WEEK 5- ASSIGNMENT PROBLEMS

Name:Ramesh Harisabapathi Chettiar

Date of Submission:17/09/25

Ans.
```
import java.time.LocalDate;
import java.util.*;

public class HospitalSystemExample {

    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) {
            this.recordId = Objects.requireNonNull(recordId, "Record ID cannot be null");
            this.patientDNA = Objects.requireNonNull(patientDNA, "DNA data cannot be null");
            this.allergies = allergies != null ? allergies.clone() : new String[0];
            this.medicalHistory = medicalHistory != null ? medicalHistory.clone() : new String[0];
            this.birthDate = Objects.requireNonNull(birthDate, "Birth date cannot be null");
            this.bloodType = Objects.requireNonNull(bloodType, "Blood type cannot be null");

            validateHIPAACompliance();
        }

        private void validateHIPAACompliance() {
            if (recordId.isEmpty() || patientDNA.isEmpty() || bloodType.isEmpty()) {
```

```
            throw new IllegalArgumentException("Required medical fields cannot be empty");
        }
    }
    public String getRecordId() { return recordId; }
    public String getPatientDNA() { return patientDNA; }
    public String[] getAllergies() { return allergies.clone(); }
    public String[] getMedicalHistory() { return medicalHistory.clone(); }
    public LocalDate getBirthDate() { return birthDate; }
    public String getBloodType() { return bloodType; }

    public final boolean isAllergicTo(String substance) {
        for (String allergy : allergies) {
            if (allergy.equalsIgnoreCase(substance)) {
                return true;
            }
        }
        return false;
    }

    @Override
    public String toString() {
        return "MedicalRecord[recordId=" + recordId + ", bloodType=" + bloodType + "]";
    }
}
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;

public Patient(String emergencyContact) {
   this.patientId = "TEMP-" + UUID.randomUUID().toString().substring(0, 8);
   this.medicalRecord = null;
   this.emergencyContact = Objects.requireNonNull(emergencyContact);
   this.currentName = "Unknown";
}

public Patient(String patientId, MedicalRecord medicalRecord, String currentName,
           String emergencyContact, String insuranceInfo) {
   this.patientId = Objects.requireNonNull(patientId);
   this.medicalRecord = Objects.requireNonNull(medicalRecord);
   this.currentName = Objects.requireNonNull(currentName);
   this.emergencyContact = Objects.requireNonNull(emergencyContact);
   this.insuranceInfo = insuranceInfo;
   validatePrivacyPermissions();
}

private void validatePrivacyPermissions() {
   if (patientId.isEmpty() || currentName.isEmpty()) {
      throw new IllegalArgumentException("Required patient information missing");
   }
}

public String getPatientId() { return patientId; }
public MedicalRecord getMedicalRecord() { return medicalRecord; }
public String getCurrentName() { return currentName; }
public String getEmergencyContact() { return emergencyContact; }
public String getInsuranceInfo() { return insuranceInfo; }
public int getRoomNumber() { return roomNumber; }
```

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

public void setCurrentName(String currentName) {
   this.currentName = Objects.requireNonNull(currentName);
}

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

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

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

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

String getBasicInfo() {
   return "Patient ID: " + patientId + ", Name: " + currentName + ", Room: " + roomNumber;
}

public String getPublicInfo() {
   return "Name: " + currentName + ", Room: " + roomNumber;
}

@Override
```

```
    public String toString() {
        return "Patient[ID=" + patientId + ", Name=" + currentName + "]";
    }
  }

  public static void main(String[] args) {

    MedicalRecord record = new MedicalRecord(
      "MR12345", "DNA_SEQUENCE_XYZ",
      new String[]{"Penicillin", "Shellfish"},
      new String[]{"Appendectomy 2018", "Broken arm 2020"},
      LocalDate.of(1985, 5, 15),
      "O+"
    );
    Patient patient = new Patient("P1001", record, "John Doe", "555-1234", "INS123");
    System.out.println("Patient Public Info: " + patient.getPublicInfo());
    System.out.println("Is allergic to Penicillin: " + record.isAllergicTo("Penicillin"));
    System.out.println("Medical Record: " + record);
  }
}
```

```
Patient Public Info: Name: John Doe, Room: 0
Is allergic to Penicillin: true
Medical Record: MedicalRecord[recordId=MR12345, bloodType=O+]
```

Ans2.
```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.*;

public class BankingSystemExample {
  static final class Transaction {
    private final String transactionId;
```

```
private final LocalDateTime timestamp;
private final double amount;
private final String transactionType;
private final String description;
private final String fromAccount;
private final String toAccount;
private final Map<String, String> metadata;

public Transaction(String transactionId, double amount, String transactionType,
                   String description, String fromAccount, String toAccount) {
    this.transactionId = Objects.requireNonNull(transactionId);
    this.timestamp = LocalDateTime.now();
    this.amount = amount;
    this.transactionType = Objects.requireNonNull(transactionType);
    this.description = description;
    this.fromAccount = fromAccount;
    this.toAccount = toAccount;
    this.metadata = new HashMap<>();
}

public String getTransactionId() { return transactionId; }
public LocalDateTime getTimestamp() { return timestamp; }
public double getAmount() { return amount; }
public String getTransactionType() { return transactionType; }
public String getDescription() { return description; }
public String getFromAccount() { return fromAccount; }
public String getToAccount() { return toAccount; }
public Map<String, String> getMetadata() { return new HashMap<>(metadata); }

public final boolean isValid() {
    return amount > 0 &&
```

```
                !transactionId.isEmpty() &&
                (transactionType.equals("DEPOSIT") ||
                 transactionType.equals("WITHDRAWAL") ||
                 transactionType.equals("TRANSFER"));
        }

        public void addMetadata(String key, String value) {
            metadata.put(key, value);
        }

        @Override
        public String toString() {
            return "Transaction[ID=" + transactionId + ", Type=" + transactionType + ", Amount=" +
amount + "]";
        }
    }

    static class BankAccount {
        private final String accountNumber;
        private final String accountType;
        private final LocalDate openDate;
        private double balance;
        private String accountStatus;
        private final String ownerId;
        private final List<Transaction> transactionHistory;

        public BankAccount(String accountNumber, String accountType, String ownerId) {
            this.accountNumber = Objects.requireNonNull(accountNumber);
            this.accountType = Objects.requireNonNull(accountType);
            this.openDate = LocalDate.now();
            this.balance = 0.0;
```

```
    this.accountStatus = "ACTIVE";
    this.ownerId = Objects.requireNonNull(ownerId);
    this.transactionHistory = new ArrayList<>();
}

public String getAccountNumber() { return accountNumber; }
public String getAccountType() { return accountType; }
public LocalDate getOpenDate() { return openDate; }
public double getBalance() { return balance; }
public String getAccountStatus() { return accountStatus; }
public String getOwnerId() { return ownerId; }
public List<Transaction> getTransactionHistory() {
    return new ArrayList<>(transactionHistory);
}

public boolean processTransaction(Transaction transaction) {
    if (!transaction.isValid()) {
        return false;
    }

    switch (transaction.getTransactionType()) {
        case "DEPOSIT":
            balance += transaction.getAmount();
            break;
        case "WITHDRAWAL":
            if (balance >= transaction.getAmount()) {
                balance -= transaction.getAmount();
            } else {
                return false;
            }
            break;
```

```
        }

        transactionHistory.add(transaction);
        return true;
    }

    public String getPublicAccountInfo() {
        String maskedNumber = "****" + accountNumber.substring(accountNumber.length() - 4);
        return "Account Type: " + accountType + ", Status: " + accountStatus +
               ", Number: " + maskedNumber;
    }

    @Override
    public String toString() {
        return "BankAccount[Number=" + accountNumber + ", Type=" + accountType +
               ", Balance=" + balance + "]";
    }
}

public static void main(String[] args) {

    BankAccount account = new BankAccount("1234567890", "SAVINGS", "C1001");

    Transaction deposit = new Transaction("TXN001", 1000.0, "DEPOSIT", "Initial deposit", "CASH",
account.getAccountNumber());
    Transaction withdrawal = new Transaction("TXN002", 200.0, "WITHDRAWAL", "ATM
withdrawal", account.getAccountNumber(), "ATM001");

    account.processTransaction(deposit);
    account.processTransaction(withdrawal);

    System.out.println("Account Info: " + account.getPublicAccountInfo());
```

```
        System.out.println("Final Balance: $" + account.getBalance());
        System.out.println("Transaction History: " + account.getTransactionHistory().size() + "
transactions");
    }
}
```

```
Account Info: Account Type: SAVINGS, Status: ACTIVE, Number: ****7890
Final Balance: $800.0
Transaction History: 2 transactions
```