Name: Abhinab Sherchan

Group:L5CG24

```
Question-1 package
Workshop7;

/**
* The Question1 class that contains all the nested classes: Address, Person, Student, Professor.
 */
public class Question1 {

   /**
* Represents an Address with attributes such as street, city, state, postal code, and country.
    */
   public class Address {
private String street;
private String city;        private
String state;        private int
postalCode;        private String
country;

       /**
* Constructor to initialize an Address object.
        *
* @param street Street of the address
* @param city City of the address
* @param state State of the address
* @param postalCode Postal code of the address
* @param country Country of the address
```

```
         */
        public Address(String street, String city, String state, int postalCode, String country) {
this.street = street;            this.city = city;            this.state = state;            this.postalCode =
postalCode;            this.country = country;
        }

        /**
* Validates the address by checking that street, city, and postal code are not empty or invalid.
         *
* @return true if the address is valid, false otherwise
         */
        public boolean validate() {            return street
!= null && !street.isEmpty() &&                   city !=
null && !city.isEmpty() &&                  postalCode >
0;
        }

        /**
* Outputs the address in a label format: street, city, state, postal code, and country.
         *
* @return A formatted string representing the address
         */
        public String outputAsLabel() {            return street + ", " + city + ", " + state
+ " - " + postalCode + ", " + country;
        }
    }

    /**
* Represents a Person with basic information like name, phone number, and email address.
```

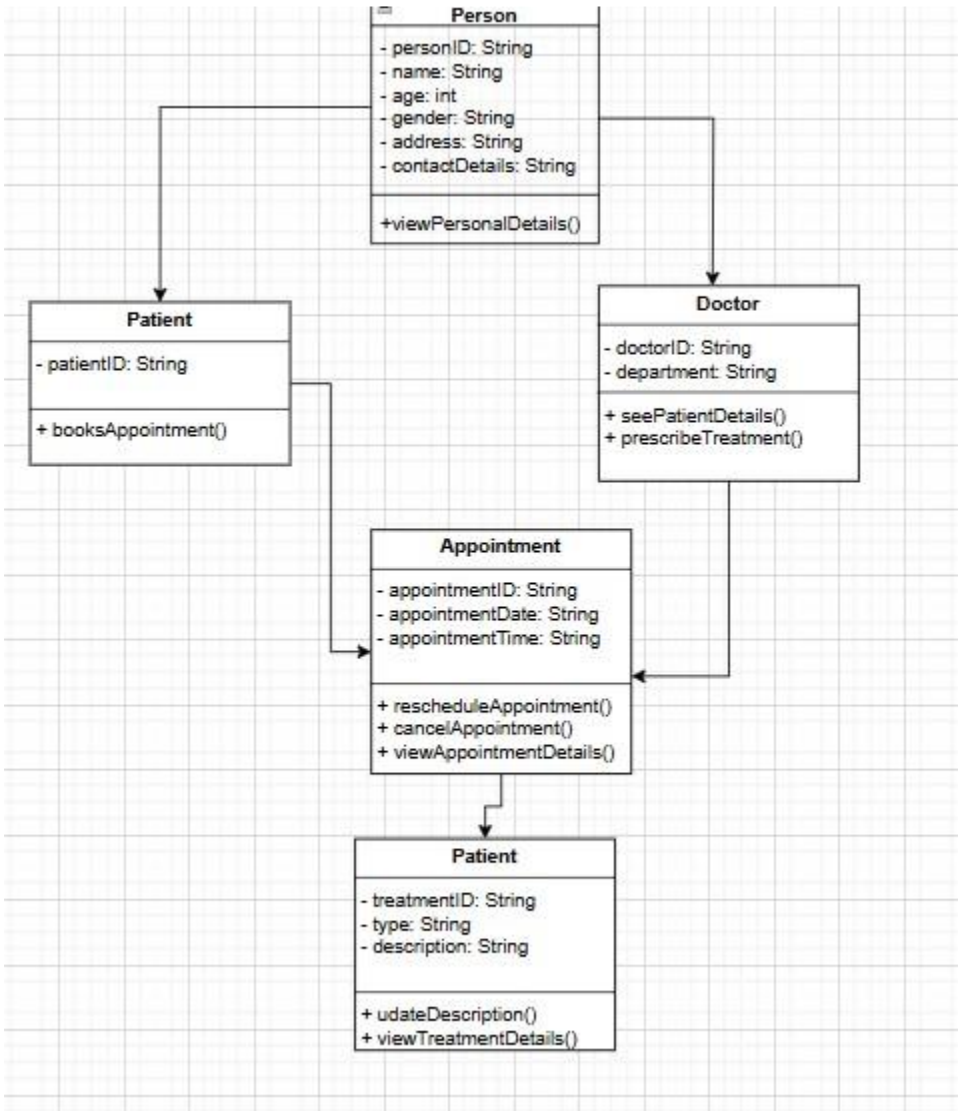
```
    */
   public class Person {
protected String name;
private String phoneNumber;
private String emailAddress;


       /**
* Constructor to initialize a Person object with name, phone number, and email address.
        *
* @param name Name of the person
* @param phoneNumber Phone number of the person
* @param emailAddress Email address of the person
        */
       public Person(String name, String phoneNumber, String emailAddress) {
this.name = name;           this.phoneNumber = phoneNumber;
this.emailAddress = emailAddress;
       }


       /**
* Allows the person to purchase a parking pass.
        */
       public void purchaseParkingPass() {
          System.out.println("Parking ticket purchased by: " + name);
       }
   }


   /**
* Represents a Student, which extends the Person class. Contains student-specific attributes      * and
  behaviors such as eligibility to enroll and seminars taken.
```

```
    */
   class Student extends Person {
private int studentNumber;        private
int averageMark;

       /**
* Constructor to initialize a Student object with name, phone number, email address,         * student
  number, and average mark.
        *
* @param name Name of the student
* @param phoneNumber Phone number of the student
* @param emailAddress Email address of the student
* @param studentNumber Unique student number
* @param averageMark Average mark of the student
        */
       public Student(String name, String phoneNumber, String emailAddress, int studentNumber, int
averageMark) {           super(name, phoneNumber, emailAddress);           this.studentNumber =
studentNumber;           this.averageMark = averageMark;
       }

       /**
* Determines if the student is eligible to enroll in a course based on their average mark.
        *
* @param course The course the student wants to enroll in
* @return true if the student has an average mark of 50 or above, false otherwise
        */
       public boolean isEligibleToEnroll(String course) {
return averageMark >= 50;
       }
```

```
      /**
* Returns the number of seminars taken by the student. Default implementation returns 0.
       *
* @return The number of seminars taken
       */
      public int getSeminarsTaken() {
         return 0;
      }
   }

   /**
* Represents a Professor, which extends the Person class. Contains professor-specific attributes      * and
  behaviors such as supervising students and displaying professor details.
    */
   class Professor extends Person {
private int staffNumber;        private
int yearsOfService;        private int
numberOfClasses;

      /**
* Constructor to initialize a Professor object with name, phone number, email address,        * staff
  number, years of service, and number of classes.

       *
* @param name Name of the professor
* @param phoneNumber Phone number of the professor
* @param emailAddress Email address of the professor
* @param staffNumber Unique staff number
* @param yearsOfService Years of service as a professor
* @param numberOfClasses Number of classes taught by the professor
     */
    public Professor(String name, String phoneNumber, String emailAddress, int staffNumber, int
yearsOfService, int numberOfClasses) {         super(name, phoneNumber, emailAddress);
this.staffNumber = staffNumber;         this.yearsOfService = yearsOfService;
this.numberOfClasses = numberOfClasses;
    }

    /**
* Supervises a student and prints a message indicating the student being supervised by the professor.
     *
* @param student The student being supervised
     */
    public void supervise(Student student) {
      System.out.println(name + " is supervising student " + student.name);
    }

    /**
* Displays the details of the professor.
     */
    public void displayProfessorDetails() {
      System.out.println("Professor Name: " + name);
      System.out.println("Years of Service: " + yearsOfService);
      System.out.println("Number of Classes: " + numberOfClasses);
    }
  }

  /**
```

```
* Main method to test the functionality of the Address, Student, and Professor classes.
    *
* @param args Command-line arguments (not used)
    */
   public static void main(String[] args) {
       Address address = new Question1().new Address("Kakani", "Nuwakot", "Bagmati", 44600, "Nepal");
       System.out.println("Address valid: " + address.validate());
       System.out.println("Formatted address: " + address.outputAsLabel());


       Student student = new Question1().new Student("Alok", "123456789", "fewfewf@gmail.com",
726125, 97);
       System.out.println("Eligible to enroll: " + student.isEligibleToEnroll("Math 101"));
       System.out.println("Seminars taken: " + student.getSeminarsTaken());


       Professor professor = new Question1().new Professor("Dr. Suraj", "987-654-3210",
"Shresthasuraj397@gmail.com", 5001, 15, 4);
professor.displayProfessorDetails();        professor.supervise(student);
   }
}
```

Question2(Hospital Management System)



package Workshop7;

import java.util.Date;

/**

* Represents a person in the hospital system. This class contains the common details

* for both patients and doctors, such as name, age, gender, address, and contact information. */

class HospitalPerson { protected

String personID; protected String

```
name;     protected int age;
protected String gender;
protected String address;
protected String contactDetails;

   /**
* Constructor to initialize a HospitalPerson object with given details.
    *
* @param personID ID of the person
* @param name Name of the person
* @param age Age of the person
* @param gender Gender of the person
* @param address Address of the person
* @param contactDetails Contact details of the person
    */
   public HospitalPerson(String personID, String name, int age, String gender, String address, String
contactDetails) {        this.personID = personID;        this.name = name;        this.age = age;
this.gender = gender;        this.address = address;        this.contactDetails = contactDetails;
   }

   /**
* Displays the personal details of the hospital person.
    */
   public void viewPersonalDetails() {
System.out.println("Name: " + name);
       System.out.println("Age: " + age);
       System.out.println("Gender: " + gender);
       System.out.println("Address: " + address);
       System.out.println("Contact Details: " + contactDetails);
   }
}

/**
* Represents a patient in the hospital system. Inherits from HospitalPerson   * and contains additional
  functionality related to the patient.
 */
class Patient extends HospitalPerson {     private
String patientID;

   /**
* Constructor to initialize a Patient object with the given details.
    *
* @param personID ID of the person
* @param name Name of the person
* @param age Age of the person
* @param gender Gender of the person
* @param address Address of the person
* @param contactDetails Contact details of the person
* @param patientID ID of the patient
    */     public Patient(String personID, String name, int age, String gender, String address,
String contactDetails, String patientID) {        super(personID, name, age, gender, address,
contactDetails);         this.patientID = patientID;
   }

   /**
* Allows the patient to book an appointment with a doctor.
    *
* @param appointment The appointment to be booked
```

```
    */
   public void bookAppointment(Appointment appointment) {
       System.out.println(name + " (Patient ID: " + patientID + ") has booked an appointment with Doctor "
+ appointment.getDoctor().getName());
   }

   /**
* Returns the patient ID.
    *
* @return The patient ID
    */
   public String getPatientID() {
return patientID;
   }
}

/**
* Represents a doctor in the hospital system. Inherits from HospitalPerson
* and contains additional functionality related to the doctor.
 */
class Doctor extends HospitalPerson {
private String doctorID;       private
String department;

   /**
* Constructor to initialize a Doctor object with the given details.
    *
* @param personID ID of the person
* @param name Name of the person
* @param age Age of the person
* @param gender Gender of the person
* @param address Address of the person
* @param contactDetails Contact details of the person
* @param doctorID ID of the doctor
* @param department Department of the doctor
    */
   public Doctor(String personID, String name, int age, String gender, String address, String
contactDetails, String doctorID, String department) {        super(personID, name, age, gender,
address, contactDetails);         this.doctorID = doctorID;         this.department = department;
   }

   /**
* Displays the patient details that the doctor is seeing.
    *
* @param patient The patient being seen by the doctor
    */     public void seePatientDetails(Patient
patient) {
       System.out.println("Doctor " + name + " is seeing patient " + patient.name + " (Patient ID: " +
patient.getPatientID() + ")");
   }

   /**
* Prescribes a treatment for a patient.
    *
* @param treatment The treatment to be prescribed
* @param patient The patient who is receiving the treatment
    */
   public void prescribeTreatment(Treatment treatment, Patient patient) {
       System.out.println("Doctor " + name + " prescribes " + treatment.getType() + " to patient " +
patient.name + " (Patient ID: " + patient.getPatientID() + ")");

   }


   /**

* Returns the name of the doctor.

    *

* @return The name of the doctor

    */

   public String getName() {

return name;

   }


   /**

* Returns the doctor ID.

    *

* @return The doctor ID

    */

   public String getDoctorID() {

return doctorID;

   }


   /**

* Returns the department of the doctor.

    *

* @return The department of the doctor

    */

   public String getDepartment() {

return department;
```

```
    }
}

/**
* Represents an appointment between a patient and a doctor in the hospital system.
 */
class Appointment {     private String
appointmentID;     private Date
appointmentDate;     private String
appointmentTime;     private Patient
patient;     private Doctor doctor;

    /**
* Constructor to initialize an Appointment object with the given details.
     *
* @param appointmentID The appointment ID
* @param appointmentDate The appointment date
* @param appointmentTime The appointment time
* @param patient The patient for the appointment
* @param doctor The doctor for the appointment
     */
    public Appointment(String appointmentID, Date appointmentDate, String appointmentTime,
Patient patient, Doctor doctor) {         this.appointmentID = appointmentID;
this.appointmentDate = appointmentDate;         this.appointmentTime = appointmentTime;
this.patient = patient;         this.doctor = doctor;
    }

    /**
* Reschedules the appointment to a new date and time.
```

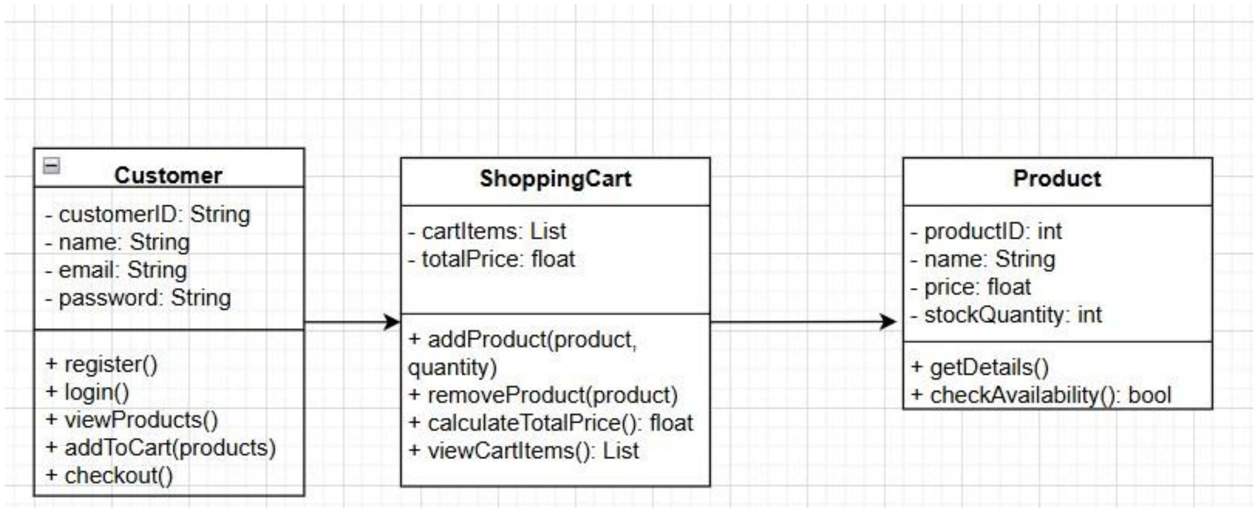
```
    *
* **@param** newDate The new appointment date
* **@param** newTime The new appointment time
    */
   public void rescheduleAppointment(Date newDate, String newTime) {
this.appointmentDate = newDate;        this.appointmentTime = newTime;
       System.out.println("Appointment has been rescheduled to " + newDate + " at " + newTime);
   }

   /**
* Cancels the appointment.
    */     public void
cancelAppointment() {
       System.out.println("Appointment has been cancelled.");
   }

   /**
* Displays the appointment details.
    */
   public void viewAppointmentDetails() {
       System.out.println("Appointment ID: " + appointmentID);
       System.out.println("Appointment Date: " + appointmentDate);
       System.out.println("Appointment Time: " + appointmentTime);
       System.out.println("Patient: " + patient.name + " (Patient ID: " + patient.getPatientID() + ")");
       System.out.println("Doctor: " + doctor.name + " (Doctor ID: " + doctor.getDoctorID() + ")");
   }

   /**
* Returns the doctor associated with this appointment.
```

```
    *
* @return The doctor
    */
   public Doctor getDoctor() {
return doctor;
   }
}

/**
* Represents a treatment prescribed to a patient in the hospital system.
 */
class Treatment {     private
String treatmentID;     private
String type;     private String
description;

   /**
* Constructor to initialize a Treatment object with the given details.
    *
* @param treatmentID The treatment ID
* @param type The treatment type
* @param description The treatment description
    */
   public Treatment(String treatmentID, String type, String description) {
this.treatmentID = treatmentID;        this.type = type;        this.description
= description;
   }

   /**
```

```
* Updates the description of the treatment.
     *
* @param newDescription The new description for the treatment
     */
    public void updateDescription(String newDescription) {
this.description = newDescription;
        System.out.println("Treatment description updated.");
    }

    /**
* Displays the details of the treatment.
     */
    public void viewTreatmentDetails() {
        System.out.println("Treatment ID: " + treatmentID);
        System.out.println("Treatment Type: " + type);
        System.out.println("Description: " + description);
    }

    /**
* Returns the treatment type.
     *
* @return The treatment type
     */
    public String getType() {
return type;
    }
}

/**
```

```
* Main class that runs the Hospital Management System.
 */
public class HospitalManagementSystem {


   /**
* Main method to run the hospital management system.
    *
* @param args Command-line arguments (not used)
    */
   public static void main(String[] args) {


       Doctor doctor = new Doctor("D1", "Dr.Suraj", 30, "Male", "Chabhail, Kathmandu", "9847873191",
"D1", "MagicianSuraj");
       Patient patient = new Patient("P1", "Samir", 21, "Male", "Dang, Nepalgunj", "9876543210", "P1");


       Appointment appointment = new Appointment("A1", new Date(), "10:00 AM", patient, doctor);


       patient.bookAppointment(appointment);


       appointment.viewAppointmentDetails();


       Treatment treatment = new Treatment("T1", "Medication", "Asthma");
doctor.prescribeTreatment(treatment, patient);


       treatment.updateDescription("Asthma");
treatment.viewTreatmentDetails();
   }
}
```

Question3(Shopping Cart)



```
package Workshop7;



import java.util.*;


/**
* Represents a Customer in the system who can register, log in, and shop for products.
 */
class Customer {     private
String customerId;     private
String name;     private String
email;     private String
password;     private
ShoppingCart cart;


   /**
* Constructor to initialize a Customer object.
```

```
     *
* @param customerId Unique ID of the customer
* @param name       Name of the customer
* @param email      Email address of the customer
* @param password   Password for the customer's account
     */
    public Customer(String customerId, String name, String email, String password) {
this.customerId = customerId;        this.name = name;        this.email = email;
this.password = password;        this.cart = new ShoppingCart();
    }

    /**
* Registers a new customer.
     */
    public void register() {
        System.out.println("Customer " + name + " registered successfully.");
    }

    /**
* Logs in an existing customer.
     *
* @param email    The email used to log in
* @param password The password used to log in
* @return True if login is successful, false otherwise
     */
    public boolean login(String email, String password) {        if
(this.email.equals(email) && this.password.equals(password)) {
System.out.println("Login successful for " + name);
```

```
            return true;
        } else {
            System.out.println("Login failed for " + name);
return false;
        }
    }

    /**
* Views the list of available products.
     *
* @param products List of all products in the system
     */
    public void viewProducts(List<Product> products) {
for (Product product : products) {
product.getDetails();
        }
    }

    /**
* Adds a product to the shopping cart.
     *
* @param product The product to add
* @param quantity The quantity of the product
     */
    public void addToCart(Product product, int quantity) {
cart.addProduct(product, quantity);
    }

    /**
```

```
* Proceeds to checkout, displaying the total price and emptying the cart.
    */
   public void checkout() {
       System.out.println("Checkout initiated for " + name);
cart.viewCartItems();
       System.out.println("Total Price: " + cart.calculateTotalPrice());        cart.clearCart();
       System.out.println("Checkout completed.");
   }
}

/**
* Represents a Product with attributes like name, price, and stock quantity.
 */
class Product {     private
String productId;     String
name;     double price;
private int stockQuantity;

   /**
* Constructor to initialize a Product object.
    *
* @param productId    Unique ID of the product
* @param name         Name of the product
* @param price        Price of the product
* @param stockQuantity Available stock quantity
    */
   public Product(String productId, String name, double price, int stockQuantity) {
this.productId = productId;        this.name = name;        this.price = price;
this.stockQuantity = stockQuantity;
```

```
    }


    /**
* Displays the product details.
     */
    public void getDetails() {
        System.out.println("Product ID: " + productId + ", Name: " + name + ", Price: " + price + ", Stock: " +
stockQuantity);
    }


    /**
* Checks if the product is available in the specified quantity.
     *
* @param quantity The quantity to check
* @return True if available, false otherwise
     */
    public boolean checkAvailability(int quantity) {
return stockQuantity >= quantity;
    }


    /**
* Reduces the stock quantity by the specified amount.
     *
* @param quantity The quantity to deduct
     */
    public  void  reduceStock(int  quantity)  {
if          (checkAvailability(quantity))          {
stockQuantity -= quantity;
        }
```

```
   }
}

/**
* Represents a Shopping Cart, which contains products and calculates the total price.
 */
class ShoppingCart {     private Map<Product,
Integer> cartItems;     private double
totalPrice;

   /**
* Constructor to initialize an empty ShoppingCart.
    */
   public ShoppingCart() {
this.cartItems = new HashMap<>();
this.totalPrice = 0.0;
   }

   /**
* Adds a product to the cart with the specified quantity.
    *
* @param product  The product to add
* @param quantity The quantity of the product
    */
   public void addProduct(Product product, int quantity) {        if
(product.checkAvailability(quantity)) {            cartItems.put(product,
cartItems.getOrDefault(product, 0) + quantity);
product.reduceStock(quantity);
          System.out.println(quantity + " units of " + product.name + " added to the cart.");
```

```
        } else {
            System.out.println("Insufficient stock for " + product.name);
        }
    }

    /**
 * Removes a product from the cart.
     *
 * @param product The product to remove
     */
    public void removeProduct(Product product) {
if (cartItems.containsKey(product)) {
cartItems.remove(product);
            System.out.println(product.name + " removed from the cart.");
        } else {
            System.out.println(product.name + " is not in the cart.");
        }
    }

    /**
 * Calculates the total price of items in the cart.
     *
 * @return The total price
     */
    public double calculateTotalPrice() {        totalPrice = 0.0;        for
(Map.Entry<Product, Integer> entry : cartItems.entrySet()) {
totalPrice += entry.getKey().price * entry.getValue();
        }
        return totalPrice;
```

```
   }

   /**
* Displays the items in the cart along with their quantities.
    */
   public void viewCartItems() {
System.out.println("Cart Items:");
      for (Map.Entry<Product, Integer> entry : cartItems.entrySet()) {
System.out.println(entry.getKey().name + " x " + entry.getValue());
      }
   }

   /**
* Clears all items from the cart.
    */
   public void clearCart() {
cartItems.clear();        totalPrice
= 0.0;
   }
}

/**
* Main class to demonstrate the shopping system.
 */
public class ShoppingCartSystem {     public static
void main(String[] args) {        List<Product>
products = Arrays.asList(            new
Product("P1", "Laptop", 50000.0, 10),            new
Product("P2", "Phone", 30000.0, 20),            new
Product("P3", "Tablet", 20000.0, 15)
        );

        Customer customer = new Customer("C1", "John Doe", "john@example.com", "password123");

        customer.register();        if
(customer.login("john@example.com", "password123")) {
customer.viewProducts(products);
customer.addToCart(products.get(0), 2);
customer.addToCart(products.get(1), 1);
customer.checkout();
        }
    }
}
```