1..Examine an ATM System: Select an ATM system, thoroughly review its system specifications, and identify and document the various software defects present.  
  
import java.util.\*;  
  
public class ATMApp {  
  
 *// ----- Domain model -----* static class Account {  
 private final String accountNumber;  
 private final String pinHash; *// simple demo hash* private long balanceCents; *// store as cents to avoid float errors* private boolean locked;  
 private int failedPinAttempts;  
  
 Account(String accountNumber, String pin, long startingBalanceCents) {  
 this.accountNumber = accountNumber;  
 this.pinHash = *simpleHash*(pin);  
 this.balanceCents = startingBalanceCents;  
 this.locked = false;  
 this.failedPinAttempts = 0;  
 }  
  
 boolean isLocked() {  
 return locked;  
 }  
  
 String getAccountNumber() {  
 return accountNumber;  
 }  
  
 static String simpleHash(String s) {  
 *// Very basic placeholder hash for demo only* return Integer.*toHexString*(Objects.*hash*(s));  
 }  
  
 boolean validatePin(String pinInput) {  
 if (locked) return false;  
 boolean ok = *simpleHash*(pinInput).equals(pinHash);  
 if (!ok) {  
 failedPinAttempts = failedPinAttempts + 1;  
 if (failedPinAttempts >= 3) {  
 locked = true;  
 }  
 } else {  
 failedPinAttempts = 0; *// reset on success* }  
 return ok;  
 }  
  
 long getBalanceCents() {  
 return balanceCents;  
 }  
  
 boolean deposit(long amountCents) {  
 if (amountCents <= 0) return false;  
 balanceCents = balanceCents + amountCents;  
 return true;  
 }  
  
 boolean withdraw(long amountCents) {  
 if (amountCents <= 0) return false;  
 if (amountCents > balanceCents) return false;  
 balanceCents = balanceCents - amountCents; *// atomic update* return true;  
 }  
 }  
  
 *// ----- ATM core -----* static class ATM {  
 private final Map<String, Account> accounts = new HashMap<>();  
 private final Scanner scanner;  
  
 ATM(Scanner scanner) {  
 this.scanner = scanner;  
 }  
  
 void addDemoAccount(String acct, String pin, long startingBalanceCents) {  
 accounts.put(acct, new Account(acct, pin, startingBalanceCents));  
 }  
  
 void start() {  
 println("==== Welcome to DemoBank ATM ====");  
 Account account = authenticate();  
 if (account == null) {  
 println("Session ended.");  
 return;  
 }  
 runMenu(account);  
 println("Thank you for banking with us.");  
 }  
  
 private Account authenticate() {  
 println("Insert card (enter account number):");  
 String acct = readLine();  
 Account account = accounts.get(acct);  
  
 if (account == null) {  
 println("Error: Account not found.");  
 return null;  
 }  
 if (account.isLocked()) {  
 println("This account is locked due to multiple invalid PIN attempts. Please contact support.");  
 return null;  
 }  
  
 *// PIN attempts handled by Account* int tries = 0;  
 while (tries < 3 && !account.isLocked()) {  
 println("Enter PIN:");  
 String pin = readLine();  
 boolean ok = account.validatePin(pin);  
 if (ok) {  
 println("Authentication successful.");  
 return account;  
 } else {  
 tries = tries + 1;  
 if (account.isLocked()) {  
 println("Too many invalid attempts. Your account has been locked. Please contact support.");  
 return null;  
 } else {  
 println("Invalid PIN. Attempts left: " + (3 - tries));  
 }  
 }  
 }  
 return null;  
 }  
  
 private void runMenu(Account account) {  
 boolean running = true;  
 while (running) {  
 println("\n---- Main Menu ----");  
 println("1) Balance Inquiry");  
 println("2) Withdraw");  
 println("3) Deposit");  
 println("4) Exit");  
 println("Select an option (1-4):");  
  
 String choice = readLine().trim();  
  
 if (choice.equals("1")) {  
 showBalance(account);  
 } else if (choice.equals("2")) {  
 handleWithdraw(account);  
 } else if (choice.equals("3")) {  
 handleDeposit(account);  
 } else if (choice.equals("4")) {  
 running = false;  
 } else {  
 println("Invalid option. Please choose 1-4.");  
 }  
 }  
 }  
  
 private void showBalance(Account account) {  
 println("Current Balance: " + formatCurrency(account.getBalanceCents()));  
 }  
  
 private void handleWithdraw(Account account) {  
 println("Enter amount to withdraw (e.g., 100.00):");  
 Long cents = parseAmountToCents(readLine());  
 if (cents == null) {  
 println("Invalid amount. Use numeric value like 100.00");  
 return;  
 }  
 boolean ok = account.withdraw(cents);  
 if (ok) {  
 printReceipt("WITHDRAWAL", cents, account.getBalanceCents());  
 } else {  
 if (cents != null && cents > account.getBalanceCents()) {  
 println("Insufficient funds.");  
 } else {  
 println("Withdrawal failed. Amount must be greater than 0.");  
 }  
 }  
 }  
  
 private void handleDeposit(Account account) {  
 println("Enter amount to deposit (e.g., 250.00):");  
 Long cents = parseAmountToCents(readLine());  
 if (cents == null) {  
 println("Invalid amount. Use numeric value like 250.00");  
 return;  
 }  
 boolean ok = account.deposit(cents);  
 if (ok) {  
 printReceipt("DEPOSIT", cents, account.getBalanceCents());  
 } else {  
 println("Deposit failed. Amount must be greater than 0.");  
 }  
 }  
  
 private void printReceipt(String type, long amountCents, long newBalanceCents) {  
 println("\n----- RECEIPT -----");  
 println("Type: " + type);  
 println("Amount: " + formatCurrency(amountCents));  
 println("New Balance: " + formatCurrency(newBalanceCents));  
 println("Date/Time: " + new Date());  
 println("-------------------\n");  
 }  
  
 *// ----- Helpers -----* private String formatCurrency(long cents) {  
 long rupees = cents / 100;  
 long remainder = Math.*abs*(cents % 100);  
 return "₹" + rupees + "." + (remainder < 10 ? "0" + remainder : remainder);  
 }  
  
 private Long parseAmountToCents(String s) {  
 try {  
 String t = s.trim().replace(",", "");  
 double d = Double.*parseDouble*(t);  
 if (d <= 0) return null;  
 long cents = Math.*round*(d \* 100.0);  
 return cents;  
 } catch (Exception e) {  
 return null;  
 }  
 }  
  
 private String readLine() {  
 return scanner.nextLine();  
 }  
  
 private void println(String s) {  
 System.*out*.println(s);  
 }  
 }  
  
 *// ----- Main with demo data -----* public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 ATM atm = new ATM(sc);  
  
 *// Demo accounts (PINs for testing: 1234, 4321)  
 // Starting balance ₹10,000.00 and ₹500.00* atm.addDemoAccount("111122223333", "1234", 1\_000\_000);  
 atm.addDemoAccount("999988887777", "4321", 50\_000);  
  
 atm.start();  
 }  
}  
  
2.Case Study on ARMS Portals (Positive Testing): Conduct a case study to analyze the requirements of ARMS Portals and develop positive test scenarios and test cases for the application.  
  
import java.util.Scanner;  
class ARMSPortalSystem {  
 boolean login(String username, String password) {  
 if (username.isEmpty() || password.isEmpty()) {  
 System.*out*.println("Error: Username and Password cannot be empty.");  
 return false;  
 }  
 if (!(username.equals("admin") && password.equals("admin123"))) {  
 System.*out*.println("Error: Invalid username or password.");  
 return false;  
 }  
 System.*out*.println("Login successful!");  
 return true;  
 }  
 boolean register(String email, String password) {  
 if (!email.contains("@") || !email.contains(".")) {  
 System.*out*.println("Error: Please enter a valid email address.");  
 return false;  
 }  
 if (password.length() < 8) {  
 System.*out*.println("Error: Password must be at least 8 characters.");  
 return false;  
 }  
 System.*out*.println("Registration successful!");  
 return true;  
 }  
  
 *// File upload validation* boolean uploadFile(String filename) {  
 if (!filename.endsWith(".pdf")) {  
 System.*out*.println("Error: Invalid file type. Only PDF allowed.");  
 return false;  
 }  
 System.*out*.println("File uploaded successfully.");  
 return true;  
 }  
}  
public class ARMSPortalTest {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 ARMSPortalSystem portal = new ARMSPortalSystem();  
 System.*out*.println("=== LOGIN TEST ===");  
 System.*out*.print("Enter username: ");  
 String user = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String pass = sc.nextLine();  
 portal.login(user, pass);  
  
 System.*out*.println("\n=== REGISTRATION TEST ===");  
 System.*out*.print("Enter email: ");  
 String email = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String regPass = sc.nextLine();  
 portal.register(email, regPass);  
  
 System.*out*.println("\n=== FILE UPLOAD TEST ===");  
 System.*out*.print("Enter file name: ");  
 String file = sc.nextLine();  
 portal.uploadFile(file);  
  
 sc.close();  
 }  
}

3.Case Study on ARMS Portals (Negative Testing): Conduct a case study to analyze the requirements of ARMS Portals and develop negative test scenarios and test cases for the application.  
  
import java.util.Scanner;  
  
class ARMSPortalService {  
  
 boolean validateLogin(String username, String password) {  
 if (username.isEmpty() || password.isEmpty()) {  
 System.*out*.println("Error: Username and Password cannot be empty.");  
 return false;  
 }  
 if (!(username.equals("admin") && password.equals("admin123"))) {  
 System.*out*.println("Error: Invalid username or password.");  
 return false;  
 }  
 System.*out*.println("Login successful!");  
 return true;  
 }  
  
  
 boolean validateRegistration(String email, String password) {  
 if (!email.contains("@") || !email.contains(".")) {  
 System.*out*.println("Error: Please enter a valid email address.");  
 return false;  
 }  
 if (password.length() < 8) {  
 System.*out*.println("Error: Password must be at least 8 characters.");  
 return false;  
 }  
 System.*out*.println("Registration successful!");  
 return true;  
 }  
  
 boolean validateFileUpload(String filename) {  
 if (!filename.endsWith(".pdf")) {  
 System.*out*.println("Error: Invalid file type. Only PDF allowed.");  
 return false;  
 }  
 System.*out*.println("File uploaded successfully.");  
 return true;  
 }  
}  
  
  
public class ARMSPortalRunner {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 ARMSPortalService service = new ARMSPortalService();  
  
  
 System.*out*.println("=== LOGIN TEST ===");  
 System.*out*.print("Enter username: ");  
 String user = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String pass = sc.nextLine();  
 service.validateLogin(user, pass);  
  
  
 System.*out*.println("\n=== REGISTRATION TEST ===");  
 System.*out*.print("Enter email: ");  
 String email = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String regPass = sc.nextLine();  
 service.validateRegistration(email, regPass);  
  
  
 System.*out*.println("\n=== FILE UPLOAD TEST ===");  
 System.*out*.print("Enter file name: ");  
 String file = sc.nextLine();  
 service.validateFileUpload(file);  
  
 sc.close();  
 }  
}

4.Case Study on e-Commerce Applications (Positive Testing): Conduct a case study to analyze the requirements of e-commerce applications and create positive test scenarios and test cases for the application.  
  
import java.util.\*;  
  
class Product {  
 private int id;  
 private String name;  
 private double price;  
 private int stock;  
  
 public Product(int id, String name, double price, int stock) {  
 this.id = id;  
 this.name = name;  
 this.price = price;  
 this.stock = stock;  
 }  
  
 public int getId() { return id; }  
 public String getName() { return name; }  
 public double getPrice() { return price; }  
 public int getStock() { return stock; }  
  
 public void reduceStock(int quantity) {  
 this.stock -= quantity;  
 }  
}  
  
class Cart {  
 private Map<Product, Integer> items = new HashMap<>();  
  
 public void addProduct(Product p, int qty) {  
 items.put(p, items.getOrDefault(p, 0) + qty);  
 }  
  
 public boolean isEmpty() {  
 return items.isEmpty();  
 }  
  
 public double getTotal() {  
 double total = 0;  
 for (Map.Entry<Product, Integer> entry : items.entrySet()) {  
 total += entry.getKey().getPrice() \* entry.getValue();  
 }  
 return total;  
 }  
  
 public void showCart() {  
 if (items.isEmpty()) {  
 System.*out*.println("Cart is empty.");  
 return;  
 }  
 System.*out*.println("Your Cart:");  
 for (Map.Entry<Product, Integer> entry : items.entrySet()) {  
 System.*out*.println(entry.getKey().getName() + " x " + entry.getValue());  
 }  
 System.*out*.println("Total: $" + getTotal());  
 }  
}  
  
class Payment {  
 public boolean processPayment(String cardNumber, double amount) {  
 *// Negative test: Card number must be 16 digits* if (cardNumber.length() != 16 || !cardNumber.matches("\\d+")) {  
 return false;  
 }  
 return true;  
 }  
}  
  
class ECommerceSystem {  
 private List<Product> products = new ArrayList<>();  
 private Cart cart = new Cart();  
 private Scanner sc = new Scanner(System.*in*);  
  
 public void addDemoProducts() {  
 products.add(new Product(1, "Laptop", 1000.00, 5));  
 products.add(new Product(2, "Phone", 500.00, 10));  
 products.add(new Product(3, "Headphones", 50.00, 0)); *// out of stock* }  
  
 public void start() {  
 boolean running = true;  
 while (running) {  
 System.*out*.println("\n--- E-Commerce Menu ---");  
 System.*out*.println("1. Browse Products");  
 System.*out*.println("2. Add to Cart");  
 System.*out*.println("3. View Cart");  
 System.*out*.println("4. Checkout");  
 System.*out*.println("5. Exit");  
 System.*out*.print("Enter choice: ");  
 String choice = sc.nextLine();  
  
 switch (choice) {  
 case "1": browseProducts(); break;  
 case "2": addToCart(); break;  
 case "3": cart.showCart(); break;  
 case "4": checkout(); break;  
 case "5": running = false; break;  
 default: System.*out*.println("Invalid choice.");  
 }  
 }  
 }  
  
 private void browseProducts() {  
 System.*out*.println("\nAvailable Products:");  
 for (Product p : products) {  
 System.*out*.println(p.getId() + ". " + p.getName() + " - $" + p.getPrice() + " (Stock: " + p.getStock() + ")");  
 }  
 }  
  
 private void addToCart() {  
 System.*out*.print("Enter product ID to add: ");  
 String idInput = sc.nextLine();  
 int pid;  
 try {  
 pid = Integer.*parseInt*(idInput);  
 } catch (NumberFormatException e) {  
 System.*out*.println("Error: Invalid product ID.");  
 return;  
 }  
  
 Product selected = null;  
 for (Product p : products) {  
 if (p.getId() == pid) {  
 selected = p;  
 break;  
 }  
 }  
  
 if (selected == null) {  
 System.*out*.println("Error: Product not found.");  
 return;  
 }  
  
 System.*out*.print("Enter quantity: ");  
 String qtyInput = sc.nextLine();  
 int qty;  
 try {  
 qty = Integer.*parseInt*(qtyInput);  
 } catch (NumberFormatException e) {  
 System.*out*.println("Error: Invalid quantity.");  
 return;  
 }  
  
 if (qty <= 0) {  
 System.*out*.println("Error: Quantity must be positive.");  
 return;  
 }  
  
 if (qty > selected.getStock()) {  
 System.*out*.println("Error: Requested quantity not available (Stock: " + selected.getStock() + ")");  
 return;  
 }  
  
 cart.addProduct(selected, qty);  
 selected.reduceStock(qty);  
 System.*out*.println("Added " + qty + " x " + selected.getName() + " to cart.");  
 }  
  
 private void checkout() {  
 if (cart.isEmpty()) {  
 System.*out*.println("Error: Your cart is empty. Please add items first.");  
 return;  
 }  
  
 cart.showCart();  
 System.*out*.print("Enter 16-digit card number for payment: ");  
 String card = sc.nextLine();  
  
 Payment payment = new Payment();  
 boolean success = payment.processPayment(card, cart.getTotal());  
  
 if (!success) {  
 System.*out*.println("Error: Invalid payment details. Please try again.");  
 } else {  
 System.*out*.println("Payment successful! Order placed.");  
 }  
 }  
}  
  
public class ECommerceApp {  
 public static void main(String[] args) {  
 ECommerceSystem system = new ECommerceSystem();  
 system.addDemoProducts();  
 system.start();  
 }  
}  
  
5.Case Study on e-Commerce Applications (Negative Testing): Conduct a case study to analyze the requirements of e-commerce applications and create negative test scenarios and test cases for the application  
  
import java.util.\*;  
  
class ECommerceNegativeTests {  
 private static Map<String, String> *users* = new HashMap<>();  
  
 static {  
 *// Predefined user  
 users*.put("user1", "Password@123");  
 }  
  
 *// Login function* public static boolean login(String username, String password) {  
 if (username.isEmpty() || password.isEmpty()) {  
 System.*out*.println("❌ Error: Username/Password cannot be empty.");  
 return false;  
 }  
 if (!*users*.containsKey(username) || !*users*.get(username).equals(password)) {  
 System.*out*.println("❌ Error: Invalid username or password.");  
 return false;  
 }  
 System.*out*.println("✅ Login successful.");  
 return true;  
 }  
  
 *// Add to cart function* public static boolean addToCart(int quantity, int stock) {  
 if (quantity <= 0) {  
 System.*out*.println("❌ Error: Quantity must be positive.");  
 return false;  
 }  
 if (quantity > stock) {  
 System.*out*.println("❌ Error: Quantity exceeds stock.");  
 return false;  
 }  
 System.*out*.println("✅ Item added to cart.");  
 return true;  
 }  
  
 public static void main(String[] args) {  
 *// Negative test cases* System.*out*.println("=== Negative Test Cases ===");  
  
 *// Login tests  
 login*("", "Password@123"); *// Empty username  
 login*("user1", ""); *// Empty password  
 login*("user1", "wrongPass"); *// Wrong password  
 login*("unknown", "Password@123"); *// Invalid user  
  
 // Cart tests  
 addToCart*(-2, 10); *// Negative quantity  
 addToCart*(20, 5); *// Quantity exceeds stock* }  
}

6.Case Study on Food Delivery Applications (Positive Testing): Conduct a case study to analyze the requirements of food delivery applications and create positive test scenarios and test cases for the application.  
  
import java.util.\*;  
  
class FoodDeliveryPositiveTests {  
 private static Map<String, String> *users* = new HashMap<>();  
 private static List<String> *menu* = Arrays.*asList*("Pizza", "Burger", "Pasta", "Sandwich");  
  
 static {  
 *// Predefined user  
 users*.put("John", "Pass@123");  
 }  
  
 *// Registration* public static boolean register(String name, String email, String password) {  
 if (!*users*.containsKey(name)) {  
 *users*.put(name, password);  
 System.*out*.println("✅ Registration successful for " + name);  
 return true;  
 } else {  
 System.*out*.println("❌ User already exists.");  
 return false;  
 }  
 }  
  
 *// Login* public static boolean login(String username, String password) {  
 if (*users*.containsKey(username) && *users*.get(username).equals(password)) {  
 System.*out*.println("✅ Login successful for " + username);  
 return true;  
 } else {  
 System.*out*.println("❌ Invalid login.");  
 return false;  
 }  
 }  
  
 *// Browse menu* public static void searchFood(String item) {  
 if (*menu*.contains(item)) {  
 System.*out*.println("✅ " + item + " is available in the menu.");  
 } else {  
 System.*out*.println("❌ Item not found.");  
 }  
 }  
  
 *// Add to cart* public static boolean addToCart(String item, int qty) {  
 if (*menu*.contains(item) && qty > 0) {  
 System.*out*.println("✅ Added " + qty + " x " + item + " to cart.");  
 return true;  
 }  
 return false;  
 }  
  
 *// Place order* public static boolean placeOrder(String address) {  
 if (!address.isEmpty()) {  
 System.*out*.println("✅ Order placed successfully. Delivery to: " + address);  
 return true;  
 }  
 return false;  
 }  
  
 *// Payment* public static boolean payment(String cardNo, String expDate, String cvv) {  
 if (cardNo.length() == 16 && cvv.length() == 3) {  
 System.*out*.println("✅ Payment successful with card ending " + cardNo.substring(12));  
 return true;  
 }  
 return false;  
 }  
  
 *// Delivery tracking* public static void trackOrder(int orderId) {  
 System.*out*.println("✅ Order #" + orderId + " is confirmed.");  
 System.*out*.println("✅ Order #" + orderId + " is out for delivery.");  
 System.*out*.println("✅ Order #" + orderId + " has been delivered.");  
 }  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
  
 System.*out*.println("=== Food Delivery Positive Test ===");  
  
 *// Registration* System.*out*.print("Enter your name for registration: ");  
 String name = sc.nextLine();  
 System.*out*.print("Enter your email: ");  
 String email = sc.nextLine();  
 System.*out*.print("Enter your password: ");  
 String password = sc.nextLine();  
 *register*(name, email, password);  
  
 *// Login* System.*out*.print("\nEnter username to login: ");  
 String uname = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String pass = sc.nextLine();  
 if (!*login*(uname, pass)) {  
 System.*out*.println("Exiting... Login failed.");  
 return;  
 }  
  
 *// Browse food* System.*out*.print("\nSearch for a food item: ");  
 String food = sc.nextLine();  
 *searchFood*(food);  
  
 *// Add to cart* System.*out*.print("Enter quantity of " + food + " to add to cart: ");  
 int qty = sc.nextInt();  
 sc.nextLine(); *// clear buffer  
 addToCart*(food, qty);  
  
 *// Place order* System.*out*.print("\nEnter delivery address: ");  
 String address = sc.nextLine();  
 *placeOrder*(address);  
  
 *// Payment* System.*out*.print("Enter 16-digit card number: ");  
 String cardNo = sc.nextLine();  
 System.*out*.print("Enter expiry date (MM/YY): ");  
 String exp = sc.nextLine();  
 System.*out*.print("Enter 3-digit CVV: ");  
 String cvv = sc.nextLine();  
 if (!*payment*(cardNo, exp, cvv)) {  
 System.*out*.println("❌ Payment failed.");  
 return;  
 }  
  
 *// Track order  
 trackOrder*(101);  
  
 sc.close();  
 }  
}

7.Case Study on Food Delivery Applications (Negative Testing): Conduct a case study to analyze the requirements of food delivery applications and create negative test scenarios and test cases for the application.  
  
import java.util.\*;  
  
class FoodDeliveryNegativeTests {  
 private static Map<String, String> *users* = new HashMap<>();  
 private static List<String> *menu* = Arrays.*asList*("Pizza", "Burger", "Pasta", "Sandwich");  
  
 static {  
 *// Predefined user  
 users*.put("John", "Pass@123");  
 }  
  
 *// Registration* public static boolean register(String name, String email, String password) {  
 if (name.isEmpty() || email.isEmpty() || password.isEmpty()) {  
 System.*out*.println("❌ Error: Name, Email, and Password cannot be empty.");  
 return false;  
 }  
 if (*users*.containsKey(name)) {  
 System.*out*.println("❌ Error: User already exists.");  
 return false;  
 }  
 *users*.put(name, password);  
 System.*out*.println("✅ Registration successful for " + name);  
 return true;  
 }  
  
 *// Login* public static boolean login(String username, String password) {  
 if (!*users*.containsKey(username) || !*users*.get(username).equals(password)) {  
 System.*out*.println("❌ Error: Invalid username or password.");  
 return false;  
 }  
 System.*out*.println("✅ Login successful for " + username);  
 return true;  
 }  
  
 *// Browse menu* public static void searchFood(String item) {  
 if (!*menu*.contains(item)) {  
 System.*out*.println("❌ Error: Item not found in menu.");  
 } else {  
 System.*out*.println("✅ " + item + " is available.");  
 }  
 }  
  
 *// Add to cart* public static boolean addToCart(String item, int qty) {  
 if (!*menu*.contains(item)) {  
 System.*out*.println("❌ Error: Item not in menu.");  
 return false;  
 }  
 if (qty <= 0) {  
 System.*out*.println("❌ Error: Quantity must be positive.");  
 return false;  
 }  
 System.*out*.println("✅ Added " + qty + " x " + item + " to cart.");  
 return true;  
 }  
  
 *// Place order* public static boolean placeOrder(String address) {  
 if (address.isEmpty()) {  
 System.*out*.println("❌ Error: Address required.");  
 return false;  
 }  
 System.*out*.println("✅ Order placed successfully. Delivery to: " + address);  
 return true;  
 }  
  
 *// Payment* public static boolean payment(String cardNo, String expDate, String cvv) {  
 if (cardNo.length() != 16) {  
 System.*out*.println("❌ Error: Invalid card number.");  
 return false;  
 }  
 if (cvv.length() != 3) {  
 System.*out*.println("❌ Error: Invalid CVV.");  
 return false;  
 }  
 System.*out*.println("✅ Payment successful with card ending " + cardNo.substring(12));  
 return true;  
 }  
  
 *// Delivery tracking* public static void trackOrder(int orderId) {  
 if (orderId != 101) {  
 System.*out*.println("❌ Error: Order not found.");  
 } else {  
 System.*out*.println("✅ Order #" + orderId + " is confirmed.");  
 System.*out*.println("✅ Order #" + orderId + " is out for delivery.");  
 System.*out*.println("✅ Order #" + orderId + " has been delivered.");  
 }  
 }  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
  
 System.*out*.println("=== Food Delivery Negative Test ===");  
  
 *// Registration* System.*out*.print("Enter name for registration: ");  
 String name = sc.nextLine();  
 System.*out*.print("Enter email: ");  
 String email = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String password = sc.nextLine();  
 *register*(name, email, password);  
  
 *// Login* System.*out*.print("\nEnter username to login: ");  
 String uname = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String pass = sc.nextLine();  
 *login*(uname, pass);  
  
 *// Food search* System.*out*.print("\nSearch for a food item: ");  
 String food = sc.nextLine();  
 *searchFood*(food);  
  
 *// Add to cart* System.*out*.print("Enter quantity of " + food + " to add to cart: ");  
 int qty = sc.nextInt();  
 sc.nextLine(); *// clear buffer  
 addToCart*(food, qty);  
  
 *// Place order* System.*out*.print("\nEnter delivery address: ");  
 String address = sc.nextLine();  
 *placeOrder*(address);  
  
 *// Payment* System.*out*.print("Enter 16-digit card number: ");  
 String cardNo = sc.nextLine();  
 System.*out*.print("Enter expiry date (MM/YY): ");  
 String exp = sc.nextLine();  
 System.*out*.print("Enter 3-digit CVV: ");  
 String cvv = sc.nextLine();  
 *payment*(cardNo, exp, cvv);  
  
 *// Track order* System.*out*.print("\nEnter order ID to track: ");  
 int orderId = sc.nextInt();  
 *trackOrder*(orderId);  
  
 sc.close();  
 }  
}

8.Case Study on Banking Applications (Positive Testing): Conduct a case study to analyze the requirements of banking applications and develop positive test scenarios and test cases for the application.  
  
import java.util.Scanner;  
  
public class BankingPositiveTests {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
  
 *// Step 1: Login* System.*out*.println("=== Banking Application (Positive Testing) ===");  
 System.*out*.print("Enter Username: ");  
 String username = sc.nextLine();  
 System.*out*.print("Enter Password: ");  
 String password = sc.nextLine();  
  
 *// Any username & password is accepted (Positive Test)* if (username.isEmpty() || password.isEmpty()) {  
 System.*out*.println("❌ Error: Username/Password cannot be empty.");  
 return;  
 }  
  
 System.*out*.println("✅ Login Successful! Welcome, " + username);  
  
 *// Step 2: Banking operations* double balance = 1000.0; *// Initial balance* int choice;  
  
 do {  
 System.*out*.println("\n--- Banking Menu ---");  
 System.*out*.println("1. Check Balance");  
 System.*out*.println("2. Deposit Money");  
 System.*out*.println("3. Withdraw Money");  
 System.*out*.println("4. Exit");  
 System.*out*.print("Enter your choice: ");  
 choice = sc.nextInt();  
  
 switch (choice) {  
 case 1:  
 *// Positive test: Always display correct balance* System.*out*.println("💰 Your current balance is: $" + balance);  
 break;  
  
 case 2:  
 *// Positive test: Deposit valid amount* System.*out*.print("Enter amount to deposit: ");  
 double deposit = sc.nextDouble();  
 if (deposit > 0) {  
 balance += deposit;  
 System.*out*.println("✅ Successfully deposited $" + deposit);  
 System.*out*.println("Updated Balance: $" + balance);  
 } else {  
 System.*out*.println("❌ Invalid deposit amount!");  
 }  
 break;  
  
 case 3:  
 *// Positive test: Withdraw valid amount within balance* System.*out*.print("Enter amount to withdraw: ");  
 double withdraw = sc.nextDouble();  
 if (withdraw > 0 && withdraw <= balance) {  
 balance -= withdraw;  
 System.*out*.println("✅ Successfully withdrew $" + withdraw);  
 System.*out*.println("Updated Balance: $" + balance);  
 } else {  
 System.*out*.println("❌ Invalid withdrawal amount!");  
 }  
 break;  
  
 case 4:  
 System.*out*.println("👋 Thank you for using our Banking Application!");  
 break;  
  
 default:  
 System.*out*.println("❌ Invalid choice. Please try again.");  
 }  
 } while (choice != 4);  
  
 sc.close();  
 }  
}

9.Case Study on Banking Applications (Negative Testing): Conduct a case study to analyze the requirements of banking applications and develop negative test scenarios and test cases for the application.  
  
import java.util.Scanner;  
  
public class BankingNegativeTests {  
 private String username;  
 private String password;  
 private double balance;  
  
 *// Constructor* public BankingNegativeTests(String username, String password, double balance) {  
 this.username = username;  
 this.password = password;  
 this.balance = balance;  
 }  
  
 *// Login Validation* public boolean login(String enteredUser, String enteredPass) {  
 if (enteredUser.isEmpty() || enteredPass.isEmpty()) {  
 System.*out*.println("❌ Error: Username/Password cannot be empty.");  
 return false;  
 }  
 if (!enteredUser.equals(username) || !enteredPass.equals(password)) {  
 System.*out*.println("❌ Error: Invalid username or password.");  
 return false;  
 }  
 return true;  
 }  
  
 *// Withdraw method* public void withdraw(double amount) {  
 if (amount <= 0) {  
 System.*out*.println("❌ Error: Withdrawal amount must be greater than 0.");  
 } else if (amount > balance) {  
 System.*out*.println("❌ Error: Insufficient balance. Current balance: " + balance);  
 } else {  
 balance -= amount;  
 System.*out*.println("✅ Withdrawal successful. New Balance: " + balance);  
 }  
 }  
  
 *// Deposit method* public void deposit(double amount) {  
 if (amount <= 0) {  
 System.*out*.println("❌ Error: Deposit amount must be greater than 0.");  
 } else {  
 balance += amount;  
 System.*out*.println("✅ Deposit successful. New Balance: " + balance);  
 }  
 }  
  
 *// Main method* public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
  
 *// User sets custom username and password* System.*out*.print("Set your username: ");  
 String user = sc.nextLine();  
 System.*out*.print("Set your password: ");  
 String pass = sc.nextLine();  
  
 BankingNegativeTests bank = new BankingNegativeTests(user, pass, 1000.0);  
  
 System.*out*.println("\n=== LOGIN ===");  
 System.*out*.print("Enter username: ");  
 String enteredUser = sc.nextLine();  
 System.*out*.print("Enter password: ");  
 String enteredPass = sc.nextLine();  
  
 if (bank.login(enteredUser, enteredPass)) {  
 System.*out*.println("\n=== Banking Operations ===");  
 System.*out*.print("Enter withdrawal amount: ");  
 double w = sc.nextDouble();  
 bank.withdraw(w);  
  
 System.*out*.print("Enter deposit amount: ");  
 double d = sc.nextDouble();  
 bank.deposit(d);  
 }  
  
 sc.close();  
 }  
}

10.Develop a comprehensive test plan document for a chosen application, such as a Library Management System, detailing the testing strategies, methodologies, and criteria to ensure a thorough evaluation and validation of the system's functionality  
  
import java.util.\*;  
  
class LibraryManagement {  
 private Map<String, String> users = new HashMap<>();  
 private Map<String, Boolean> books = new HashMap<>();  
  
 public LibraryManagement() {  
 *// Predefined users* users.put("user1", "pass123");  
 users.put("admin", "admin123");  
  
 *// Predefined books* books.put("Java Programming", true);  
 books.put("C++ Basics", true);  
 }  
  
 public boolean login(String username, String password) {  
 return users.containsKey(username) && users.get(username).equals(password);  
 }  
  
 public String searchBook(String title) {  
 if (books.containsKey(title)) {  
 return books.get(title) ? "Book Available" : "Book Not Available";  
 }  
 return "Book Not Found";  
 }  
  
 public String issueBook(String title) {  
 if (books.containsKey(title) && books.get(title)) {  
 books.put(title, false);  
 return "Book Issued Successfully!";  
 }  
 return "Book Not Available!";  
 }  
  
 public String returnBook(String title, int lateDays) {  
 if (books.containsKey(title)) {  
 books.put(title, true);  
 if (lateDays > 0) {  
 int fine = lateDays \* 10; *// ₹10 per day* return "Book Returned! Fine: ₹" + fine;  
 }  
 return "Book Returned Successfully!";  
 }  
 return "Invalid Book!";  
 }  
}  
  
public class LibraryPositiveTests {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 LibraryManagement lms = new LibraryManagement();  
  
 System.*out*.println("Enter username:");  
 String user = sc.nextLine();  
 System.*out*.println("Enter password:");  
 String pass = sc.nextLine();  
  
 if (lms.login(user, pass)) {  
 System.*out*.println("✅ Login Successful!");  
  
 *// Positive test cases* System.*out*.println("Search Result: " + lms.searchBook("Java ramming"));  
 System.*out*.println(lms.issueBook("Java Programming"));  
 System.*out*.println(lms.returnBook("Java Programming", 0));  
 System.*out*.println(lms.returnBook("Java Programming", 3)); *// late return* } else {  
 System.*out*.println("❌ Login Failed!");  
 }  
 sc.close();  
 }  
}