Topic 1: Class Diagram – Structure Overview

class Account {

private String accountNumber;

private double balance;

private String accountType;

public Account(String accountNumber, String accountType) {

this.accountNumber = accountNumber;

this.accountType = accountType;

this.balance = 0.0;

}

public void deposit(double amount) {

balance += amount;

System.out.println("Deposited: $" + amount + " | Balance: $" + balance);

}

public void withdraw(double amount) {

if(balance >= amount) {

balance -= amount;

System.out.println("Withdrawn: $" + amount + " | Balance: $" + balance);

} else {

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

}

}

public double getBalance() {

return balance;

}

public String getAccountNumber() {

return accountNumber;

}

}

class Customer {

private String customerId;

private String name;

private String email;

private Account[] accounts;

private int accountCount;

public Customer(String customerId, String name, String email) {

this.customerId = customerId;

this.name = name;

this.email = email;

this.accounts = new Account[5];

this.accountCount = 0;

}

public void addAccount(Account account) {

if(accountCount < 5) {

accounts[accountCount] = account;

accountCount++;

System.out.println("Account added for " + name);

}

}

public void displayAccounts() {

System.out.println("\nCustomer: " + name);

for(int i = 0; i < accountCount; i++) {

System.out.println("Account: " + accounts[i].getAccountNumber() +

" | Balance: $" + accounts[i].getBalance());

}

}

public String getName() {

return name;

}

}

class Bank {

private String bankName;

private String branchCode;

private Customer[] customers;

private int customerCount;

public Bank(String bankName, String branchCode) {

this.bankName = bankName;

this.branchCode = branchCode;

this.customers = new Customer[100];

this.customerCount = 0;

}

public void addCustomer(Customer customer) {

customers[customerCount] = customer;

customerCount++;

System.out.println("Customer registered: " + customer.getName());

}

public void displayAllCustomers() {

System.out.println("\n=== " + bankName + " ===");

for(int i = 0; i < customerCount; i++) {

customers[i].displayAccounts();

}

}

}

public class Main {

public static void main(String[] args) {

Bank bank = new Bank("State Bank", "BR001");

Customer customer1 = new Customer("C001", "Rahul", "rahul@email.com");

Customer customer2 = new Customer("C002", "Priya", "priya@email.com");

Account acc1 = new Account("ACC1001", "Savings");

Account acc2 = new Account("ACC1002", "Current");

Account acc3 = new Account("ACC1003", "Savings");

customer1.addAccount(acc1);

customer1.addAccount(acc2);

customer2.addAccount(acc3);

bank.addCustomer(customer1);

bank.addCustomer(customer2);

System.out.println("\n--- Transactions ---");

acc1.deposit(5000);

acc1.withdraw(2000);

acc2.deposit(10000);

acc3.deposit(8000);

acc3.withdraw(1000);

bank.displayAllCustomers();

}

}

Topic 2: Object Diagram – Runtime Instances

class Account {

String accountNumber;

String accountType;

double balance;

public Account(String accountNumber, String accountType, double balance) {

this.accountNumber = accountNumber;

this.accountType = accountType;

this.balance = balance;

}

public void display() {

System.out.println("Account: " + accountNumber);

System.out.println("Type: " + accountType);

System.out.println("Balance: $" + balance);

}

}

class Customer {

String customerId;

String name;

Account account;

public Customer(String customerId, String name) {

this.customerId = customerId;

this.name = name;

}

public void setAccount(Account account) {

this.account = account;

}

public void display() {

System.out.println("Customer ID: " + customerId);

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

if (account != null) {

account.display();

}

System.out.println();

}

}

class Bank {

String bankName;

Customer[] customers;

int count;

public Bank(String bankName, int max) {

this.bankName = bankName;

this.customers = new Customer[max];

this.count = 0;

}

public void addCustomer(Customer customer) {

if (count < customers.length) {

customers[count] = customer;

count++;

}

}

public void display() {

System.out.println("Bank: " + bankName);

System.out.println("Total Customers: " + count);

System.out.println();

for (int i = 0; i < count; i++) {

customers[i].display();

}

}

}

public class ObjectDiagramDemo {

public static void main(String[] args) {

Bank bank1 = new Bank("City Bank", 3);

Account acc1 = new Account("ACC001", "Savings", 5000.0);

Account acc2 = new Account("ACC002", "Checking", 12000.0);

Customer cust1 = new Customer("C001", "John Smith");

Customer cust2 = new Customer("C002", "Sarah Johnson");

cust1.setAccount(acc1);

cust2.setAccount(acc2);

bank1.addCustomer(cust1);

bank1.addCustomer(cust2);

bank1.display();

}

}

Topic 3: Sequence Diagram – Method Interaction Over Time

class Cart {

String[] items;

int count;

double total;

public Cart() {

items = new String[10];

count = 0;

total = 0;

}

public boolean addItem(String item, double price) {

System.out.println(" -> Cart.addItem(" + item + ", $" + price + ")");

if (count < items.length) {

items[count] = item;

count++;

total += price;

System.out.println(" <- Item added successfully");

return true;

}

System.out.println(" <- Cart is full");

return false;

}

public double getTotal() {

System.out.println(" -> Cart.getTotal()");

System.out.println(" <- Total: $" + total);

return total;

}

}

class PaymentService {

public boolean makePayment(double amount, String cardNumber) {

System.out.println(" -> PaymentService.makePayment($" + amount + ", " + cardNumber + ")");

System.out.println(" Processing payment...");

System.out.println(" <- Payment successful");

return true;

}

}

class OrderService {

int orderNumber = 1000;

public String confirmOrder(String customerName, double amount) {

System.out.println(" -> OrderService.confirmOrder(" + customerName + ", $" + amount + ")");

String orderId = "ORD" + orderNumber;

orderNumber++;

System.out.println(" Creating order...");

System.out.println(" <- Order confirmed: " + orderId);

return orderId;

}

}

class Customer {

String name;

String cardNumber;

public Customer(String name, String cardNumber) {

this.name = name;

this.cardNumber = cardNumber;

}

public void placeOrder(Cart cart, PaymentService payment, OrderService order) {

System.out.println("\nCustomer: " + name + " starts placing order");

System.out.println();

cart.addItem("Laptop", 800.0);

cart.addItem("Mouse", 20.0);

System.out.println();

double total = cart.getTotal();

System.out.println();

boolean paymentSuccess = payment.makePayment(total, cardNumber);

System.out.println();

if (paymentSuccess) {

String orderId = order.confirmOrder(name, total);

System.out.println();

System.out.println("Customer receives order ID: " + orderId);

}

}

}

public class SequenceDiagramDemo {

public static void main(String[] args) {

System.out.println("SEQUENCE DIAGRAM - Online Shopping System\n");

Customer customer = new Customer("John", "1234-5678-9012");

Cart cart = new Cart();

PaymentService payment = new PaymentService();

OrderService order = new OrderService();

customer.placeOrder(cart, payment, order);

System.out.println("\n\nSequence Flow:");

System.out.println("1. Customer -> Cart: addItem()");

System.out.println("2. Cart -> Customer: return success");

System.out.println("3. Customer -> Cart: getTotal()");

System.out.println("4. Cart -> Customer: return total");

System.out.println("5. Customer -> PaymentService: makePayment()");

System.out.println("6. PaymentService -> Customer: return success");

System.out.println("7. Customer -> OrderService: confirmOrder()");

System.out.println("8. OrderService -> Customer: return orderID");

}

}

Topic 5: Activity Diagram – Workflow Visualization

import java.util.Scanner;

class RegistrationSystem {

public boolean validateForm(String name, String email) {

System.out.println("[System] Validating form data...");

if (name.length() > 0 && email.contains("@")) {

System.out.println("[System] Form is valid");

return true;

}

System.out.println("[System] Form is invalid");

return false;

}

public boolean validateDocuments(String docName) {

System.out.println("[System] Checking documents...");

if (docName.length() > 0) {

System.out.println("[System] Documents accepted");

return true;

}

System.out.println("[System] Documents rejected");

return false;

}

public boolean processPayment(double amount) {

System.out.println("[System] Processing payment of $" + amount);

if (amount > 0) {

System.out.println("[System] Payment successful");

return true;

}

System.out.println("[System] Payment failed");

return false;

}

public String generateConfirmation(String name) {

System.out.println("[System] Generating confirmation...");

String confirmationId = "REG" + (int)(Math.random() \* 9000 + 1000);

System.out.println("[System] Registration complete!");

return confirmationId;

}

}

class Student {

String name;

String email;

public void fillForm(Scanner sc) {

System.out.println("\n[User] Filling registration form");

System.out.print("Enter name: ");

name = sc.nextLine();

System.out.print("Enter email: ");

email = sc.nextLine();

}

public String uploadDocuments(Scanner sc) {

System.out.println("\n[User] Uploading documents");

System.out.print("Enter document name: ");

return sc.nextLine();

}

public double payFees(Scanner sc) {

System.out.println("\n[User] Paying registration fees");

System.out.print("Enter amount: ");

return sc.nextDouble();

}

}

public class ActivityDiagramDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Student student = new Student();

RegistrationSystem system = new RegistrationSystem();

System.out.println("ACTIVITY DIAGRAM - University Registration");

System.out.println("START\n");

boolean formValid = false;

while (!formValid) {

student.fillForm(sc);

formValid = system.validateForm(student.name, student.email);

if (!formValid) {

System.out.println("\nRetry filling form");

}

}

boolean docsValid = false;

while (!docsValid) {

String docName = student.uploadDocuments(sc);

docsValid = system.validateDocuments(docName);

if (!docsValid) {

System.out.println("\nRetry uploading documents");

}

}

boolean paymentDone = false;

while (!paymentDone) {

double amount = student.payFees(sc);

sc.nextLine();

paymentDone = system.processPayment(amount);

if (!paymentDone) {

System.out.println("\nRetry payment");

}

}

String confirmationId = system.generateConfirmation(student.name);

System.out.println("\n[User] Received Confirmation ID: " + confirmationId);

System.out.println("\nEND");

System.out.println("\n\nActivity Flow:");

System.out.println("START -> Fill Form -> [Valid?]");

System.out.println(" [No] -> Retry Fill Form");

System.out.println(" [Yes] -> Upload Documents -> [Valid?]");

System.out.println(" [No] -> Retry Upload");

System.out.println(" [Yes] -> Pay Fees -> [Success?]");

System.out.println(" [No] -> Retry Payment");

System.out.println(" [Yes] -> Get Confirmation -> END");

sc.close();

}

}