**OOAD**

**1. What is OOAD?**

**Object-Oriented Analysis and Design (OOAD)** is the process of **analyzing** a problem and **designing** a solution using **object-oriented principles**.

🔄 OOAD = Understanding the problem (Analysis) + Creating the solution (Design) using Objects.

**2. OOAD Phases**

| **Phase** | **Description** |
| --- | --- |
| **Object-Oriented Analysis (OOA)** | Understand the real-world problem and identify objects/entities. |
| **Object-Oriented Design (OOD)** | Create class diagrams, interactions, and design patterns to build software architecture. |
| **Object-Oriented Programming (OOP)** | Implement the design using OOP languages like Java. |

**3. Banking Domain Example**

**Problem**: Build a system to manage **Bank Accounts**, perform **Deposits**, **Withdrawals**, and **Account Statement generation**.

**4. Object-Oriented Analysis (OOA)**

**Identify Real-World Objects:**

* Customer
* Account
* Transaction
* Bank
* ATM (optional)

**Identify Attributes and Behaviors:**

| **Object** | **Attributes** | **Behaviors** |
| --- | --- | --- |
| Customer | name, ID, address | openAccount(), getDetails() |
| Account | accNumber, balance, type | deposit(), withdraw(), getBalance() |
|  |  |  |
| Transaction | date, amount, type (debit/credit) | printTransaction() |

**5. Object-Oriented Design (OOD)**

**🔷 UML Class Diagram (Textual Form)**

Customer

└── name: String

└── id: int

└── accounts: List<Account>

└── openAccount()

└── getDetails()

Account

└── accNumber: int

└── balance: double

└── transactions: List<Transaction>

└── deposit()

└── withdraw()

└── getStatement()

Transaction

└── date: Date

└── amount: double

└── type: String

└── printTransaction()

**6. Java Implementation (Simplified)**

import java.util.\*;

class Transaction {

Date date;

double amount;

String type;

public Transaction(double amount, String type) {

this.date = new Date();

this.amount = amount;

this.type = type;

}

public void printTransaction() {

System.out.println(date + " | " + type + " | $" + amount);

}

}

class Account {

int accNumber;

double balance;

List<Transaction> transactions = new ArrayList<>();

public Account(int accNumber) {

this.accNumber = accNumber;

this.balance = 0;

}

public void deposit(double amount) {

balance += amount;

transactions.add(new Transaction(amount, "CREDIT"));

}

public void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

transactions.add(new Transaction(amount, "DEBIT"));

} else {

System.out.println("Insufficient Balance");

}

}

public void getStatement() {

System.out.println("Account No: " + accNumber + " | Balance: $" + balance);

for (Transaction t : transactions) {

t.printTransaction();

}

}

}

class Customer {

String name;

int id;

List<Account> accounts = new ArrayList<>();

public Customer(String name, int id) {

this.name = name;

this.id = id;

}

public void openAccount(int accNumber) {

accounts.add(new Account(accNumber));

}

public void showAccounts() {

System.out.println("Customer: " + name);

for (Account acc : accounts) {

acc.getStatement();

}

}

}

**7. Testing the Banking System**

java

CopyEdit

public class BankingApp {

public static void main(String[] args) {

Customer c1 = new Customer("Alice", 101);

c1.openAccount(12345);

Account acc = c1.accounts.get(0);

acc.deposit(1000);

acc.withdraw(300);

c1.showAccounts();

}

}

**8. Key OOAD Concepts Demonstrated**

| **Concept** | **Code Example** |
| --- | --- |
| **Encapsulation** | Attributes are private, accessed via methods |
| **Abstraction** | Only essential details shown (deposit, withdraw) |
| **Inheritance** | You can extend Account into SavingsAccount, CurrentAccount |
| **Polymorphism** | Overriding withdraw() for specific account types |
| **Association/Aggregation** | Customer has a list of Account (Has-A relationship) |

**9. Advantages of Using OOAD in Banking**

* **Modular**: Easy to update business logic.
* **Scalable**: Add new features like Loans, Credit Cards easily.
* **Real-world Mapping**: Objects map naturally to real entities.
* **Reusable**: Classes like Transaction and Account can be reused across modules.

**10. Summary**

| **OOAD Step** | **Banking Example** |
| --- | --- |
| Analysis | Identify: Customer, Account, Transaction |
| Design | Define relationships, behaviors |
| Programming | Java classes & objects implementation |