)
Logic: 8 Mark
Output: 2 Mark