CSE130 P - Object Oriented Programming in Java

1. Create a class Calculator with overloaded methods to calculate the sum of two integers, three integers, and two double values. The method should return the sum based on the type and number of parameters.

```
class Calculator{

    //addition of two integers     int
sum(int a,int b){
        return a+b;
    }

    //addition of three integers     int
sum(int a,int b,int c){         return a+b+c;
    }

    //addition of two double values     double
sum(double a,double b){
        return a+b;
    }

}
class prac1{
    public static void main(String[] args) {

        Calculator c=new  Calculator();

        System.out.println(c.sum(1,2));
        System.out.println(c.sum(1,2,3));
        System.out.println(c.sum(1.2,1.3));
    }
 OUTPUT:
  3
  6
  2.5
```

2. Write a program that validates a password based on rules (at least 8 characters, must contain an uppercase letter, a number, and a special character).If the password is invalid, provide specific feedback on why it failed.

```java
import java.util.Scanner;

class PasswordValidator {
     private String password;

     // Constructor to initialize the password
     public PasswordValidator(String password) {
          this.password = password;
     }

     // Method to validate the password
     public String validate() {
          if (password.length() < 8) {
               return "Password must be at least 8 characters long.";
          } else if (!containsUppercase()) {
               return "Password must contain at least one uppercase letter.";
          } else if (!containsDigit()) {
               return "Password must contain at least one number.";
          } else if (!containsSpecialCharacter()) {
               return "Password must contain at least one special character.";
          }
          return "Password is valid!";
     }

     // Check for at least one uppercase letter
     private boolean containsUppercase() {
          for (int i = 0; i < password.length(); i++) {
               // Get the character at the current index
               char c = password.charAt(i);
               if (Character.isUpperCase(c)) {
                    return true;
               }
          }
          return false;
     }

     // Check for at least one digit
     private boolean containsDigit() {
          for (int i = 0; i < password.length(); i++) {
               char c = password.charAt(i);
```

```
                if (Character.isDigit(c)) {
                    return true;
                }
            }
            return false;
        }

        // Check for at least one special character
        private boolean containsSpecialCharacter() {
            String specialCharacters = "!@#$%^&*()_+\\-
=\\[\\]{};':\"\\\\|,.<>\\/?";
            for (int i = 0; i < password.length(); i++) {
                char c = password.charAt(i);
                if (specialCharacters.indexOf(c) != -1) {
                    return true;
                }
            }
            return false;
        }
}

class Main {
        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter a password to validate: ");
            String password = scanner.nextLine();
            // Create an instance of PasswordValidator
            PasswordValidator validator = new PasswordValidator(password);
            // Validate the password and print the result
            String result = validator.validate();
            System.out.println(result);
        }
}
```

OUTPUT:

```
Enter a password to validate: abcdef

Password must be at least 8 characters long.
```

3. Implement a warehouse inventory system using a single-dimensional array where each index represents a product type.
Update quantities based on user input for adding and removing items from the inventory and handle invalid inputs.

```java
import java.util.Scanner;


class WarehouseInventory {

     private int[] inventory; // Array to hold quantities of each product type

     private int productTypes; // Total number of product types

     // Constructor to initialize the inventory with the number of product types


     public WarehouseInventory(int productTypes) {

          this.productTypes = productTypes;

          this.inventory = new int[productTypes]; // Create an array for the


     }


     // Method to add items to the inventory

     public void addItems(int productIndex, int quantity) {

          // Check if the product index is valid

          if (isValidProductIndex(productIndex)) {

               inventory[productIndex] += quantity; // Add quantity to the inventory

               System.out.println("Added " + quantity + " items to product type " +

                         productIndex + ".");

          } else {

               System.out.println("Invalid product type index.");

          }

     }


     // Method to remove items from the inventory
     public void removeItems(int productIndex, int quantity) {

          // Check if the product index is valid

          if (isValidProductIndex(productIndex)) {

               // Check if there are enough items to remove

               if (inventory[productIndex] >= quantity) {

                    inventory[productIndex] -= quantity; // Remove quantity from the


                    System.out.println("Removed " + quantity + " items from product type
" + productIndex + ".");

               } else {

                    System.out.println("Not enough items in inventory to remove " +
quantity

                              + " items from product type " + productIndex + ".");

               }

          } else {

               System.out.println("Invalid product type index.");

          }

     }


     // Method to display the current inventory

public void displayInventory() {

System.out.println("Current Inventory:");

for (int i = 0; i < productTypes; i++) {

System.out.println("Product Type " + i + ": " + inventory[i] + "

items");

}

}

     // Method to check if product index is valid

     private boolean isValidProductIndex(int index) {

          return index >= 0 && index < productTypes; // Check if index is within
```

```
    }

}


class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of product types in the warehouse:");

        int productTypes = scanner.nextInt(); // Get the number of product types


        // Create an instance of WarehouseInventory

        WarehouseInventory warehouse = new WarehouseInventory(productTypes);


        // Main loop for user interaction

        while (true) {

            System.out.println("\nOptions:");

            System.out.println("1. Add Items");

            System.out.println("2. Remove Items");

            System.out.println("3. Display Inventory");

            System.out.println("4. Exit");

            System.out.print("Choose an option: ");

            int choice = scanner.nextInt(); // Get the user's choice from users

            switch (choice) {

                case 1: // Add items

                    System.out.print("Enter product type index to add items: ");

                    int addIndex = scanner.nextInt(); // Get product index

                    System.out.print("Enter quantity to add: ");

                    int addQuantity = scanner.nextInt(); // Get quantity to add

                    warehouse.addItems(addIndex, addQuantity); // Add items to
```

```
                        break;
                    case 2: // Remove items
                        System.out.print("Enter product type index to remove items: ");
                        int removeIndex = scanner.nextInt(); // Get product index
                        System.out.print("Enter quantity to remove: ");
                        int removeQuantity = scanner.nextInt(); // Get quantity to
remove
                        warehouse.removeItems(removeIndex, removeQuantity); //
                        break;
                    case 3: // Display inventory
                        warehouse.displayInventory(); // Show current inventory
                        break;
                    case 4: // Exit the program
                        System.out.println("Exiting the program.");
                        scanner.close(); // Close the scanner
                        return; // Exit the main method
                    default: // Invalid option
                        System.out.println("Invalid option. Please choose again.");
                }
            }
        }
    }
```

OUTPUT :

```
Enter the number of product types in the warehouse: 3


Options:

1. Add Items

2. Remove Items

3. Display Inventory

4. Exit

Choose an option: 1

Enter product type index to add items: 0

Enter quantity to add: 10

Added 10 items to product type 0.


Options:

1. Add Items

2. Remove Items

3. Display Inventory

4. Exit

Choose an option: 1

Enter product type index to add items: 1

Enter quantity to add: 5

Added 5 items to product type 1.


Options:

1. Add Items

2. Remove Items

3. Display Inventory
```

```
4. Exit
Choose an option: 3
Current Inventory:
Product Type 0: 10 items
Product Type 1: 5 items
Product Type 2: 0 items

Options:
1. Add Items
2. Remove Items
3. Display Inventory
4. Exit
Choose an option: 2
Enter product type index to remove items: 0
Enter quantity to remove: 4
Removed 4 items from product type 0.

Options:
1. Add Items
2. Remove Items
3. Display Inventory
4. Exit
Choose an option: 3
Current Inventory:
Product Type 0: 6 items
Product Type 1: 5 items
Product Type 2: 0 items
```

```
Options:

1. Add Items

2. Remove Items

3. Display Inventory

4. Exit

Choose an option: 4

Exiting the program.
```

4. Create a Movie class with private attributes: title, director, genre, and rating. Create an array of five Movie objects and use constructors to initialize those objects. Write a static method getMoviesByDirector() in the Main class that takes an array of Movie objects and a director's name as input, and returns a list of movies directed by that director. Write the necessary getters to return the attributes.

```
import java.util.Scanner;

class Movie {
     private String title;
     private String director;
     private String genre;
     private float rating;

     Movie(String title, String director, String genre, float rating) {
          this.title = title;
          this.director = director;
          this.genre = genre;
          this.rating = rating;
     }

     public String getTitle() {
          return title;
     }

     public String getDirector() {
          return director;
     }

     public String getGenre() {
          return genre;
     }

     public float getRating() {
          return rating;
     }

}

class PracticleQ4java {
     public static void main(String[] args) {
          Movie[] M = new Movie[5];
          Scanner sc = new Scanner(System.in);
          for (int i = 0; i < M.length; i++) {
               System.out.println("Enter Title");
```

```
            String title = sc.nextLine();
            System.out.println("Enter Director");
            String director = sc.nextLine();
            System.out.println("Enter genre");
            String genre = sc.nextLine();
            System.out.println("Enter rating");
            float rating = sc.nextFloat();
            sc.nextLine();
            M[i] = new Movie(title, director, genre, rating);
        }
        System.out.println("Enter Director to filter");
        String director = sc.nextLine();
        Movie[] filterMovie = getMoviesByDirector(M, director);
        for (Movie M1 : filterMovie) {
            System.out.println(M1.getTitle() + "(" + M1.getDirector() + ")");
        }

    }

    static Movie[] getMoviesByDirector(Movie[] M, String director) {
        Movie[] M2 = new Movie[M.length];
        int index = 0;
        for (Movie M1 : M) {
            if (M1.getDirector().equalsIgnoreCase(director)) {
                M2[index++] = M1;
            }
        }
        Movie[] result = new Movie[index];
        for (int i = 0; i < index; i++) {
            result[i] = M2[i];
        }
        return result;
    }
}
```

```
OUTPUT:
Enter Title:
Inception
Enter Director:
Christopher Nolan
Enter Genre:
Sci-Fi
Enter Rating:
9.0

Enter Title:
```

```
Interstellar
Enter Director:
Christopher Nolan
Enter Genre:
Sci-Fi
Enter Rating:
8.6

Enter Title:
Dunkirk
Enter Director:
Christopher Nolan
Enter Genre:
War
Enter Rating:
7.9

Enter Title:
The Matrix
Enter Director:
Lana Wachowski
Enter Genre:
Sci-Fi
Enter Rating:
8.7

Enter Title:
John Wick
Enter Director:
Chad Stahelski
Enter Genre:
Action
Enter Rating:
7.5

Enter Director to filter:
Christopher Nolan

Inception (Christopher Nolan)
Interstellar (Christopher Nolan)
Dunkirk (Christopher Nolan)
```

5. Create a base class Product with attributes like productID, productName, and price. Extend the class into Clothing with additional attributes like size and material. Further, create a subclass MenClothing with attributes like type (e.g., formal, casual). Create methods in each class to handle actions like adding a product, displaying product details, and calculating discounts.Implement a program that creates different types of clothing items for men, showing how the inheritance chain works in action.

```
// Base class

class Product {

     int productID;

     String productName;

     double price;


     public void addProduct(int productID, String productName, double price) {

          this.productID = productID;

          this.productName = productName;

          this.price = price;

     }


     public void displayProductDetails() {

          System.out.println("Product ID -> " + productID);

          System.out.println("Product Name -> " + productName);

          System.out.println("Price -> " + price);

     }


     public double calculatingDiscounts(double discount) {

          return price - (price * discount / 100);

     }

}


// Child class of Product
```

```
class Clothing extends Product {
    String size;
    String material;

    public void addProduct(int productID, String productName, double price, String
size, String material) {
        super.addProduct(productID, productName, price);
        this.size = size;
        this.material = material;
    }

    public void displayProductDetails() {
        super.displayProductDetails();
        System.out.println("Size -> " + size);
        System.out.println("Material -> " + material);
    }
}

// Subclass of Clothing
class MenClothing extends Clothing {
    String type;

    public void addProduct(int productID, String productName, double price, String
size, String material,
            String type) {
        super.addProduct(productID, productName, price, size, material);
        this.type = type;
    }
```

```
     public void displayProductDetails() {
          super.displayProductDetails();
          System.out.println("Type -> " + type);
     }
}

// Main class
class Main {
     public static void main(String[] args) {
          MenClothing m = new MenClothing();
          m.addProduct(1, "Shirt", 450.0, "L", "Silicon", "Formal");
          System.out.println("Product Details :");
          m.displayProductDetails();
          System.out.println("Discounted Price -> " +
                    m.calculatingDiscounts(10));
     }
}
```

OUTPUT:

```
Product Details:
Product ID -> 1
Product Name -> Shirt
Price -> 450.0
Size -> L
Material -> Silicon
Type -> Formal
Discounted Price -> 405.0
```

6. Create a base class Person with attributes name, age, and gender. Extend the class to Student, Teacher, and Staff. Each subclass should have its own attributes (e.g., Student can have grade and rollNumber, Teacher can have subject and salary, Staff can have department and designation).
Implement methods in each subclass to display details and specific information related to their roles (e.g., getGrade() for students, getSubject() for teachers).
Use a list to store various types of people in the school and display their details.

```
 //base class

class person {

     String name;

     int age;

     String gender;


     person(String name, int age, String gender) {

          this.name = name;

          this.age = age;

          this.gender = gender;

     }


     public void displayDetails() {

          System.out.println("Name : " + name);

          System.out.println("Age : " + age);

          System.out.println("Gender : " + gender);

     }

}


class Student extends person {

     String grade;

     int rollNumber;
```

```
    Student(String name, int age, String gender, String grade, int rollNumber) {
        super(name, age, gender);
        this.grade = grade;
        this.rollNumber = rollNumber;
    }

    String getGrade() {
        return grade;
    }

    public void displayDetails() {
        super.displayDetails();
        System.out.println("Grade => " + grade);
        System.out.println("RollNumber => " + rollNumber);
    }
}

class Teacher extends person {
    String subject;
    int salary;

    Teacher(String name, int age, String gender, String subject, int salary) {
        super(name, age, gender);
        this.subject = subject;
        this.salary = salary;
    }

    String getSubject() {
          return subject;
     }

     public void displayDetails() {
          super.displayDetails();
          System.out.println("Subject => " + subject);
          System.out.println("salary => " + salary);
     }
}

class Staff extends person {
     String departement;
     String designation;

     Staff(String name, int age, String gender, String departement, String
designation) {
          super(name, age, gender);
          this.departement = departement;
          this.designation = designation;
     }

     String getDepartement() {
          return departement;
     }

     public void displayDetails() {
          super.displayDetails();
          System.out.println("Departement => " + departement);
          System.out.println("Designation => " + designation);
     }
}

class eleven {
     public static void main(String[] args) {
          person[] people = new person[3];
          // Adding a Student, Teacher, and Staff to the array
          people[0] = new Student("Shreyansh", 16, "male", "10th
         Grade", 101);
          people[1] = new Teacher("Jitu", 45, "Male", "Mathematics",
         50000);
          people[2] = new Staff("Kishu", 50, "male", "Administration",
         "Clerk");
          // Displaying the details of each person in the array
          for (person person : people) {
          System.out.println("\nDetails:");
          person.displayDetails();
         }
         }
}
```

OUTPUT:

```
Details:
Name : Shreyansh
Age : 16
Gender : Male
Grade => 10th Grade
```

```
RollNumber => 101

Details:
Name : Jitu
Age : 45
Gender : Male
Subject => Mathematics
Salary => 50000

Details:
Name : Kishu
Age : 50
Gender : Male
Department => Administration
Designation => Clerk
```

7. Create a package employee that contains classes Employee, Manager, and HR. The Employee class should contain details like name, salary, and designation, while Manager and HR should extend the Employee class with additional functionalities. Create another package payroll where salary calculations and payslips are generated.

```
 //employee Package
//Employee.java
package employee;

public class Employee {
     protected String name;
     protected double salary;
     protected String designation;

     // Constructor
     public Employee(String name, double salary, String designation) {
          this.name = name;
          this.salary = salary;
          this.designation = designation;
     }

     // Getter and Setter Methods
     public String getName() {
          return name;
     }

     public void setName(String name) {
          this.name = name;
     }

     public double getSalary() {
          return salary;
     }

     public void setSalary(double salary) {
          this.salary = salary;
     }

     public String getDesignation() {
          return designation;
     }

     public void setDesignation(String designation) {
          this.designation = designation;
     }
```

```
    // Method to display employee details
    public void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
        System.out.println("Designation: " + designation);
    }
}



//Manager.java
package employee;

public class Manager extends Employee {
    private double bonus;

    // Constructor
    public Manager(String name, double salary, String designation,
            double bonus) {
        super(name, salary, designation);
        this.bonus = bonus;
    }

    // Getter and Setter Methods
    public double getBonus() {
        return bonus;
    }

    public void setBonus(double bonus) {
        this.bonus = bonus;
    }

    // Overridden displayDetails to include bonus
    @Override
    public void displayDetails() {
        super.displayDetails();
        System.out.println("Bonus: " + bonus);
    }

    // Method to calculate total salary (Salary + Bonus)
    public double calculateSalary() {
        return getSalary() + bonus;
    }
}

//HR.java
```

```
package employee;

public class HR extends Employee {
     private double allowance;

     // Constructor
     public HR(String name, double salary, String designation, double allowance) {
          super(name, salary, designation);
          this.allowance = allowance;
     }

     // Getter and Setter Methods
     public double getAllowance() {
          return allowance;
     }

     public void setAllowance(double allowance) {
          this.allowance = allowance;
     }

     // Overridden displayDetails to include allowance
     @Override
     public void displayDetails() {
          super.displayDetails();
          System.out.println("Allowance: " + allowance);
     }

     // Method to calculate total salary (Salary + Allowance)
     public double calculateSalary() {
          return getSalary() + allowance;
     }
}

//payroll Package
//SalaryCalculator.java
package payroll;

import employee.Employee;
import employee.Manager;
import employee.HR;

public class SalaryCalculator {
     // Method to calculate salary based on employee type
     public double calculateEmployeeSalary(Employee employee) {
          if (employee instanceof Manager) {
               return ((Manager) employee).calculateSalary(); // Manager's salary
includes bonus
```

```
          } else if (employee instanceof HR) {
               return ((HR) employee).calculateSalary(); // HR's salary includes
allowance
          } else {
               return employee.getSalary(); // For general employees, no
bonus/allowance
          }
     }
}

//PayslipGenerator.java
package payroll;

import employee.Employee;
import employee.Manager;
import employee.HR;

public class PayslipGenerator {
     // Method to generate payslip for any employee
     public void generatePayslip(Employee employee) {
          System.out.println("\nGenerating Payslip for " +
                    employee.getName());
          System.out.println("--------------------------------------------------
");
          System.out.println("Name: " + employee.getName());
          System.out.println("Designation: " +
                    employee.getDesignation());
          System.out.println("Basic Salary: " + employee.getSalary());
          if (employee instanceof Manager) {
               Manager manager = (Manager) employee;
               System.out.println("Bonus: " + manager.getBonus());
               System.out.println("Total Salary: " +
                         manager.calculateSalary());
          } else if (employee instanceof HR) {
               HR hr = (HR) employee;
               System.out.println("Allowance: " + hr.getAllowance());
               System.out.println("Total Salary: " + hr.calculateSalary());
          } else {
               System.out.println("Total Salary: " + employee.getSalary());
          }
          System.out.println("--------------------------------------------------
");
     }
}

//Main.java
import employee.Employee;
import employee.Manager;
import employee.HR;
import payroll.SalaryCalculator;
import payroll.PayslipGenerator;

public class Main {
     public static void main(String[] args) {
          // Create Employee, Manager, and HR objects
          Employee employee = new Employee("John Doe", 50000,
                    "Employee");
          Manager manager = new Manager("Alice Smith", 70000,
                    "Manager", 10000);
          HR hr = new HR("Bob Johnson", 60000, "HR", 5000);
          // Create SalaryCalculator and PayslipGenerator
          SalaryCalculator salaryCalculator = new SalaryCalculator();
          PayslipGenerator payslipGenerator = new PayslipGenerator();
          // Display Employee Details
          employee.displayDetails();
          manager.displayDetails();
          hr.displayDetails();
          // Calculate and Display Payslips
          payslipGenerator.generatePayslip(employee);
          payslipGenerator.generatePayslip(manager);
          payslipGenerator.generatePayslip(hr);
     }
}
```

```
OUTPUT :
Name: John Doe
Salary: 50000.0
Designation: Employee

Name: Alice Smith
Salary: 70000.0
Designation: Manager
Bonus: 10000.0

Name: Bob Johnson
Salary: 60000.0
Designation: HR
Allowance: 5000.0

Generating Payslip for John Doe
--------------------------------------------------------------------------------
Name: John Doe
Designation: Employee
Basic Salary: 50000.0
```

```
Total Salary: 50000.0
```

```
Generating Payslip for Alice Smith
Name: Alice Smith
Designation: Manager
Basic Salary: 70000.0
Bonus: 10000.0
Total Salary: 80000.0
```

```
Generating Payslip for Bob Johnson
Name: Bob Johnson
Designation: HR
Basic Salary: 60000.0
Allowance: 5000.0
Total Salary: 65000.0
--------------------------------------------------
```

8. Create a banking application that allows users to withdraw and deposit money. Implement exception handling for scenarios like:

InsufficientFundsException when a user tries to withdraw more money than available in their account.

InvalidAmountException when the user tries to deposit or withdraw a negative or zero amount. Demonstrate how to handle these custom exceptions and notify the user with appropriate messages.

```
    import java.util.Scanner;

    class InsufficientFundsException extends Exception {
         InsufficientFundsException(String s) {
              super(s);
         }
    }

    class InvalidAmountException extends Exception {
         InvalidAmountException(String s) {
              super(s);
         }
    }

    class Bank {
         double balance;

         Bank(double balance) {
              this.balance = balance;
         }

         public void withdraw(double amount) throws InsufficientFundsException,
    InvalidAmountException {
              if (amount <= 0) {
```

```
            throw new InvalidAmountException("amount must be positive or 
greater than 0");
        }
        if (amount > balance) {
            throw new InsufficientFundsException("insuffficient funds 
remain");
        }
        balance -= amount;
        System.out.println("amount withdrawn = " + amount);
        System.out.println("current balance = " + balance);
    }

    public void deposit(double amount) throws InvalidAmountException {
        if (amount <= 0) {
            throw new InvalidAmountException("deposit valid amount");
        }
        balance += amount;
        System.out.println("amount deposited = " + amount);
        System.out.println("current balance = " + balance);
    }
}

// main class
class prac8 {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        Bank ba = new Bank(10000);
        try {
            System.out.println("enter your choice withdraw or deposit?? 
(w/d)");
            String choice = sc.next().toLowerCase();
            if (choice.equals("w")) {
                    System.out.println("enter money to be withdrawn");
                    double am = sc.nextDouble();
                    ba.withdraw(am);
               } else if (choice.equals("d")) {
                    System.out.println("enter money to be deposited");
                    double dep = sc.nextDouble();
                    ba.deposit(dep);
               } else {
                    System.out.println("Invalid choice! Please enter 'w' for
withdrawal or 'd' for deposit.");
               }
          } catch (InsufficientFundsException e) {
               System.out.println(e.getMessage());
          } catch (InvalidAmountException e) {
               System.out.println(e.getMessage());
          } finally {
               sc.close(); // Properly close the scanner object
          }
     }
}
```

OUTPUT:

```
Enter your choice: withdraw or deposit? (w/d)
W
Enter amount to be withdrawn:
3000
Amount withdrawn = 3000.0
Current balance = 7000.0
```

Q.9 Design a system where multiple users (threads) can book movie tickets simultaneously.   Implement multithreading to simulate the booking process, ensuring that:

Thread safety is maintained when accessing shared resources (e.g., the total number of available tickets).

Use synchronization to prevent multiple users from booking the same seat at the same time.

Ans.

```
Class MovieTicketBooking {

    Private int availableSeats;


    // Constructor to initialize available seats

    Public MovieTicketBooking(int seats) {

        This.availableSeats = seats;

    }


    // Synchronized method to book a ticket

    Public synchronized void bookTicket(String user, int numberOfTickets) {

        System.out.println(user + " is trying to book " + numberOfTickets + "
ticket(s).");


        // Check if enough seats are available

        If (numberOfTickets <= availableSeats) {

            System.out.println("Booking successful for " + user + ". " +
numberOfTickets + " ticket(s) booked.");

            availableSeats -= numberOfTickets;

        } else {

            System.out.println("Booking failed for " + user + ". Not enough seats
available.");

        }

        System.out.println("Seats remaining: " + availableSeats);

    }
```

```
}


// Thread class representing a user trying to book a ticket

Class UserThread extends Thread {

    Private MovieTicketBooking bookingSystem;

    Private String userName;

    Private int ticketsToBook;


    // Constructor to initialize thread details

    Public UserThread(MovieTicketBooking bookingSystem, String userName, int
ticketsToBook) {

        This.bookingSystem = bookingSystem;

        This.userName = userName;

        This.ticketsToBook = ticketsToBook;

    }


    // Run method of thread

    @Override

    Public void run() {

        bookingSystem.bookTicket(userName, ticketsToBook);

    }

}


// Main class to simulate the ticket booking process

Public class MovieTicketBookingSystem {

    Public static void main(String[] args) {

        // Initialize the booking system with 10 available seats

        MovieTicketBooking bookingSystem = new MovieTicketBooking(10);


        // Create user threads trying to book tickets
        UserThread user1 = new UserThread(bookingSystem, “User1”, 4);

        UserThread user2 = new UserThread(bookingSystem, “User2”, 5);

        UserThread user3 = new UserThread(bookingSystem, “User3”, 3);


        // Start user threads

        User1.start();

        User2.start();

        User3.start();

    }

}
```

OUTPUT:

```
User1 is trying to book 4 ticket(s).

Booking successful for User1. 4 ticket(s) booked.

Seats remaining: 6

User2 is trying to book 5 ticket(s).

Booking successful for User2. 5 ticket(s) booked.

Seats remaining: 1

User3 is trying to book 3 ticket(s).

Booking failed for User3. Not enough seats available.

Seats remaining: 1
```

Q.10 Create a library management system using a List to store book objects where:

- Each book has attributes like title, author, and genre.
- Implement functions to add new books, search for books by author or genre, and display all available books.
- Use an ArrayList to maintain the collection of books and implement sorting functionality to display books in alphabetical order.

Ans.

```
Import java.util.ArrayList;

Import java.util.Scanner;

// Book class to store attributes of each book

Class Book {

    String title;

    String author;

    String genre;

    // Constructor

    Public Book(String title, String author, String genre) {

        This.title = title;

        This.author = author;

        This.genre = genre;

    }

    // Method to display book details

    @Override

    Public String toString() {

        Return "Title: " + title + ", Author: " + author + ", Genre: " + genre;

    }

}


// Library class to manage the list of books

Class Library {

    ArrayList<Book> books = new ArrayList<>();
```

```
// Method to add a new book to the library
Public void addBook(String title, String author, String genre) {
    Book newBook = new Book(title, author, genre);
    books.add(newBook);
    System.out.println("Book added successfully!");
}

// Method to search for books by author
Public void searchByAuthor(String author) {
    Boolean found = false;
    System.out.println("\nBooks by author '" + author + "':");
    For (Book book : books) {
        If (book.author.equalsIgnoreCase(author)) {
            System.out.println(book);
            Found = true;
        }
    }
    If (!found) {
        System.out.println("No books found by this author.");
    }
}

// Method to search for books by genre
Public void searchByGenre(String genre) {
    Boolean found = false;
    System.out.println("\nBooks in genre '" + genre + "':");
    For (Book book : books) {
        If (book.genre.equalsIgnoreCase(genre)) {
            System.out.println(book);
            Found = true;
            }
        }
        If (!found) {
            System.out.println("No books found in this genre.");
        }
    }

    // Method to display all books sorted alphabetically by title using Bubble
Sort
    Public void displayAllBooks() {
        If (books.isEmpty()) {
            System.out.println("No books available in the library.");
            Return;
        }

        // Bubble Sort to sort books by title
        For (int I = 0; I < books.size() - 1; i++) {
            For (int j = 0; j < books.size() - I - 1; j++) {
                If (books.get(j).title.compareToIgnoreCase(books.get(j +
1).title) > 0) {
                    // Swap books[j] and books[j + 1]
                    Book temp = books.get(j);
                    Books.set(j, books.get(j + 1));
                    Books.set(j + 1, temp);
                }
            }
        }

        System.out.println("\nList of all available books (sorted by title):");
        For (Book book : books) {
            System.out.println(book);
        }
    }
}

// Main class
Public class LibraryManagementSystem {
    Public static void main(String[] args) {
        Library library = new Library();
        Scanner scanner = new Scanner(System.in);

        While (true) {
            System.out.println("\nLibrary Management System");
            System.out.println("1. Add New Book");
            System.out.println("2. Search Book by Author");
            System.out.println("3. Search Book by Genre");
            System.out.println("4. Display All Books");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");

            Int choice = scanner.nextInt();
            Scanner.nextLine(); // Consume newline

            Switch (choice) {
                Case 1:
                    System.out.print("Enter book title: ");
                    String title = scanner.nextLine();
                    System.out.print("Enter book author: ");
                    String author = scanner.nextLine();
                    System.out.print("Enter book genre: ");
                    String genre = scanner.nextLine();

                    Library.addBook(title, author, genre);

                    Break;


                Case 2:

                    System.out.print(“Enter author name to search: “);

                    String searchAuthor = scanner.nextLine();

                    Library.searchByAuthor(searchAuthor);

                    Break;


                Case 3:

                    System.out.print(“Enter genre to search: “);

                    String searchGenre = scanner.nextLine();

                    Library.searchByGenre(searchGenre);

                    Break;


                Case 4:

                    Library.displayAllBooks();

                    Break;


                Case 5:

                    System.out.println(“Exiting the system. Goodbye!”);

                    Scanner.close();

                    Return;


                Default:

                    System.out.println(“Invalid choice. Please try again.”);

            }

        }

    }
```

```
}
```

OUTPUT:

```
Library Management System

1. Add New Book

2. Search Book by Author

3. Search Book by Genre

4. Display All Books

5. Exit

Enter your choice: 1


Enter book title: JAVA

Enter book author: BHOOMI MAM

Enter book genre: PROGRAMMING

Book added successfully!


Library Management System

1. Add New Book

2. Search Book by Author

3. Search Book by Genre

4. Display All Books

5. Exit

Enter your choice: 4


List of all available books (sorted by title):

Title: JAVA, Author: BHOOMI MAM, Genre: PROGRAMMING
```

Que.11 Develop a system to manage student grades using a Map where:

- The student name (or ID) is the key, and their grade is the value.
- Implement features to add, remove, update, and retrieve student grades.
- Use the HashMap collection to store student data and implement operations like searching for students with the highest grade, average grade, etc.

Ans.

```
Import java.util.HashMap;

Import java.util.Map;

Import java.util.Scanner;


Public class StudentGradesSystem {

    // HashMap to store student name (or ID) as key and grade as value

    Private HashMap<String, Double> studentGrades;


    // Constructor

    Public StudentGradesSystem() {

        studentGrades = new HashMap<>();

    }


    // Method to add or update a student's grade

    Public void addOrUpdateStudent(String name, double grade) {

        studentGrades.put(name, grade);

        System.out.println("Student grade added/updated successfully!");

    }


    // Method to remove a student

    Public void removeStudent(String name) {

        If (studentGrades.containsKey(name)) {

            studentGrades.remove(name);

            System.out.println("Student removed successfully!");
        } else {
            System.out.println("Student not found.");
        }
    }

    // Method to retrieve a student's grade
    Public void getStudentGrade(String name) {
        If (studentGrades.containsKey(name)) {
            System.out.println("Grade of " + name + ": " +
studentGrades.get(name));
        } else {
            System.out.println("Student not found.");
        }
    }

    // Method to find the student(s) with the highest grade
    Public void getHighestGrade() {
        If (studentGrades.isEmpty()) {
            System.out.println("No students in the system.");
            Return;
        }

        Double highestGrade = Double.MIN_VALUE;
        For (double grade : studentGrades.values()) {
            If (grade > highestGrade) {
                highestGrade = grade;
            }
        }
        System.out.println("Students with the highest grade (" + highestGrade +
"):");
        For (Map.Entry<String, Double> entry : studentGrades.entrySet()) {
            If (entry.getValue() == highestGrade) {
                System.out.println(entry.getKey());
            }
        }
    }

    // Method to calculate the average grade
    Public void getAverageGrade() {
        If (studentGrades.isEmpty()) {
            System.out.println("No students in the system.");
            Return;
        }

        Double sum = 0;
        For (double grade : studentGrades.values()) {
            Sum += grade;
        }

        Double averageGrade = sum / studentGrades.size();
        System.out.println("Average grade of all students: " + averageGrade);
    }

    // Method to display all students and their grades
    Public void displayAllStudents() {
        If (studentGrades.isEmpty()) {
            System.out.println("No students in the system.");
            Return;
        }


        System.out.println("List of all students and their grades:");
        For (Map.Entry<String, Double> entry : studentGrades.entrySet()) {
            System.out.println("Student: " + entry.getKey() + ", Grade: " +
entry.getValue());
        }
    }


    // Main method
    Public static void main(String[] args) {
        StudentGradesSystem gradesSystem = new StudentGradesSystem();
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nStudent Grades Management System");
        System.out.println("1. Add/Update Student Grade");
        System.out.println("2. Remove Student");
        System.out.println("3. Retrieve Student Grade");
        System.out.println("4. Display All Students");
        System.out.println("5. Find Student(s) with Highest Grade");
        System.out.println("6. Calculate Average Grade");
        System.out.println("7. Exit");


        While (true) {


            System.out.print("Enter your choice: ");


            Int choice = scanner.nextInt();
            Scanner.nextLine(); // Consume newline


            Switch (choice) {
                Case 1:

                    System.out.print(“Enter student name (or ID): “);

                    String name = scanner.nextLine();

                    System.out.print(“Enter student grade: “);

                    Double grade = scanner.nextDouble();

                    gradesSystem.addOrUpdateStudent(name, grade);

                    break;


                case 2:

                    System.out.print(“Enter student name (or ID) to remove: “);

                    String removeName = scanner.nextLine();

                    gradesSystem.removeStudent(removeName);

                    break;


                case 3:

                    System.out.print(“Enter student name (or ID) to retrieve
grade: “);

                    String retrieveName = scanner.nextLine();

                    gradesSystem.getStudentGrade(retrieveName);

                    break;


                case 4:

                    gradesSystem.displayAllStudents();

                    break;


                case 5:

                    gradesSystem.getHighestGrade();

                    break;


                case 6:
                    gradesSystem.getAverageGrade();
                    break;

                case 7:
                    System.out.println(“Exiting the system. Goodbye!”);
                    Scanner.close();
                    Return;

                Default:
                    System.out.println(“Invalid choice. Please try again.”);
            }
        }
    }
}
```

OUTPUT:

```
Student Grades Management System
1. Add/Update Student Grade
2. Remove Student
3. Retrieve Student Grade
4. Display All Students
5. Find Student(s) with Highest Grade
6. Calculate Average Grade
7. Exit
Enter your choice: 1

Enter student name (or ID): Student1
Enter student grade: 85.5
Student grade added/updated successfully!
```

```
Enter your choice: 1

Enter student name (or ID): Student2

Enter student grade: 92.0

Student grade added/updated successfully!


Enter your choice: 5

Students with the highest grade (92.0):

Student2


Enter your choice: 6

Average grade of all students: 88.75
```

Q.12 Design an address book application where contacts (name, phone number, address) are stored in a file. Allow users to add, remove, update, and search for contacts. Use file I/O to save the contacts to a text file and load them when the program starts

Ans.

```
    import java.io.*;

    import java.util.*;


    class Contact {

        private String name;

        private String phoneNumber;

        private String address;


        public Contact(String name, String phoneNumber, String address) {

            this.name = name;

            this.phoneNumber = phoneNumber;

            this.address = address;

        }


        public String getName() {

            return name;

        }


        public String getPhoneNumber() {

            return phoneNumber;

        }


        public String getAddress() {

            return address;

        }


        public void setPhoneNumber(String phoneNumber) {

            this.phoneNumber = phoneNumber;
```

```java
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return name + "," + phoneNumber + "," + address;
    }

    public static Contact fromString(String contactStr) {
        String[] parts = contactStr.split(",");
        return new Contact(parts[0], parts[1], parts[2]);
    }
}

public class AddressBook {
    private static final String FILE_NAME = "contacts.txt";
    private List<Contact> contacts = new ArrayList<>();

    // Load contacts from file
    public void loadContacts() {
        try (BufferedReader br = new BufferedReader(new FileReader(FILE_NAME))) {
            String line;
            while ((line = br.readLine()) != null) {
                contacts.add(Contact.fromString(line));
            }
        } catch (IOException e) {
            System.out.println("Error loading contacts: " + e.getMessage());
        }
    }
    // Save contacts to file
    public void saveContacts() {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(FILE_NAME))) {
            for (Contact contact : contacts) {
                bw.write(contact.toString());
                bw.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error saving contacts: " + e.getMessage());
        }
    }

    // Add a new contact
    public void addContact(String name, String phoneNumber, String address) {
        contacts.add(new Contact(name, phoneNumber, address));
        System.out.println("Contact added successfully.");
    }

    // Remove a contact by name
    public void removeContact(String name) {
        boolean removed = contacts.removeIf(contact ->
contact.getName().equalsIgnoreCase(name));
        if (removed) {
            System.out.println("Contact removed successfully.");
        } else {
            System.out.println("Contact not found.");
        }
    }

    // Update a contact by name
    public void updateContact(String name, String newPhoneNumber, String newAddress)
{
        for (Contact contact : contacts) {
            if (contact.getName().equalsIgnoreCase(name)) {
                contact.setPhoneNumber(newPhoneNumber);
                contact.setAddress(newAddress);
                System.out.println("Contact updated successfully.");
                return;
            }
        }
        System.out.println("Contact not found.");
    }

    // Search for a contact by name
    public void searchContact(String name) {
        for (Contact contact : contacts) {
            if (contact.getName().equalsIgnoreCase(name)) {
                System.out.println("Contact found: " + contact);
                return;
            }
        }
        System.out.println("Contact not found.");
    }

    // Display all contacts
    public void displayContacts() {
        if (contacts.isEmpty()) {
            System.out.println("No contacts available.");
        } else {
            System.out.println("Address Book Contacts:");
            for (Contact contact : contacts) {
                System.out.println(contact);
            }
        }
    }

    // Main menu
    public void menu() {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\nAddress Book Menu:");
            System.out.println("1. Add Contact");
            System.out.println("2. Remove Contact");
            System.out.println("3. Update Contact");
            System.out.println("4. Search Contact");
            System.out.println("5. Display All Contacts");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();  // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter phone number: ");
                    String phoneNumber = scanner.nextLine();
                    System.out.print("Enter address: ");
                    String address = scanner.nextLine();
                    addContact(name, phoneNumber, address);
                    saveContacts();
                    break;
                case 2:
                    System.out.print("Enter name to remove: ");
                    name = scanner.nextLine();
                    removeContact(name);
                    saveContacts();
                    break;
                case 3:
                    System.out.print("Enter name to update: ");
                    name = scanner.nextLine();
                    System.out.print("Enter new phone number: ");
                    phoneNumber = scanner.nextLine();
                    System.out.print("Enter new address: ");
                    address = scanner.nextLine();
                    updateContact(name, phoneNumber, address);
                    saveContacts();
                    break;
                case 4:
                    System.out.print("Enter name to search: ");
                    name = scanner.nextLine();
                    searchContact(name);
                    break;
                case 5:
                    displayContacts();
                    break;
                case 6:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
    public static void main(String[] args) {
        AddressBook addressBook = new AddressBook();
        addressBook.loadContacts();
        addressBook.menu();
    }
}
```

OUTPUT:

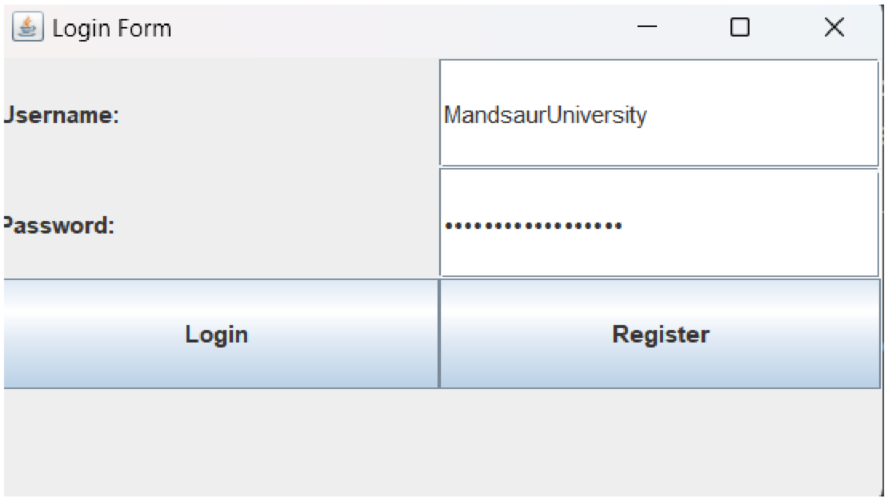
```
Address Book Menu:
1. Add Contact
2. Remove Contact
3. Update Contact
4. Search Contact
5. Display All Contacts
6. Exit
```

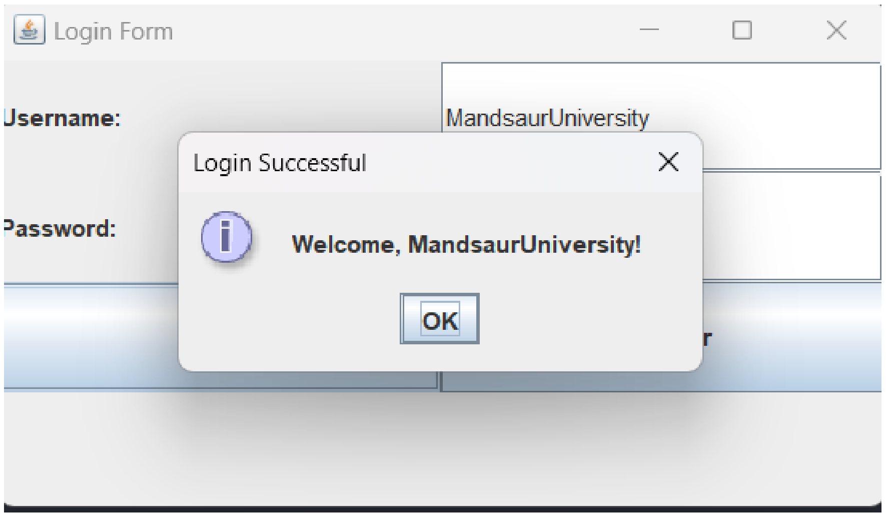
Q.13 Design a login form using Swings where users enter their username and password. On successful login, show a welcome message; on failure, show an error message. Implement a registration form and validate if the username is already taken.

Ans.

```
    import javax.swing.*;

    import java.awt.*;

    import java.awt.event.*;

    import java.io.*;

    import java.util.HashMap;

    import java.util.Map;


    public class UserAuthentication {

        private JFrame frame;

        private JTextField usernameField;

        private JPasswordField passwordField;

        private JTextField regUsernameField;

        private JPasswordField regPasswordField;

        private static final String FILE_NAME = "users.txt";

        private Map<String, String> users = new HashMap<>();


        public UserAuthentication() {

            loadUsers();

            showLoginForm();

        }


        // Load users from the file

        private void loadUsers() {

            try (BufferedReader br = new BufferedReader(new FileReader(FILE_NAME))) {

                String line;

                while ((line = br.readLine()) != null) {

                    String[] parts = line.split(",");
                    if (parts.length == 2) {
                        users.put(parts[0], parts[1]);
                    }
                }
            } catch (IOException e) {
                System.out.println("Error loading user data: " + e.getMessage());
            }
        }

        // Save a new user to the file
        private void saveUser(String username, String password) {
            try (BufferedWriter bw = new BufferedWriter(new FileWriter(FILE_NAME, true)))
    {
                bw.write(username + "," + password);
                bw.newLine();
            } catch (IOException e) {
                System.out.println("Error saving user data: " + e.getMessage());
            }
        }

        // Show the login form
        private void showLoginForm() {
            frame = new JFrame("Login Form");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 200);
            frame.setLayout(new GridLayout(4, 2));

            JLabel usernameLabel = new JLabel("Username:");
            usernameField = new JTextField();
            JLabel passwordLabel = new JLabel("Password:");
            passwordField = new JPasswordField();
            JButton loginButton = new JButton("Login");
            JButton registerButton = new JButton("Register");

            loginButton.addActionListener(e -> loginUser());
            registerButton.addActionListener(e -> showRegistrationForm());

            frame.add(usernameLabel);
            frame.add(usernameField);
            frame.add(passwordLabel);
            frame.add(passwordField);
            frame.add(loginButton);
            frame.add(registerButton);

            frame.setVisible(true);
        }

        // Show the registration form
        private void showRegistrationForm() {
            frame.dispose();
            frame = new JFrame("Registration Form");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(400, 200);
            frame.setLayout(new GridLayout(4, 2));

            JLabel regUsernameLabel = new JLabel("Enter Username:");
            regUsernameField = new JTextField();
            JLabel regPasswordLabel = new JLabel("Enter Password:");
            regPasswordField = new JPasswordField();

            JButton registerButton = new JButton("Register");
            JButton backButton = new JButton("Back");
            registerButton.addActionListener(e -> registerUser());
            backButton.addActionListener(e -> {
                frame.dispose();
                showLoginForm();
            });

            frame.add(regUsernameLabel);
            frame.add(regUsernameField);
            frame.add(regPasswordLabel);
            frame.add(regPasswordField);
            frame.add(registerButton);
            frame.add(backButton);

            frame.setVisible(true);
        }

        // Validate login credentials
        private void loginUser() {
            String username = usernameField.getText().trim();
            String password = new String(passwordField.getPassword()).trim();

            if (users.containsKey(username) && users.get(username).equals(password)) {
                JOptionPane.showMessageDialog(frame, "Welcome, " + username + "!", "Login
    Successful", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog(frame, "Invalid username or password!",
    "Login Failed", JOptionPane.ERROR_MESSAGE);
            }
        }

        // Register a new user
        private void registerUser() {
            String username = regUsernameField.getText().trim();

            String password = new String(regPasswordField.getPassword()).trim();


            if (username.isEmpty() || password.isEmpty()) {

                JOptionPane.showMessageDialog(frame, "Please fill in all fields!",
    "Registration Failed", JOptionPane.ERROR_MESSAGE);

                return;

            }


            if (users.containsKey(username)) {

                JOptionPane.showMessageDialog(frame, "Username already taken. Please
    choose a different username.", "Registration Failed", JOptionPane.ERROR_MESSAGE);

            } else {

                users.put(username, password);

                saveUser(username, password);

                JOptionPane.showMessageDialog(frame, "Registration successful!",
    "Success", JOptionPane.INFORMATION_MESSAGE);

                frame.dispose();

                showLoginForm();

            }

        }


        public static void main(String[] args) {

            SwingUtilities.invokeLater(UserAuthentication::new);

        }

    }
```

OUTPUT:





Welcome, MandsaurUniversity!

Q.14 Create a servlet-based product management system where users can:

Add new products (name, price, quantity) via an HTML form.

List all products on a separate page.

Ans.

1. Project Structure:

```
ProductManagementSystem/
├── src/
│   └── com/
│       └── example/
│           ├── servlet/
│           │   ├── AddProductServlet.java
│           │   └── ListProductServlet.java
│           └── model/
│               └── Product.java
├── WebContent/
│   ├── index.html
│   ├── addProduct.html
│   ├── listProducts.jsp
│   └── WEB-INF/
│       └── web.xml
```

Product.java (Model Class):

```java
package com.example.model;

public class Product {
    private String name;
    private double price;
    private int quantity;

    public Product(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}
```

AddProductServlet.java (Servlet for Adding Products):

```java
package com.example.servlet;

import com.example.model.Product;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class AddProductServlet extends HttpServlet {

    private static List<Product> products = new ArrayList<>();

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        String name = request.getParameter("name");
        double price = Double.parseDouble(request.getParameter("price"));
        int quantity = Integer.parseInt(request.getParameter("quantity"));

        Product product = new Product(name, price, quantity);
        products.add(product);

        response.sendRedirect("listProducts");
    }
}
```

ListProductServlet.java (Servlet for Listing Products):

```
package com.example.servlet;

import com.example.model.Product;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.util.List;
```

```
public class ListProductServlet extends HttpServlet {

    private static List<Product> products = AddProductServlet.products;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        request.setAttribute("products", products);
        request.getRequestDispatcher("listProducts.jsp").forward(request, response);
    }
}
```

addProduct.html (HTML Form for Adding Products):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add Product</title>
</head>
<body>
    <h2>Add New Product</h2>
    <form action="addProduct" method="post">
        <label for="name">Product Name:</label>
        <input type="text" id="name" name="name" required><br><br>

        <label for="price">Price:</label>
        <input type="number" step="0.01" id="price" name="price" required><br><br>
        <label for="quantity">Quantity:</label>
        <input type="number" id="quantity" name="quantity" required><br><br>


        <button type="submit">Add Product</button>
    </form>
    <br>
    <a href="listProducts">View All Products</a>
</body>
</html>
```

listProducts.jsp (JSP for Listing Products):

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Product List</title>
</head>
<body>
    <h2>Product List</h2>
    <table border="1">
        <tr>
            <th>Name</th>
            <th>Price</th>
            <th>Quantity</th>
        </tr>
        <c:forEach var="product" items="${products}">
```

```
            <tr>
                <td>${product.name}</td>
                <td>${product.price}</td>
                <td>${product.quantity}</td>
            </tr>
        </c:forEach>
    </table>
    <br>
    <a href="addProduct.html">Add Another Product</a>
</body>
</html>
```

web.xml (Servlet Configuration):

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <servlet>
        <servlet-name>AddProductServlet</servlet-name>
        <servlet-class>com.example.servlet.AddProductServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AddProductServlet</servlet-name>
        <url-pattern>/addProduct</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>ListProductServlet</servlet-name>
        <servlet-class>com.example.servlet.ListProductServlet</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ListProductServlet</servlet-name>
        <url-pattern>/listProducts</url-pattern>
    </servlet-mapping>
</web-app>
```

index.html (Optional Home Page):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Product Management</title>
</head>
<body>
    <h1>Welcome to Product Management System</h1>
    <a href="addProduct.html">Add Product</a><br>
    <a href="listProducts">View All Products</a>
</body>
</html>
```