**WEEK 5 – HW**

Q1. Hospital Management System with Patient Privacy

Soln.

import java.time.LocalDate;

import java.util.\*;

public class HospitalSystem {

public static 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) {

if (recordId == null || recordId.isEmpty()) throw new IllegalArgumentException("Record ID required");

if (patientDNA == null || patientDNA.isEmpty()) throw new IllegalArgumentException("DNA required");

if (birthDate == null || birthDate.isAfter(LocalDate.now())) throw new IllegalArgumentException("Invalid birth date");

if (bloodType == null || bloodType.isEmpty()) throw new IllegalArgumentException("Blood type required");

this.recordId = recordId;

this.patientDNA = patientDNA;

this.allergies = Arrays.copyOf(allergies, allergies.length);

this.medicalHistory = Arrays.copyOf(medicalHistory, medicalHistory.length);

this.birthDate = birthDate;

this.bloodType = bloodType;

}

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) {

for (String a : allergies) if (a.equalsIgnoreCase(substance)) return true;

return false;

}

@Override

public String toString() {

return "MedicalRecord{recordId='" + recordId + "', birthDate=" + birthDate + ", bloodType=" + bloodType + "}";

}

@Override

public int hashCode() { return Objects.hash(recordId, patientDNA); }

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof MedicalRecord)) return false;

MedicalRecord m = (MedicalRecord) o;

return recordId.equals(m.recordId) && patientDNA.equals(m.patientDNA);

}

}

public static class Patient {

private final String patientId;

private final MedicalRecord medicalRecord;

private String currentName;

private String emergencyContact;

private String insuranceInfo;

private int roomNumber;

private String attendingPhysician;

Patient(String name) {

this(UUID.randomUUID().toString(), new MedicalRecord(UUID.randomUUID().toString(), "TEMP-DNA",

new String[]{}, new String[]{}, LocalDate.of(2000,1,1), "Unknown"), name, "Unknown", "Unknown", 0, "TBD");

}

Patient(String patientId, MedicalRecord record, String name, String emergencyContact, String insuranceInfo,

int roomNumber, String physician) {

if (record == null) throw new IllegalArgumentException("Medical record required");

this.patientId = patientId;

this.medicalRecord = record;

this.currentName = name;

this.emergencyContact = emergencyContact;

this.insuranceInfo = insuranceInfo;

this.roomNumber = roomNumber;

this.attendingPhysician = physician;

}

Patient(MedicalRecord record, String name) {

this(UUID.randomUUID().toString(), record, name, "Unknown", "Unknown", 0, "TBD");

}

public String getPatientId() { return patientId; }

public MedicalRecord getMedicalRecord() { return medicalRecord; }

public String getCurrentName() { return currentName; }

public void setCurrentName(String name) { if (name!=null&&!name.isEmpty()) this.currentName=name; }

public String getEmergencyContact() { return emergencyContact; }

public void setEmergencyContact(String c) { if (c!=null&&!c.isEmpty()) this.emergencyContact=c; }

public String getInsuranceInfo() { return insuranceInfo; }

public void setInsuranceInfo(String i) { if (i!=null&&!i.isEmpty()) this.insuranceInfo=i; }

public int getRoomNumber() { return roomNumber; }

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

public String getAttendingPhysician() { return attendingPhysician; }

public void setAttendingPhysician(String p) { if (p!=null&&!p.isEmpty()) this.attendingPhysician=p; }

String getBasicInfo() {

return "Patient[name="+currentName+", room="+roomNumber+", physician="+attendingPhysician+"]";

}

public String getPublicInfo() {

return "Patient[name="+currentName+", room="+roomNumber+"]";

}

@Override

public String toString() {

return "Patient{id="+patientId+", name="+currentName+", room="+roomNumber+"}";

}

}

public static class Doctor {

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 new HashSet<>(certifications);}

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

}

public static class Nurse {

private final String nurseId;

private final String shift;

private final List<String> qualifications;

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

public String getNurseId(){return nurseId;}

public String getShift(){return shift;}

public List<String> getQualifications(){return new ArrayList<>(qualifications);}

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

}

public static class Administrator {

private final String adminId;

private final List<String> accessPermissions;

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

public String getAdminId(){return adminId;}

public List<String> getAccessPermissions(){return new ArrayList<>(accessPermissions);}

@Override public String toString(){return "Administrator{id="+adminId+", perms="+accessPermissions+"}";}

}

public static class Hospital {

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

public static final String PRIVACY\_POLICY="HIPAA-COMPLIANT";

public boolean admitPatient(Object patient,Object staff){

if(!(patient instanceof Patient)) return false;

if(!validateStaffAccess(staff,patient)) return false;

patientRegistry.put(((Patient)patient).getPatientId(),patient);

return true;

}

private boolean validateStaffAccess(Object staff,Object patient){

if(staff instanceof Doctor) return true;

if(staff instanceof Nurse) return ((Nurse)staff).getQualifications().contains("ER");

if(staff instanceof Administrator) return ((Administrator)staff).getAccessPermissions().contains("ADMIT");

return false;

}

void printRegistry(){System.out.println("Registry: "+patientRegistry.values());}

}

public static void main(String[] args) {

MedicalRecord rec=new MedicalRecord("REC1","DNA-123",new String[]{"Peanuts"},new String[]{"Asthma"},LocalDate.of(1990,5,10),"O+");

Patient p1=new Patient("P1",rec,"Alice","12345","InsuranceA",101,"Dr. Smith");

Doctor d=new Doctor("LIC123","Cardiology",Set.of("BoardCertified"));

Nurse n=new Nurse("N1","Night",List.of("ER","ICU"));

Administrator a=new Administrator("A1",List.of("ADMIT","VIEW"));

Hospital h=new Hospital();

System.out.println("Doctor admit: "+h.admitPatient(p1,d));

System.out.println("Nurse admit: "+h.admitPatient(p1,n));

System.out.println("Admin admit: "+h.admitPatient(p1,a));

h.printRegistry();

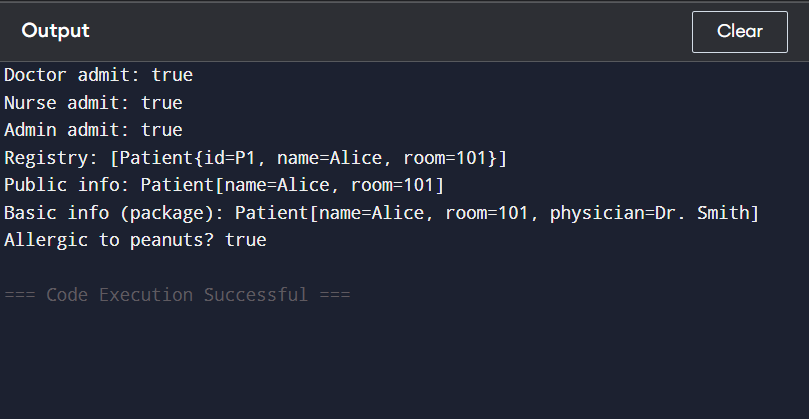
System.out.println("Public info: "+p1.getPublicInfo());

System.out.println("Basic info (package): "+p1.getBasicInfo());

System.out.println("Allergic to peanuts? "+rec.isAllergicTo("Peanuts"));

}

}



Q2. E-Commerce Order Processing with Immutable Products

Soln.

import java.time.LocalDateTime;

import java.util.\*;

public final class ECommerceSystem {

private static final Map<String,Object> productCatalog = new HashMap<>();

public static 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 id,String name,String category,String manufacturer,double basePrice,double weight,String[] features,Map<String,String> specs) {

if(id==null||id.isEmpty()) throw new IllegalArgumentException("ID required");

if(basePrice<0||weight<0) throw new IllegalArgumentException("Invalid values");

this.productId=id;

this.name=name;

this.category=category;

this.manufacturer=manufacturer;

this.basePrice=basePrice;

this.weight=weight;

this.features=Arrays.copyOf(features,features.length);

this.specifications=new HashMap<>(specs);

}

public static Product createElectronics(String name,String manufacturer,double basePrice,String[] features,Map<String,String> specs) {

return new Product(UUID.randomUUID().toString(),name,"Electronics",manufacturer,basePrice,1.0,features,specs);

}

public static Product createClothing(String name,String manufacturer,double basePrice,String[] features,Map<String,String> specs) {

return new Product(UUID.randomUUID().toString(),name,"Clothing",manufacturer,basePrice,0.5,features,specs);

}

public static Product createBooks(String name,String manufacturer,double basePrice,String[] features,Map<String,String> specs) {

return new Product(UUID.randomUUID().toString(),name,"Book",manufacturer,basePrice,0.3,features,specs);

}

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 new HashMap<>(specifications);}

public final double calculateTax(String region) {

if("US".equalsIgnoreCase(region)) return basePrice\*0.07;

if("EU".equalsIgnoreCase(region)) return basePrice\*0.2;

return basePrice\*0.1;

}

@Override public String toString(){return "Product{name="+name+", category="+category+", price="+basePrice+"}";}

}

public static class Customer {

private final String customerId;

private final String email;

private final String accountCreationDate;

private String name;

private String phoneNumber;

private String preferredLanguage;

public Customer(String email) {

this(UUID.randomUUID().toString(),email,"Guest", "Unknown","EN");

}

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

this(UUID.randomUUID().toString(),email,name,phone,"EN");

}

public Customer(String id,String email,String name,String phone,String lang) {

this.customerId=id;

this.email=email;

this.accountCreationDate=LocalDateTime.now().toString();

this.name=name;

this.phoneNumber=phone;

this.preferredLanguage=lang;

}

public String getCustomerId(){return customerId;}

public String getEmail(){return email;}

public String getAccountCreationDate(){return accountCreationDate;}

public String getName(){return name;}

public void setName(String n){if(n!=null&&!n.isEmpty())this.name=n;}

public String getPhoneNumber(){return phoneNumber;}

public void setPhoneNumber(String p){if(p!=null&&!p.isEmpty())this.phoneNumber=p;}

public String getPreferredLanguage(){return preferredLanguage;}

public void setPreferredLanguage(String l){if(l!=null&&!l.isEmpty())this.preferredLanguage=l;}

String getCreditRating(){return "Good";}

public String getPublicProfile(){return "Customer{name="+name+"}";}

@Override public String toString(){return "Customer{id="+customerId+", name="+name+"}";}

}

public static class ShoppingCart {

private final String cartId;

private final String customerId;

private final List<Object> items;

private double totalAmount;

private int itemCount;

public ShoppingCart(String customerId) {

this.cartId=UUID.randomUUID().toString();

this.customerId=customerId;

this.items=new ArrayList<>();

this.totalAmount=0;

this.itemCount=0;

}

public boolean addItem(Object product,int quantity) {

if(!(product instanceof Product)) return false;

for(int i=0;i<quantity;i++) items.add(product);

itemCount+=quantity;

totalAmount+=((Product)product).getBasePrice()\*quantity;

return true;

}

private double calculateDiscount() {

if(itemCount>5) return totalAmount\*0.1;

return 0;

}

String getCartSummary() {

double discount=calculateDiscount();

return "Cart{items="+itemCount+", total="+totalAmount+", discount="+discount+"}";

}

@Override public String toString(){return getCartSummary();}

}

public static class Order {

private final String orderId;

private final LocalDateTime orderTime;

public Order(){this.orderId=UUID.randomUUID().toString();this.orderTime=LocalDateTime.now();}

public String getOrderId(){return orderId;}

public LocalDateTime getOrderTime(){return orderTime;}

@Override public String toString(){return "Order{id="+orderId+", time="+orderTime+"}";}

}

public static class PaymentProcessor {

private final String processorId;

private final String securityKey;

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

public boolean processPayment(Order o,Customer c,double amount){return amount>0;}

@Override public String toString(){return "PaymentProcessor{id="+processorId+"}";}

}

public static class ShippingCalculator {

private final Map<String,Double> shippingRates;

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

public double calculateShipping(String region,double weight){return shippingRates.getOrDefault(region,10.0)\*weight;}

@Override public String toString(){return "ShippingCalculator{rates="+shippingRates+"}";}

}

public static boolean processOrder(Object order,Object customer) {

return (order instanceof Order) && (customer instanceof Customer);

}

public static void addToCatalog(String key,Object product){productCatalog.put(key,product);}

public static Map<String,Object> getCatalog(){return new HashMap<>(productCatalog);}

public static void main(String[] args) {

Product laptop=Product.createElectronics("Laptop","Dell",800,new String[]{"i7","16GB"},Map.of("Color","Black"));

Product shirt=Product.createClothing("Shirt","Levis",50,new String[]{"Cotton"},Map.of("Size","M"));

Product book=Product.createBooks("Book","Penguin",20,new String[]{"Hardcover"},Map.of("Pages","300"));

addToCatalog("L1",laptop);

addToCatalog("S1",shirt);

addToCatalog("B1",book);

Customer c=new Customer("user@mail.com","Alice","12345");

ShoppingCart cart=new ShoppingCart(c.getCustomerId());

cart.addItem(laptop,1);

cart.addItem(shirt,2);

cart.addItem(book,3);

System.out.println(cart.getCartSummary());

Order o=new Order();

PaymentProcessor pp=new PaymentProcessor("PP1","SEC123");

ShippingCalculator sc=new ShippingCalculator(Map.of("US",5.0,"EU",8.0));

double shipping=sc.calculateShipping("US",laptop.getWeight()+shirt.getWeight()+book.getWeight());

boolean paid=pp.processPayment(o,c,cart.totalAmount+shipping);

System.out.println("Order processed: "+processOrder(o,c)+", Paid: "+paid+", Shipping: "+shipping);

}

}

