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));
   }
```

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.**

```
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);


    }
}
```

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.

```
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 inventory
    }

    // 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
inventory
                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
bounds
    }
}

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 from user


        // 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


            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
inventory
                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); //
Remove items from inventory
                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.");
        }
    } } }
```

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;
}
}
```

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));
}
}
```

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();
 }
}
}
```

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);
   }
}
```

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
    }
}
}
```