```java
//Q1.
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class Administrator implements Staff {
    private final String adminId;
    private final List<String> accessPermissions;

    public Administrator(String adminId, List<String> accessPermissions) {
        this.adminId = adminId;
        this.accessPermissions = new ArrayList<>(accessPermissions);
    }

    public String getAdminId() {
        return adminId;
    }

    public List<String> getAccessPermissions() {
        return Collections.unmodifiableList(accessPermissions);
    }

    @Override
    public String getStaffId() {
        return "Admin-" + adminId;
    }

    @Override
    public String getRole() {
        return "Administrator";
    }

    @Override
    public String toString() {
        return "Administrator{" +
                "adminId='" + adminId + "'" +
                '}';
    }
}
```

```java
import java.util.Set;
import java.util.HashSet;
import java.util.Collections;

public class Doctor implements Staff {
    private final String licenseNumber;
    private final String specialty;
    private final Set<String> certifications;

    public Doctor(String licenseNumber, String specialty, Set<String>
certifications) {
        this.licenseNumber = licenseNumber;
        this.specialty = specialty;
        this.certifications = new HashSet<>(certifications);
    }

    public String getLicenseNumber() {
        return licenseNumber;
    }

    public String getSpecialty() {
        return specialty;
    }

    public Set<String> getCertifications() {
        return Collections.unmodifiableSet(certifications);
    }

    @Override
    public String getStaffId() {
        return "Dr-" + licenseNumber;
    }

    @Override
    public String getRole() {
        return "Doctor";
    }

    @Override
    public String toString() {
        return "Doctor{" +
                "licenseNumber='" + licenseNumber + "'" +
                ", specialty='" + specialty + "'" +
                '}';
    }
}
```

```java
import java.util.HashMap;
import java.util.Map;

public class HospitalSystem {

    // Static final constants for policies
    public static final String HOSPITAL_NAME = "General Hospital";
    public static final String PRIVACY_POLICY_URL =
"http://example.com/privacy";

    private final Map<String, Patient> patientRegistry = new HashMap<>();

    public boolean admitPatient(Patient patient, Staff staff) {
        if (validateStaffAccess(staff, patient)) {
            patientRegistry.put(patient.getPatientId(), patient);
            System.out.println("Patient " + patient.getCurrentName() + "
admitted successfully by " + staff.getRole() + " " + staff.getStaffId());
            return true;
        } else {
            System.out.println("Admission failed: Staff " +
staff.getRole() + " " + staff.getStaffId() + " does not have
permission.");
            return false;
        }
    }

    private boolean validateStaffAccess(Object staff, Object patient) {
        // Doctors and Administrators can admit any patient.
        // Nurses can only admit patients if they are assigned a room
(simulating a workflow).
```

```
        if (staff instanceof Doctor || staff instanceof Administrator) {
            return true;
        }
        if (staff instanceof Nurse && patient instanceof Patient) {
            Patient p = (Patient) patient;
            // Simplified logic: A nurse can handle admission if a room is
being assigned.
            if (p.getRoomNumber() > 0) {
                return true;
            }
        }
        return false;
    }

    // Package-private method for internal operations
    void dischargePatient(String patientId) {
        if (patientRegistry.containsKey(patientId)) {
            Patient p = patientRegistry.remove(patientId);
            System.out.println("Internal operation: Patient " +
p.getCurrentName() + " discharged.");
        }
    }

    public Patient findPatient(String patientId) {
        return patientRegistry.get(patientId);
    }

    // Example of role-based data access
    public String getPatientInfo(String patientId, Staff staff) {
        Patient patient = patientRegistry.get(patientId);
        if (patient == null) {
            return "Patient not found.";
        }

        if (staff instanceof Doctor) {
            // Doctors get full details
            return patient.getFullDetails(staff);
        } else if (staff instanceof Nurse || staff instanceof
Administrator) {
            // Nurses and Admins get basic info
```

```java
            return patient.getBasicInfo();
        } else {
            // Default to public info for others
            return patient.getPublicInfo();
        }
    }
}
```

```java
import java.time.LocalDate;
import java.util.Arrays;
import java.util.Objects;

public final class MedicalRecord {

    private final String recordId;
    private final String patientDNA;
    private final String[] allergies;
    private final String[] medicalHistory;
    private final LocalDate birthDate;
    private final String bloodType;

    public MedicalRecord(String recordId, String patientDNA, String[]
allergies, String[] medicalHistory, LocalDate birthDate, String bloodType)
{
        // HIPAA compliance validation
        Objects.requireNonNull(recordId, "Record ID cannot be null");
        Objects.requireNonNull(patientDNA, "Patient DNA cannot be null");
        Objects.requireNonNull(allergies, "Allergies array cannot be
null");
        Objects.requireNonNull(medicalHistory, "Medical history array
cannot be null");
        Objects.requireNonNull(birthDate, "Birth date cannot be null");
        Objects.requireNonNull(bloodType, "Blood type cannot be null");

        if (recordId.trim().isEmpty() || patientDNA.trim().isEmpty() ||
bloodType.trim().isEmpty()) {
            throw new IllegalArgumentException("IDs, DNA, and blood type
cannot be empty.");
```

```java
        }

        this.recordId = recordId;
        this.patientDNA = patientDNA;
        this.allergies = Arrays.copyOf(allergies, allergies.length); //
Defensive copy
        this.medicalHistory = Arrays.copyOf(medicalHistory,
medicalHistory.length); // Defensive copy
        this.birthDate = birthDate;
        this.bloodType = bloodType;
    }

    // Getters with defensive copying for mutable arrays
    public String getRecordId() {
        return recordId;
    }

    public String getPatientDNA() {
        return patientDNA;
    }

    public String[] getAllergies() {
        return Arrays.copyOf(allergies, allergies.length);
    }

    public String[] getMedicalHistory() {
        return Arrays.copyOf(medicalHistory, medicalHistory.length);
    }

    public LocalDate getBirthDate() {
        return birthDate;
    }

    public String getBloodType() {
        return bloodType;
    }

    public final boolean isAllergicTo(String substance) {
        if (substance == null || substance.trim().isEmpty()) {
            return false;
        }
        for (String allergy : this.allergies) {
            if (allergy.equalsIgnoreCase(substance)) {
                return true;
            }
        }
        return false;
    }

    @Override
    public String toString() {
        return "MedicalRecord{" +
                "recordId='" + recordId + "'" +
                ", birthDate=" + birthDate +
                ", bloodType='" + bloodType + "'" +
                ", allergies=" + Arrays.toString(allergies) +
                ", medicalHistory=" + Arrays.toString(medicalHistory) +
                '}';
    }
}
```

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class Nurse implements Staff {
    private final String nurseId;
    private String shift;
    private final List<String> qualifications;

    public Nurse(String nurseId, String shift, List<String>
qualifications) {
        this.nurseId = nurseId;
        this.shift = shift;
        this.qualifications = new ArrayList<>(qualifications);
    }

    public String getNurseId() {
        return nurseId;
    }

    public String getShift() {
        return shift;
    }

    public void setShift(String shift) {
        this.shift = shift;
    }

    public List<String> getQualifications() {
        return Collections.unmodifiableList(qualifications);
    }

    @Override
    public String getStaffId() {
        return "Nurse-" + nurseId;
    }

    @Override
    public String getRole() {
        return "Nurse";
    }

    @Override
    public String toString() {
        return "Nurse{" +
                "nurseId='" + nurseId + "'" +
                ", shift='" + shift + "'" +
                '}';
    }
}
```

```
import java.util.Objects;
import java.util.UUID;
import java.time.LocalDate;
```

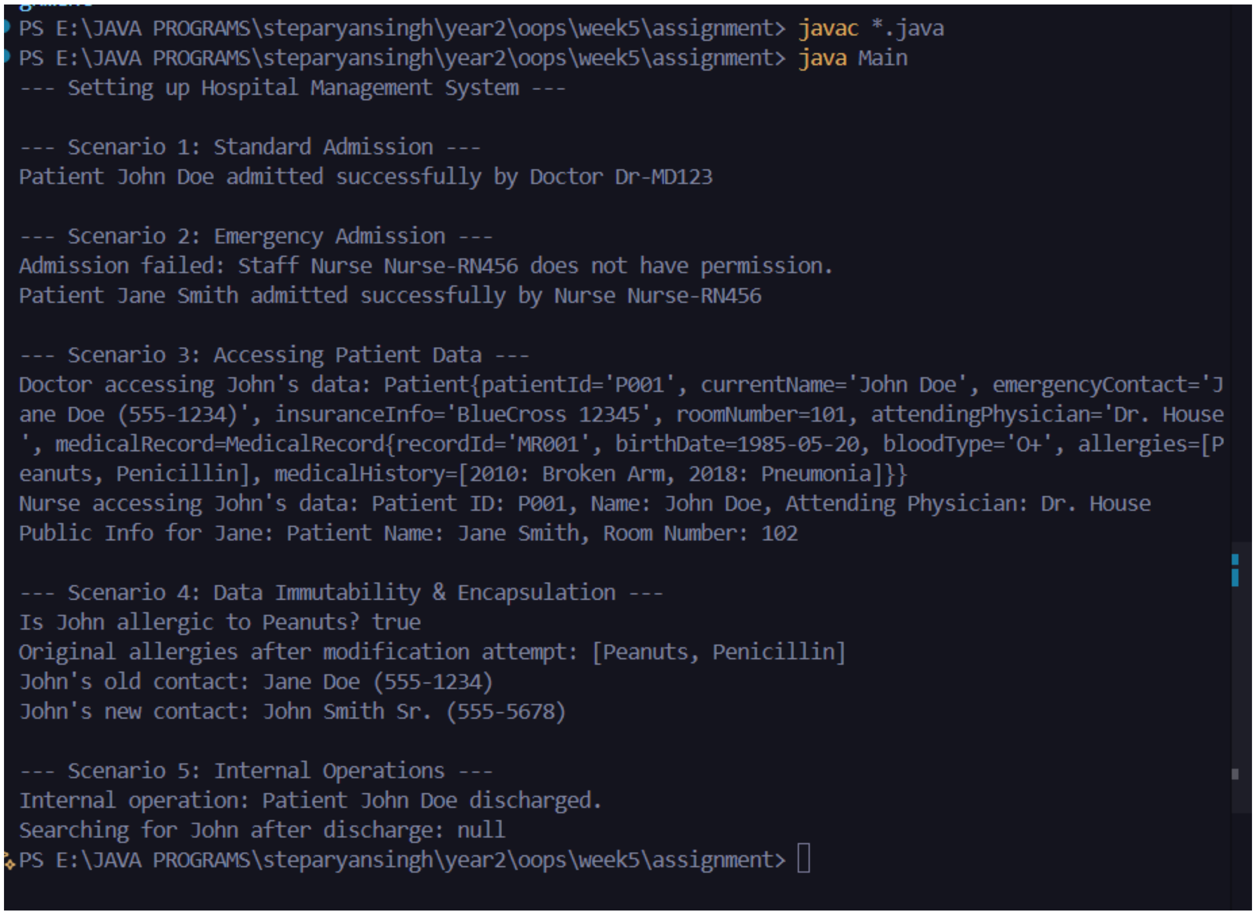
```
public class Patient {

    // Protected health information (PHI)
    private final String patientId;
    private final MedicalRecord medicalRecord;

    // Modifiable personal data
    private String currentName;
    private String emergencyContact;
    private String insuranceInfo;

    // Current treatment info
    private int roomNumber;
    private String attendingPhysician;

    // Constructor for Standard Admission (full info)
    public Patient(String patientId, String currentName, String
emergencyContact, String insuranceInfo, int roomNumber, String
attendingPhysician, MedicalRecord medicalRecord) {
        // Privacy and data integrity validation
        Objects.requireNonNull(patientId, "Patient ID cannot be null");
        Objects.requireNonNull(currentName, "Patient name cannot be
null");
        Objects.requireNonNull(medicalRecord, "Medical record cannot be
null");

        this.patientId = patientId;
        this.currentName = currentName;
        this.emergencyContact = emergencyContact;
        this.insuranceInfo = insuranceInfo;
        this.roomNumber = roomNumber;
        this.attendingPhysician = attendingPhysician;
        this.medicalRecord = medicalRecord;
    }

    // Constructor for Emergency Admission (minimal data)
    public Patient(String currentName) {
        this("TEMP-" + UUID.randomUUID().toString(), currentName, "N/A",
"N/A", -1, "On-call",
            new MedicalRecord("TEMP-REC-" + UUID.randomUUID().toString(),
"UNKNOWN", new String[]{}, new String[]{}, LocalDate.now(), "UNKNOWN"));
    }

    // Constructor for Transfer Admission (imports existing record)
    public Patient(String patientId, String currentName, String
emergencyContact, String insuranceInfo, MedicalRecord medicalRecord) {
        this(patientId, currentName, emergencyContact, insuranceInfo, -1,
"TBD", medicalRecord);
    }

    // --- Getters and Setters (JavaBean style) ---

    // Publicly accessible, non-sensitive info
    public String getPublicInfo() {
        return "Patient Name: " + currentName + ", Room Number: " +
(roomNumber > 0 ? roomNumber : "N/A");
    }

    // Package-private for internal hospital staff
    String getBasicInfo() {
        return "Patient ID: " + patientId + ", Name: " + currentName + ",
Attending Physician: " + attendingPhysician;
    }

    // Full access for authorized personnel (e.g., Doctor)
    public String getFullDetails(Staff staff) {
        if (staff instanceof Doctor) {
            return toString();
        }
        return "Access Denied: Insufficient privileges.";
    }


    public String getPatientId() {
        return patientId;
    }

    public MedicalRecord getMedicalRecord() {
        // The record itself is immutable, so no defensive copy needed.
        return medicalRecord;
    }

    public String getCurrentName() {
        return currentName;
    }

    public void setCurrentName(String currentName) {
        // Validation
        Objects.requireNonNull(currentName, "Patient name cannot be
null");
        this.currentName = currentName;
    }

    public String getEmergencyContact() {
        return emergencyContact;
    }

    public void setEmergencyContact(String emergencyContact) {
        this.emergencyContact = emergencyContact;
    }

    public String getInsuranceInfo() {
        return insuranceInfo;
    }

    public void setInsuranceInfo(String insuranceInfo) {
        this.insuranceInfo = insuranceInfo;
    }

    public int getRoomNumber() {
        return roomNumber;
    }

    public void setRoomNumber(int roomNumber) {
        this.roomNumber = roomNumber;
    }

    public String getAttendingPhysician() {
        return attendingPhysician;
```

```java
    }

    public void setAttendingPhysician(String attendingPhysician) {
        this.attendingPhysician = attendingPhysician;
    }

    // Audit trail via toString()
    @Override
    public String toString() {
        return "Patient{" +
                "patientId='" + patientId + "'" +
                ", currentName='" + currentName + "'" +
                ", emergencyContact='" + emergencyContact + "'" +
                ", insuranceInfo='" + insuranceInfo + "'" +
                ", roomNumber=" + roomNumber +
                ", attendingPhysician='" + attendingPhysician + "'" +
                ", medicalRecord=" + medicalRecord.toString() +
                '}';
    }
}
```

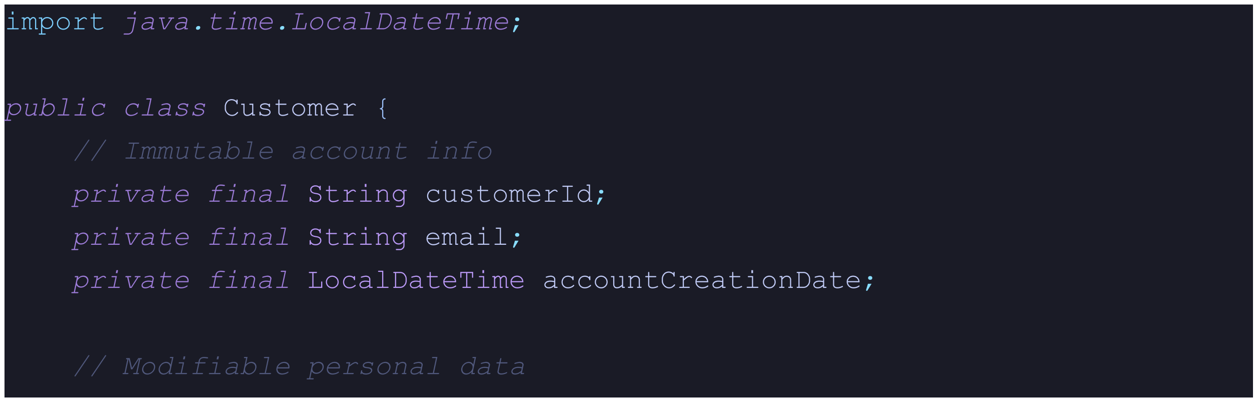
```java
import java.time.LocalDate;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;

public class Main {
    public static void main(String[] args) {
        System.out.println("--- Setting up Hospital Management System
---");
        HospitalSystem hospital = new HospitalSystem();

        // Create Staff
        Doctor drHouse = new Doctor("MD123", "Nephrology", new
HashSet<>(Arrays.asList("Board Certified", "PhD")));
        Nurse nurseJackie = new Nurse("RN456", "Day",
Collections.singletonList("Certified Nurse Anesthetist"));
```

```
        Administrator adminCuddy = new Administrator("ADM789",
Collections.singletonList("Full Access"));

        System.out.println("\n--- Scenario 1: Standard Admission ---");
        MedicalRecord recordJohn = new MedicalRecord(
            "MR001",
            "AGCT...",
            new String[]{"Peanuts", "Penicillin"},
            new String[]{"2010: Broken Arm", "2018: Pneumonia"},
            LocalDate.of(1985, 5, 20),
            "O+"
        );
        Patient patientJohn = new Patient(
            "P001",
            "John Doe",
            "Jane Doe (555-1234)",
            "BlueCross 12345",
            101,
            "Dr. House",
            recordJohn
        );
        hospital.admitPatient(patientJohn, drHouse); // Doctor admits

        System.out.println("\n--- Scenario 2: Emergency Admission ---");
        Patient patientJane = new Patient("Jane Smith"); // Minimal info
        hospital.admitPatient(patientJane, nurseJackie); // Nurse cannot
admit without a room
        patientJane.setRoomNumber(102); // Assign a room
        hospital.admitPatient(patientJane, nurseJackie); // Now nurse can
admit

        System.out.println("\n--- Scenario 3: Accessing Patient Data
---");
        System.out.println("Doctor accessing John's data: " +
hospital.getPatientInfo("P001", drHouse));
        System.out.println("Nurse accessing John's data: " +
hospital.getPatientInfo("P001", nurseJackie));
        System.out.println("Public Info for Jane: " +
patientJane.getPublicInfo());
```

```
        System.out.println("\n--- Scenario 4: Data Immutability &
Encapsulation ---");
        Patient retrievedJohn = hospital.findPatient("P001");
        if (retrievedJohn != null) {
            System.out.println("Is John allergic to Peanuts? " +
retrievedJohn.getMedicalRecord().isAllergicTo("Peanuts"));

            // Try to modify immutable data (will fail to compile if
setters existed)
            // retrievedJohn.getMedicalRecord().setBloodType("A-"); //
This would be a compilation error

            // Try to modify the returned array
            String[] allergies =
retrievedJohn.getMedicalRecord().getAllergies();
            allergies[0] = "Shellfish"; // Modify the copy

            // Verify original data is unchanged
            System.out.println("Original allergies after modification
attempt: " +
Arrays.toString(retrievedJohn.getMedicalRecord().getAllergies()));

            // Modify mutable data via setter
            System.out.println("John's old contact: " +
retrievedJohn.getEmergencyContact());
            retrievedJohn.setEmergencyContact("John Smith Sr.
(555-5678)");
            System.out.println("John's new contact: " +
retrievedJohn.getEmergencyContact());
        }

        System.out.println("\n--- Scenario 5: Internal Operations ---");
        hospital.dischargePatient("P001");
        System.out.println("Searching for John after discharge: " +
hospital.findPatient("P001"));
    }
}
```



```
PS E:\JAVA PROGRAMS\stepAryansingh\year2\oops\week5\assignment> javac *.java
PS E:\JAVA PROGRAMS\stepAryansingh\year2\oops\week5\assignment> java Main
--- Setting up Hospital Management System ---

--- Scenario 1: Standard Admission ---
Patient John Doe admitted successfully by Doctor Dr-MD123

--- Scenario 2: Emergency Admission ---
Admission failed: Staff Nurse Nurse-RN456 does not have permission.
Patient Jane Smith admitted successfully by Nurse Nurse-RN456

--- Scenario 3: Accessing Patient Data ---
Doctor accessing John's data: Patient{patientId='P001', currentName='John Doe', emergencyContact='J
ane Doe (555-1234)', insuranceInfo='BlueCross 12345', roomNumber=101, attendingPhysician='Dr. House
', medicalRecord=MedicalRecord{recordId='MR001', birthDate=1985-05-20, bloodType='O+', allergies=[P
eanuts, Penicillin], medicalHistory=[2010: Broken Arm, 2018: Pneumonia]}}
Nurse accessing John's data: Patient ID: P001, Name: John Doe, Attending Physician: Dr. House
Public Info for Jane: Patient Name: Jane Smith, Room Number: 102

--- Scenario 4: Data Immutability & Encapsulation ---
Is John allergic to Peanuts? true
Original allergies after modification attempt: [Peanuts, Penicillin]
John's old contact: Jane Doe (555-1234)
John's new contact: John Smith Sr. (555-5678)

--- Scenario 5: Internal Operations ---
Internal operation: Patient John Doe discharged.
Searching for John after discharge: null
PS E:\JAVA PROGRAMS\stepAryansingh\year2\oops\week5\assignment>
```



```java
import java.time.LocalDateTime;

public class Customer {
    // Immutable account info
    private final String customerId;
    private final String email;
    private final LocalDateTime accountCreationDate;

    // Modifiable personal data
    private String name;
    private String phoneNumber;
    private String preferredLanguage;

    // Constructor for registered customers
    public Customer(String customerId, String email, String name, String
phoneNumber, String preferredLanguage) {
        this.customerId = customerId;
        this.email = email;
        this.name = name;
        this.phoneNumber = phoneNumber;
        this.preferredLanguage = preferredLanguage;
        this.accountCreationDate = LocalDateTime.now();
    }

    // Constructor for guest checkout
    public Customer(String email, String phoneNumber) {
        this("GUEST_" + System.currentTimeMillis(), email, "Guest",
phoneNumber, "en");
    }

    // Constructor for premium members
    public Customer(String customerId, String email, String name) {
        this(customerId, email, name, null, "en");
        // Special logic for premium members could be added here
    }

    // Constructor for corporate accounts
    public Customer(String customerId, String email, String name, String
companyName) {
        this(customerId, email, name, null, "en");
        // Validation for corporate accounts
        if (companyName == null || companyName.isEmpty()) {
            throw new IllegalArgumentException("Corporate accounts require
a company name.");
        }
    }

    // Getters for all fields
    public String getCustomerId() {
        return customerId;
    }

    public String getEmail() {
        return email;
    }

    public LocalDateTime getAccountCreationDate() {
        return accountCreationDate;
    }

    public String getName() {
        return name;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public String getPreferredLanguage() {
        return preferredLanguage;
    }

    // Setters for modifiable data
    public void setName(String name) {
        this.name = name;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public void setPreferredLanguage(String preferredLanguage) {
        this.preferredLanguage = preferredLanguage;
    }

    // Access control methods
    String getCreditRating() { // Package-private for internal use
        // Dummy logic for credit rating
        return "Good";
    }

    public String getPublicProfile() { // Public for safe information
        return "Customer: " + name + ", ID: " + customerId.substring(0, 5)
+ "...";
    }
}
```

```java
import java.util.HashMap;
import java.util.Map;

public final class ECommerceSystem {
    private static final Map<String, Product> productCatalog = new
HashMap<>();

    static {
        // Populate the product catalog
        productCatalog.put("101", Product.createElectronics("101",
"Laptop", "TechCorp", 1200.0, 3.5, new String[]{"16GB RAM"}, Map.of("CPU",
"Intel i7")));
        productCatalog.put("102", Product.createClothing("102", "T-Shirt",
"FashionWear", 25.0, 0.2, new String[]{"Cotton"}, Map.of()));
        productCatalog.put("103", Product.createBooks("103", "Java Guide",
"CodePub", 45.0, 1.0, new String[]{"Hardcover"}, Map.of("Author", "J.
Doe")));
    }

    public static boolean processOrder(Order order, Customer customer) {
        System.out.println("Processing order " + order.getOrderId() + "
for customer " + customer.getName());

        // Validate order and customer
        if (order.getCart().getTotalAmount() <= 0) {
            System.out.println("Order failed: Cart is empty or invalid.");
            return false;
        }

        // Get total amount and process payment
        PaymentProcessor paymentProcessor = new PaymentProcessor("P-1",
"SEC-KEY-123");
        boolean paymentSuccess =
paymentProcessor.processPayment(order.getCart().getTotalAmount());

        if (paymentSuccess) {
            System.out.println("Payment successful. Fulfilling order.");
            return fulfillOrder(order);
        } else {
            System.out.println("Order failed: Payment could not be
processed.");
            return false;
        }
    }

    // Static method for inventory management
    public static boolean checkInventory(String productId, int quantity) {
        Product product = productCatalog.get(productId);
        if (product != null) {
            System.out.println("Inventory check for " + product.getName()
+ ": available.");
            return true;
        }
        System.out.println("Inventory check failed: Product " + productId
+ " not found.");
        return false;
    }

    private static boolean fulfillOrder(Order order) {
        System.out.println("Order " + order.getOrderId() + " fulfilled.
Shipping items...");
        // Dummy shipping logic
        return true;
    }

    public static void main(String[] args) {
        // Scenario 1: Registered customer order
        Customer customer = new Customer("C-123", "jane.doe@example.com",
"Jane Doe", "555-1111", "en");
```

```java
        ShoppingCart cart = new ShoppingCart("CART-01",
customer.getCustomerId());

        Product laptop = productCatalog.get("101");
        cart.addItem(laptop, 1);

        Order order = new Order("ORD-001", cart,
customer.getCustomerId());
        ECommerceSystem.processOrder(order, customer);

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

        // Scenario 2: Guest checkout
        Customer guest = new Customer("guest@example.com", "555-2222");
        ShoppingCart guestCart = new ShoppingCart("G-CART-02",
guest.getCustomerId());
        Product tShirt = productCatalog.get("102");

        guestCart.addItem(tShirt, 3);

        Order guestOrder = new Order("ORD-002", guestCart,
guest.getCustomerId());
        ECommerceSystem.processOrder(guestOrder, guest);
    }
}
```

```java
import java.time.LocalDateTime;

public class Order {
    private final String orderId;
    private final LocalDateTime orderTime;
    private final ShoppingCart cart;
    private final String customerId;

    public Order(String orderId, ShoppingCart cart, String customerId) {
        this.orderId = orderId;
        this.orderTime = LocalDateTime.now();
        this.cart = cart;
        this.customerId = customerId;
    }

    public String getOrderId() {
        return orderId;
    }

    public LocalDateTime getOrderTime() {
        return orderTime;
    }

    public ShoppingCart getCart() {
        return cart;
    }

    public String getCustomerId() {
        return customerId;
    }
}
```

```java
public class PaymentProcessor {
    private final String processorId;
    private final String securityKey;

    public PaymentProcessor(String processorId, String securityKey) {
        this.processorId = processorId;
        this.securityKey = securityKey;
    }

    public boolean processPayment(double amount) {
        // Dummy payment processing logic
        System.out.println("Processing payment of $" +
String.format("%.2f", amount) + " with processor " + processorId);
        return amount > 0;
    }
}
```

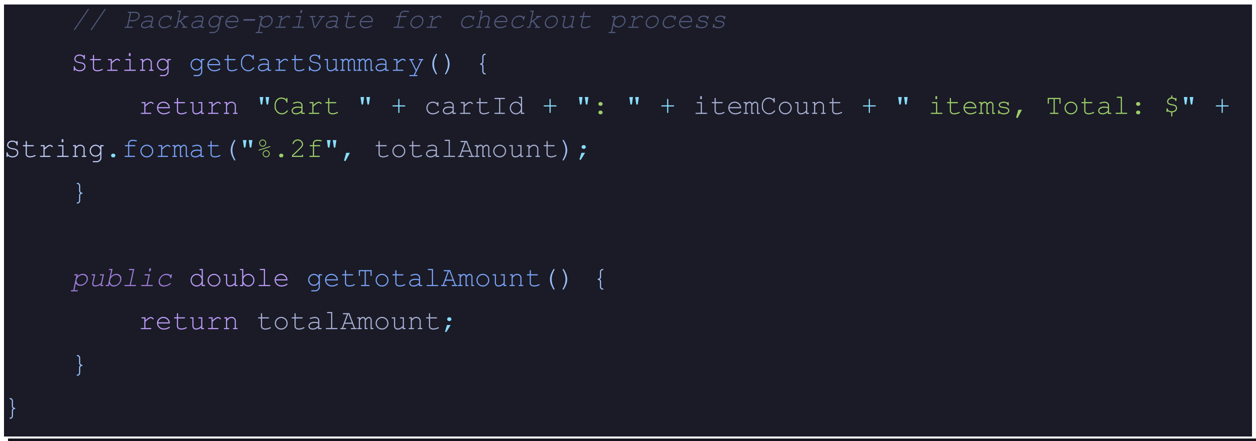
```java
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;

public final class Product {
    private final String productId;
    private final String name;
    private final String category;
    private final String manufacturer;
    private final double basePrice;
    private final double weight;
    private final String[] features;
    private final Map<String, String> specifications;

    private Product(String productId, String name, String category, String
manufacturer,
                    double basePrice, double weight, String[] features,
Map<String, String> specifications) {
        this.productId = productId;
        this.name = name;
        this.category = category;
        this.manufacturer = manufacturer;
        this.basePrice = basePrice;
        this.weight = weight;
        this.features = features != null ? Arrays.copyOf(features,
features.length) : new String[0];
        this.specifications = specifications != null ? new
HashMap<>(specifications) : Collections.emptyMap();
    }

    // Factory methods
    public static Product createElectronics(String id, String name, String
manufacturer, double price, double weight, String[] features, Map<String,
String> specs) {
        return new Product(id, name, "Electronics", manufacturer, price,
weight, features, specs);
    }
```

```java
    public static Product createClothing(String id, String name, String
manufacturer, double price, double weight, String[] features, Map<String,
String> specs) {
        return new Product(id, name, "Clothing", manufacturer, price,
weight, features, specs);
    }

    public static Product createBooks(String id, String name, String
manufacturer, double price, double weight, String[] features, Map<String,
String> specs) {
        return new Product(id, name, "Books", manufacturer, price, weight,
features, specs);
    }

    // Getters with defensive copying
    public String getProductId() {
        return productId;
    }

    public String getName() {
        return name;
    }

    public String getCategory() {
        return category;
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public double getBasePrice() {
        return basePrice;
    }

    public double getWeight() {
        return weight;
    }

    public String[] getFeatures() {
        return Arrays.copyOf(features, features.length);
    }

    public Map<String, String> getSpecifications() {
        return Collections.unmodifiableMap(new HashMap<>(specifications));
    }

    // Business rule consistency method
    public final double calculateTax(String region) {
        if ("US".equalsIgnoreCase(region)) {
            return basePrice * 0.08; // Example tax rate
        } else if ("EU".equalsIgnoreCase(region)) {
            return basePrice * 0.20;
        }
        return 0;
    }

    @Override
    public String toString() {
        return "Product{" +
                "productId='" + productId + '\'' +
                ", name='" + name + '\'' +
                ", category='" + category + '\'' +
                ", basePrice=" + basePrice +
                '}';
    }
}
```

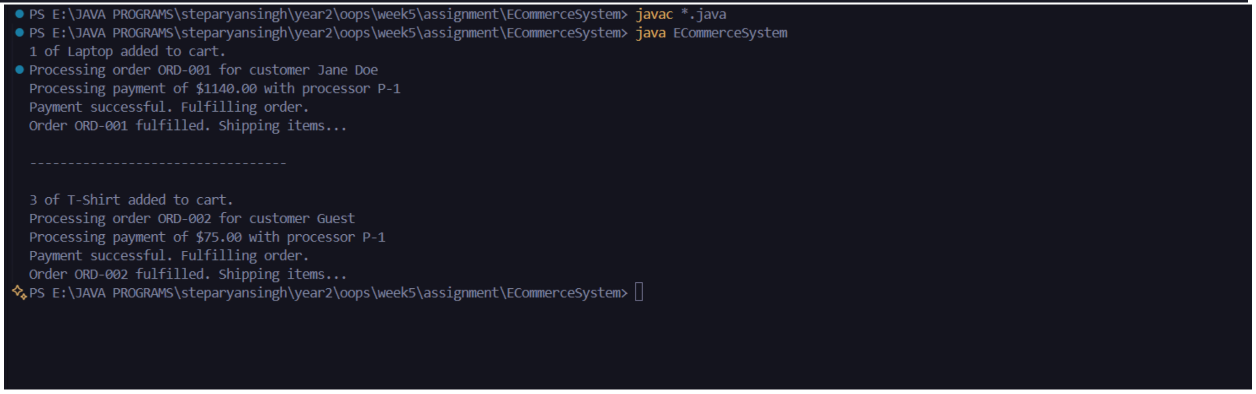
```java
import java.util.Map;

public class ShippingCalculator {
    private final Map<String, Double> shippingRates;

    public ShippingCalculator(Map<String, Double> rates) {
        this.shippingRates = rates;
    }

    public double calculateShippingCost(String region, double weight) {
        double rate = shippingRates.getOrDefault(region, 0.0);
        return rate * weight;
    }
}
```

```
import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {
    private final String cartId;
    private final String customerId;
    private final List<CartItem> items;
    private double totalAmount;
    private int itemCount;

    private static class CartItem {
        private final Product product;
        private final int quantity;

        public CartItem(Product product, int quantity) {
            this.product = product;
            this.quantity = quantity;
        }

        public Product getProduct() {
            return product;
        }

        public int getQuantity() {
            return quantity;
        }
    }

    public ShoppingCart(String cartId, String customerId) {
        this.cartId = cartId;
        this.customerId = customerId;
        this.items = new ArrayList<>();
        this.totalAmount = 0.0;
        this.itemCount = 0;
    }
```

```java
    public boolean addItem(Object product, int quantity) {
        if (!(product instanceof Product)) {
            System.out.println("Invalid item. Only products can be added to the cart.");
            return false;
        }

        Product p = (Product) product;
        if (quantity <= 0) {
            System.out.println("Quantity must be greater than zero.");
            return false;
        }

        items.add(new CartItem(p, quantity));
        updateCartTotals();
        System.out.println(quantity + " of " + p.getName() + " added to cart.");
        return true;
    }

    private void updateCartTotals() {
        this.totalAmount = 0.0;
        this.itemCount = 0;
        for (CartItem item : items) {
            this.totalAmount += item.getProduct().getBasePrice() * item.getQuantity();
            this.itemCount += item.getQuantity();
        }
        this.totalAmount -= calculateDiscount();
    }

    private double calculateDiscount() { // Internal pricing logic
        // Example: 5% discount on orders over $500
        if (totalAmount > 500) {
            return totalAmount * 0.05;
        }
        return 0;
    }
```



```java
    // Package-private for checkout process
    String getCartSummary() {
        return "Cart " + cartId + ": " + itemCount + " items, Total: $" +
String.format("%.2f", totalAmount);
    }

    public double getTotalAmount() {
        return totalAmount;
    }
}
```



```
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week5\assignment\ECommerceSystem> javac *.java
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week5\assignment\ECommerceSystem> java ECommerceSystem
1 of Laptop added to cart.
Processing order ORD-001 for customer Jane Doe
Processing payment of $1140.00 with processor P-1
Payment successful. Fulfilling order.
Order ORD-001 fulfilled. Shipping items...

----------------------------------

3 of T-Shirt added to cart.
Processing order ORD-002 for customer Guest
Processing payment of $75.00 with processor P-1
Payment successful. Fulfilling order.
Order ORD-002 fulfilled. Shipping items...
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week5\assignment\ECommerceSystem>
```