DCIT 201: PROGRAMMING I
ASSIGNMENT 2

INSTRUCTIONS: Answer one (1) question from each section

ENCAPSULATION

QUESTION 1.

You are required to create a `CommissionEmployee` class in PHP to represent an employee who is compensated based on a percentage of their total sales. The class must enforce encapsulation principles.

Class

`CommissionEmployee`

Attributes

- `firstName` (string): The employee's first name.
- `lastName` (string): The employee's last name.
- `socialSecurityNumber` (string): The employee's social security number.
- `grossSales` (float): The employee's total gross sales (must be ≥ 0.0).
- `commissionRate` (float): The percentage of gross sales paid as commission (must be between 0.0 and 1.0).

Methods

Constructor - Initializes all class attributes.

`earnings()` - Returns the employee's earnings using the formula: `grossSales` * `commissionRate`

Implementation Tasks

A. Implement the `CommissionEmployee` class in PHP with encapsulation.

B. Create an instance of the `CommissionEmployee` class.

C. Update the employee's `grossSales` and `commissionRate`, then display the updated details.

D.. Calculate and display the employee's earnings using the `earnings()` method.

QUESTION 2.

You are required to implement a Library Management System in PHP using the principles of encapsulation. The system should manage books and library members, allowing members to borrow and return books while tracking book availability.

Class 1

`Book Class`

Encapsulate Class Attributes

- `bookId` (string): Unique identifier for the book.
- `title` (string): Title of the book.
- `author` (string): Author of the book.
- `availableCopies` (int): Number of available copies of the book.

Methods

Constructor: Initializes all attributes.

Getters and Setters: Retrieve and update attributes.

`borrowBook()`: Reduces available copies by 1 if a copy is available.

`returnBook()`: Increases available copies by 1.

Class 2

`Member Class`

Encapsulated Class Attributes

- `memberId` (string): Unique identifier for the library member.
- `name` (string): Name of the member.

- `borrowedBooks` (Book|null)): List of books borrowed by the member.

Methods

Constructor: Initializes `memberId` and `name`.

Getters and Setters: Retrieve and update attributes.

`borrowBook(Book $book)`: Allows borrowing if the member has no book and the book is available.

`returnBook()`: Returns the borrowed book.

Class 3

`Library Class`

Class Attributes

- `book` (Book|null): Unique identifier for the library member.
- `member` (Member | null): Name of the member.

Methods:

`setBook(Book $book)`: Assigns a book to the library.

`setMember(Member $member)`: Registers a member.

`borrowBook()`: Allows the member to borrow the book if available.

`returnBook()`: Allows the member to return the book.

`displayBook()`: Displays book details and availability.

Implementation Tasks

A. Implement the `Book`, `Member`, and `Library` classes in PHP with encapsulation.
B. Write PHP code to do the following:

Creates **one book** and **one member**.
Simulates **borrowing** and **returning** the book.

QUESTION 3.

You are required to design a **Hospital Management System** in PHP using **encapsulation** principles. The system should securely manage patient and doctor details while ensuring proper validation.

Class 1

`Patient Class`

Encapsulated Class Attributes:

- `patientId` (string): Unique identifier for the patient.
- `name` (string): Name of the patient.
- `age` (int): Age of the patient.
- `diagnosis` (string): Current diagnosis of the patient.

Methods

Constructor**:** Initializes all attributes with validation.

Getters and Setters: Retrieve and update all attributes

`setAge(int $age)`: Ensures age is greater than 0; otherwise, prints "Invalid age."

`setDiagnosis(string $diagnosis)`: Ensures diagnosis is not empty; otherwise, prints "Diagnosis cannot be empty."

`updateDiagnosis(string $newDiagnosis)`: Updates diagnosis and prints "Diagnosis updated successfully to: <newDiagnosis>."

Class 2

`Doctor Class`

Encapsulated Class Attributes

- `doctorId` (string): Unique identifier for the doctor.
- `name` (string): Name of the doctor.

- `specialization` (string): Doctor's field of specialization.

Methods

Constructor**:** Initializes `doctorId`, `name`, and `specialization`.

Getters and Setters: Initialize and retrieve attributes

`treatPatient()`: Logs `patientID` and `diagnosis` and prints "Patient <patientId> treated for <diagnosis> successfully. "

Implementation Tasks

A. Implement the `Patient` and `Doctor` classes in PHP with encapsulation.
B. Create a `Patient` object with the following details:
Patient ID: `"P001"`
Name: `"John Smith"`
Age: `45`
Diagnosis: `"Fever"`

C. Creates a `Doctor` object with the following details:
Doctor ID: `"D101"`
Name: `"Dr. Alice"`
Specialization: `"General Medicine"`

D. Write PHP code to perform the following operations:

Update the patient's diagnosis to "Flu".
Treat the patient and log the treated patient info.

QUESTION 4

You need to design an **Airline Reservation System** in PHP using **encapsulation** to securely manage flight details, passenger information, and reservation operations.

Class 1
`Flight Class`

Encapsulated Class Attributes:

- `flightNumber` (string): Unique identifier for the flight.
- `destination` (string): Flight destination.
- `capacity` (int): Total number of seats.
- `bookedSeats` (int): Seats currently booked.

Methods

Constructor: Initializes all attributes.

Getters and Setters: Intialialize and retrieve attribute values

`setCapacity(int $capacity)`: Ensures `capacity` is greater than or equal to `bookedSeats`.

`bookSeat()`: Increases `bookedSeats` by 1 if seats are available.

`cancelSeat()`: Decreases `bookedSeats` by 1 if at least one booking exists

Class 2
`Passenger Class`

Encapsulated Class Attributes:

- `passengerId` (string): Unique identifier for the passenger.
- `name` (string): Passenger's name.

- `contactNumber` (string): Passenger's contact number.
- `flightBooked` (string): Flight number of the booked flight (initially `null`).

Methods

Constructor: Initializes all attributes.

Getters and Setters: Initialize and retrieve all attribute values

`setContactNumber(string contactNumber)`: Ensures the contact number is 10 digits.

`bookFlight(string $flightNumber)`: Assigns `flightBooked` to `flightNumber` if not already booked.

`cancelFlight()`: Sets `flightBooked` to `null` if a booking exists.

Implementation Tasks

A. Implement the `Flight` and `Passenger` classes in PHP using encapsulation.
B. Creates a Flight object with
Flight Number: `"AI101"`
Destination: `"New York"`
Capacity: `200`
Booked Seats: `150`

C. Creates a Passenger object with
Passenger ID: `"P123"`
Name: `"Sarah Connor"`
Contact Number: `"9876543210"`

D. Write PHP code to perform the following operations
Book a seat for the passenger and update flight details.
Attempt to book again (should not allow duplicate booking).
Cancel the booking and update flight details.
Attempt to cancel again (should not allow cancellation if no booking exists).
Attempt to set an invalid flight capacity (less than booked seats).
Attempt to set an invalid contact number (not 10 digits).

QUESTION 5

You are tasked with designing a **Banking System** that manages customer accounts, transactions, and financial analytics using encapsulation principles.

Class 1

`BankAccount Class`

Encapsulated Class Attributes:

- `accountNumber` (string): The account number of the bank account
- `accountHolder` (string): The name of the account holder.
- `balance` (float): The current balance in the account (must be ≥ 0.0).
- `interestRate` (float): The annual interest rate for the account (must be between 0.0 and 1.0).

Methods

Constructor: Initializes all class attributes.

`deposit(float $amount)`: Adds the specified amount to the balance.

`withdraw(float $amount)`: Deducts the specified amount from the balance (if sufficient funds are available).

`calculateInterest`: Returns the annual interest earned using the formula: balance * interestRate.

`getBalance`: Returns the current balance.

Implementation Tasks

A. Implement the BankAccount class with encapsulation in PHP.
B. Create an instance of the BankAccount class.
C. Update the account balance and display updated details

D. Calculate and display annual interest earned.

INHERITANCE

QUESTION 1

Extend a CommissionEmployee class into a subclass called BasePlusCommissionEmployee. The system should manage employees who earn based on commission, and those who have a base salary in addition to commissions.

Base Class

`CommissionEmployee`

Class Attributes

`firstName` (string) – The employee's first name.
`lastName` (string) – The employee's last name.
`socialSecurityNumber` (string) – A unique identifier for the employee.
`grossSales` (float) – The employee's total sales amount.
`commissionRate` (float) – The commission percentage (between 0 and 1).

Methods

Constructor: Initializes all attributes
`earnings()`: Returns the calculated commission (`grossSales * commissionRate`).
`display()`: Outputs employee details, including earnings.

Derived Class

`BasePlusCommission`

Class Attributes

Inherits all fields from `CommissionEmployee`.
`baseSalary` (float) – A guaranteed base salary for the employee.

Constructor

Calls the superclass constructor to initialize inherited fields.
Initializes `baseSalary`

Methods

`earnings()`: Calculates total earnings as `baseSalary + (grossSales * commissionRate)`.
`setBaseSalary(float $newSalary)`: Updates the base salary with validation.
`display()`: Outputs employee details, including base salary and total earnings.

Implementation Taaks

A. Create an instance of Commission-Only Employees in PHP
B. Create an instance of Base Salary + Commission Employees
C. Calculate and Display Earnings on each employee
D. Update `baseSalary` for a `BasePlusCommissionEmployee` instance and print the update earnings.

QUESTION 2

You are tasked with developing a **Vehicle Rental Management System** using **inheritance** in PHP. The system should manage different types of vehicles.

Base Class

`Vehicle`

Class Attributes

`vehicleId` (string) – Unique identifier.
`brand` (string) – Brand name.
`model` (string) – Model name.
`isAvailable` (bool) – Availability status.

Constructor

Initializes `vehicleId`, `brand`, `model`, and sets `isAvailable` to `true` (default).

Methods

`rentVehicle()`:

- If `isAvailable`, marks it as `false` and prints "Vehicle <vehicleId> rented successfully."Otherwise, prints "Vehicle <vehicleId> is not available."

`returnVehicle()`: Marks `isAvailable` as `true` and prints "Vehicle <vehicleId> returned successfully."

Derived Class

`Car`

Class Attributes

`seatingCapacity` (int) – Number of seats.

Constructor

Initializes `vehicleId`, `brand`, `model`, and `seatingCapacity`.

Methods

`calculateCarRentalCost(int $days)`:

- Computes rental cost using the formula:
  `1000 * days + seatingCapacity * 50`
- Prints "Rental cost for <days> days: <calculated amount>"

Implementation Tasks

A. Create a Car Instance
B. Rent and return a vehicle
C. Calculate Rental Cost

QUESTION 3

You are tasked with designing an **E-Commerce System** to manage different types of users and orders using inheritance principles.

Base Class

`User`

Class Attributes

`userId` (string) – Unique identifier.

`name` (string) – User's name.

Constructor

Initializes `userId` and `name`

Methods

`printUserDetails()`: Displays `userId` and `name`.

Derived Class

`Customer - Extends User Class`

Class Attributes

`email` (string) – Customer's email.

`cart` (string) – A comma-separated list of items.

Constructor

Initializes `userId`, `name`, `email`, and sets an empty `cart`.

Methods

`addItemToCart(string $item)`: Adds an item to the `cart`. Prints "<item> added to cart."

`viewCart()`: Displays all items in the cart.

Derived Class

`Order - Extends Customer Class`

Class Attributes

`orderId` (string) – Unique order identifier.

`orderDetails` (string) – Items in the order (comma-separated).

Constructor

Initializes `orderId` along with inheriting Customer's attributes.

Methods

`printOrderDetails()`: Displays Order ID, Customer Info (ID, Name, Email), and Items Ordered.

Implementation Tasks

A. Create Users:

```
Customer("C001", "Alice", "alice@example.com")
Customer("C002", "Bob", "bob@example.com")
```

B.Customers Add Items to Cart:

```
Alice: "Laptop", "Mouse"
Bob: "Smartphone", "Headphones"
```

C. Place Orders:

Alice places an order with **her cart items.**

Bob places an order with **his cart items.**

E. View Users and Orders:

Print details using `printUserDetails()` and `printOrderDetails()`

QUESTION 4

You are tasked with designing a **Hospital Management System** to manage different types of staff and their roles using inheritance principles.

Base Class

`Staff`

Class Attributes

`staffId` (string) – Unique staff identifier.
`name` (string) – Staff member's name.
`department` (string) – Assigned department.

Constructor

Initializes `staffId`, `name`, and `department`.

Methods

`displayStaffDetails():`
`Prints "Staff ID: <staffId>, Name: <name>, Department: <department>"`

Derived Class

`Doctor - Extends Staff, representing a doctor with a specialization.`

Class Attributes

`specialization` (string) – Doctor's area of expertise.
`yearsOfExperience` (int) – Number of years practiced.

Constructor

Initializes `staffId`, `name`, `department`, `specialization`, and `yearsOfExperience`

Methods

`displayDoctorDetails()`:
Prints "Doctor ID: <staffId>, Name: <name>, Department: <department>, Specialization: <specialization>, Experience: <yearsOfExperience> years"

Derived Class

`Nurse` - Extends `Staff`, representing a nurse with shift details..

Class Attributes

`shift` (string) – Assigned shift (e.g., "Day", "Night").

`patientsAssigned` (int) – Number of patients under care.

Constructor

Initializes `staffId`, `name`, `department`, `specialization`, and `yearsOfExperience`

Methods

`displayNurseDetails()`: Prints

"Nurse ID: <staffId>, Name: <name>, Department: <department>, Shift: <shift>, Patients Assigned: <patientsAssigned>"

Independent Class

`HospitalManagementSystem`

Methods

`registerDoctor(Doctor $doctor)`: Calls `displayDoctorDetails()`.

`registerNurse(Nurse $nurse)`: Calls `displayNurseDetails()`.

Implementation Tasks

A.Create Staff Members:

```
Doctor("S001", "Dr. Smith", "Cardiology", "Cardiology", 15")
Doctor("S002", "Dr. Lee", "Neurology", "Neurology", 8")
Nurse("S003", "Nurse Kelly", "Emergency", "Night", 5")
```

B. Register and Display Staff Details:

Call `registerDoctor()` for each doctor.

Call `registerNurse()` for the nurse

QUESTION 5

You are tasked with designing a **Restaurant Management System** to manage various staff roles using inheritance principles. This system will focus on role-specific responsibilities and task delegation.

Base Class

`Employee - Represents a general restaurant employee.`

Class Attributes

`employeeId` (string) – Unique identifier.
`name` (string) – Employee's name.

Constructor

Initializes `employeeId` and `name`.

Methods

`displayEmployeeDetails()`: Prints
"Employee ID: <employeeId>, Name: <name>".

Derive Class

`Chef - Represents a chef with additional details.`

Class Attributes

`specialty` (string) – Type of cuisine prepared.

Constructor

Initializes `employeeId`, `name`, and `specialty`.

Methods

`displayChefDetails()`: Prints
"Chef ID: <employeeId>, Name: <name>, Specialty: <specialty>"
`prepareDishes()`: Prints
"Chef <name> is preparing <specialty> dishes."

Derive Class

`Waiter` - Represents a Waiter with additional details.

Class Attributes

`assignedSection` (string) – Section of responsibility.

Constructor

Initializes `employeeId`, `name`, and `assignedSection`.

Methods

`displayWaiterDetails()`: Prints

"Waiter ID: <employeeId>, Name: <name>, Section: <assignedSection>"

`serveCustomers()`: Prints

"Waiter <name> is serving customers in the <assignedSection> section."

Independent Class

`RestaurantManagementSystem` - Handles employee tasks.

Methods

`assignChefTask(Chef $chef)`: Calls `prepareDishes()`.

`assignWaiterTask(Waiter $waiter)`: Calls `serveCustomers()`.

Implementation Tasks

A. Create Employees:

```
Chef("E001", "Alice", "Italian")
Waiter("E002", "Bob", "Outdoor")
```

B. Assign Tasks:

Call `assignChefTask()` for the chef.

Call `assignWaiterTask()` for the waiter.

POLYMORPHISM

QUESTION 1

You are tasked with designing a **Transportation Management System** to handle various types of vehicles and their operations using polymorphism. The system must demonstrate both **runtime polymorphism** (method overriding) and **compile-time polymorphism** (method overloading).

Abstract Class

`Vehicle`

Class Attributes

`vehicleId` (string) – Unique identifier

`model` (string) – Model name

`fuelLevel` (float) – Fuel in liters

Methods

`refuel(float $liters)`: Adds fuel and prints the updated level.

`calculateRange()`: Abstract method to be implemented in subclasses.

Derive Class

`Car - Extends Vehicle`

Class Attributes

`fuelEfficiency` (float) – Km per liter

Methods

Overrides `calculateRange():` range = fuelLevel * fuelEfficiency.

Independent Class

`TransportationManager`

Method

`operateVehicle(Vehicle $vehicle)`: Calls `calculateRange()`, demonstrating polymorphism.

Implementation Tasks

A. Create Vehicles:

`Car("C001", "Sedan", 50, 15)`

B. Refuel Sedan

C. Use `operateVehicle()` to process all

QUESTION 2

You are tasked with designing a **Banking System** that demonstrates **both compile-time (method overloading)** and **run-time polymorphism (method overriding)**. The system should handle different types of accounts and operations.

Base Class

`BankAccount`

Class Attributes

`accountHolderName` (string) – Name of the account holder

`accountNumber` (string) – Unique account number

`balance` (float) – Current account balance

Methods

`deposit(float $amount)`: Increases balance and prints the updated balance.

`deposit(float $amount, string $note)`: Overloaded method that also prints the transaction note.

`withdraw(float $amount)`: Decreases balance if funds are available, else prints an error.

`displayAccountDetails()`: Prints account details (overridden in subclasses).

Derive Class

`savingsAccount - Extends Bank Account`

Class Attributes

`interestRate` (float) – Annual interest rate

Methods

Overrides `withdraw(float $amount)`: Prevents withdrawal if balance falls below $100.

`calculateInterest()`: Computes annual interest and displays it.

Overrides `displayAccountDetails()`: Includes `interestRate` in account details

Implementation Tasks

A. Create Accounts:

`SavingsAccount("Alice", "SA123", 500, 3%)`

B. Deposit to savings accounts using one and two arguments.

C. Withdraw from `SavingsAccount`, testing the minimum balance limit.

D. Display account details for both.

QUESTION 3

You are tasked with designing an **E-Commerce System** to handle various types of products and dynamic pricing using polymorphism.

Abstract Class

`Product`

Class Attributes

`productId` (string) – Unique product ID

`productName` (string) – Name of the product

`basePrice` (float) – Original price of the product

Methods

`applyDiscount(float $percentage)`: Reduces price by a given percentage.

`calculateFinalPrice()`: Abstract method for product-specific price calculations.

Derive Class

`Cart`

Methods

`addProduct(Product $product)`: Adds a product to the cart.

`calculateTotalPrice(...$products)`: Overloaded method to calculate total cost for multiple products.

Implementation Tasks

A. Create Products

```
Electronics("E001", "Laptop", 1000.0, 24)
Clothing("C001", "Winter Jacket", 200.0, "M", 20.0)
```

B. Apply 10% discount to Laptop

C. Calculate final price of Laptop and Winter Jacket.

D. Add both to Cart and calculate total price.

QUESTION 4

You are tasked with designing a **Staff Management System** for an organization. The system must demonstrate **polymorphism** to calculate staff Annual salary

Abstract Class

`StaffMember`

Class Attributes

`name` (string) – Staff member's name

`id` (string) – Unique identifier

Methods

`getAnnualSalary()`: Abstract method to calculate annual pay.

`toString()`: Returns staff details.

Derive Class

`Staff`

Methods

`addStaff(StaffMember $staff)`: Adds a staff member.

`getAnnualSalary(int $monthlySalary)`: Computes total monthly salary.

`displayStaff()`: Prints staff details.

Implementation Tasks

A. Create a staff Member
B. Display Staff Details
C. Calculate total Annual Salary

QUESTION 5

You are tasked with designing a Church Management System that demonstrates method overriding (runtime polymorphism) and method overloading (compile-time polymorphism).

Abstract Class

`StaffMember`

Class Attributes

`name` (String): Member's name.

`memberId` (String): Unique identifier.

Constructor

Initializes `name` and `memberId`.

Methods

`getContribution()`: Returns `0.0` (default for general members).

`giveOffering(double amount)`: Prints `"Offering given: <amount>"`.

Derive Class

`Pastor - Extends ChurchMember`

Class Attributes

`tithe` (double): Monthly tithe contribution.

Constructor

Initializes `name`, `memberId`, and `tithe`

Methods

Override `getContribution()`: Returns `tithe`.

Overload `giveOffering(double amount, String message)`: Prints `"Offering given: <amount>. Note: <message>"`.

ABSTRACTION

QUESTION 1

You are tasked with implementing **abstraction** in a payroll system using PHP. The system should define an abstract `Employee` class as a base for different types of employees.

Abstract Class

`Employee`

Encapsulated Class Attributes

`name` (String): The name of the employee.

`employeeId` (String): The unique ID for the employee

Constructor

A two-argument constructor to initialize `$name` and `$employeeId`

Methods

Getter methods: Provide access to `$name` and `$employeeId`.

Abstract method `calculatePay()`: Must be implemented by subclasses.

Derive Class

`FullTimeEmployee - Extends Employee`

Encapsulated Class Attributes

`$salary` (double): The full-time employee's salary.

Constructor

Initializes `name`, `employeeId`, and `salary`

Methods

Implement `calculatePay()` to return: `"FullTimeEmployee Pay: <salary>"`.

Getter method for `$salary`.

Implementation Tasks

A. Create the abstract `Employee` class with the specified attributes and methods in PHP

B. Implement the `FullTimeEmployee` subclass and define `calculatePay()`.

C. Instantiate a `FullTimeEmployee` object.

D. Display the employee's details.

E. Call `calculatePay()` to test different salary values.

QUESTION 2

You are tasked with designing a **Medical Record Management System** using PHP. The system should enforce **abstraction** by defining a base class for different types of medical personnel while allowing specific roles and specializations to vary.

Abstract Class

`MedicalPersonnel`

Encapsulated Class Attributes

`name` (String): The name of the personnel.

`id` (String): A unique ID for the personnel.

Constructor

A two-argument constructor to initialize `$name` and `$employeeId`

Methods

Getter methods: Provide access to `$name` and `$id`.

`performDuties()`: Abstract method that defines the duties of the personnel.

`getSpecialization()`: Abstract method that defines the personnel's specialization.

`displayDetails()`: Concrete method that Prints the name and ID of the personnel.

.

Derive Class

`Doctor - Extends MedicalPersonnel`

Encapsulated Class Attributes

`specialization` (String): The medical specialty (e.g., Cardiologist, Pediatrician).

Constructor

A three-argument constructor to initialize `$name`, `$id`, and `$specialization`.

Methods

Implement `performDuties()` to return:

"Doctor <name>: Diagnoses patients, prescribes medication, and conducts surgeries."

Implement `getSpecialization()` to return `$specialization`.

Derive Class

`Nurse - Extends MedicalPersonnel`

Encapsulated Class Attributes

`department` (String): The department the nurse works in (e.g., ICU, Emergency).

Constructor

A three-argument constructor to initialize `$name`, `$id`, and `$department`.

Methods

Implement `performDuties()` to return:

"Nurse <name>: Provides patient care, administers medications, and assists doctors."

Implement `getSpecialization()` to return `$department`

Implementation Tasks

A. Create the `MedicalPersonnel` abstract class with the required methods.

B. Implement the `Doctor` subclass, ensuring they define `performDuties()` and `getSpecialization()`.

C. Write a main script to:

Create a list of at least one `Doctor`, one `Nurse`, and one `Pharmacist`.

Use a loop to:

Call `displayDetails()` for each object.

Call `performDuties()` for each object.

Call `getSpecialization()` for each object.

QUESTION 3

You are tasked with designing a **Device Management System** for a tech company that manages different types of devices used by employees. The system must use **abstraction** to provide a blueprint for handling various device operations while allowing specific implementations for different device types.

Abstract Class

`Device`

Class Attributes

`deviceId` (String): Unique identifier.

`brand` (String): Device brand.

`model` (String): Device model.

Constructor

Initializes `$deviceId`, `$brand`, and `$model`

Abstract Methods

`calculatePowerConsumption()`: Computes power consumption in kWh.

`calculateMaintenanceCost()`: Computes yearly maintenance cost.

Abstract Methods

`getDetails()`: Returns device details.

Derive Class

`Laptop - Extends Device`

Class Attributes

`processorPower` (double): Power in watts.

`dailyUsageHours` (double): Daily usage in hours.

`maintenanceCostPerYear` (double): Fixed yearly cost.

Constructor

Initializes all fields.

Methods

`calculatePowerConsumption()`: `(processorPower * dailyUsageHours * 365) / 1000`

`calculateMaintenanceCost()`: returns `maintenanceCostPerYear`

Implementation Tasks

A. Create a Laptop instance (`45W`, `5 hrs/day`, `$150/year`).

B. Display Device details.

C. Display Power consumption.

D. Display Maintenance cost.

QUESTION 4

You are tasked with creating a **2D Shape Management System** for a design application that allows users to manage, resize, and render various 2D shapes. Use **abstraction** to define the core operations for all shapes and provide specific implementations for different shape types.

Interface

`shape2D`

Methods:

`draw()`: Outputs shape details.

`resize(double factor)`: Resizes the shape.

`move(double deltaX, double deltaY)`: Updates position.

Class

`Rectangle - Implements 2D interface`

Class Attributes

`$color`(string)

`$positionX`,(double)

`$positionY`(double)

`$width` (double)

`$height`(double)

Constructor:

Initializes all fields.

Methods:

`draw()`: Prints color, position, width, and height.

`resize(factor)`: Multiplies `$width` and `$height` by factor.

`move(deltaX, deltaY)`: Updates position.

Class

`shapeManager`

Class Atrributes

`shapes`: Stores `Shape2D` objects.

Methods

`addShape($shape)`: Adds a shape.

`drawShape()`: Calls `draw()` on shapes.

`resizeShapes($factor)`: Resizes shapes.

`moveShape($deltaX, $deltaY)`: Moves shapes to a different position.

Implementation Tasks

A. Create Shapes:

Rectangle: Red, `(2,3)`, `5,10`.

B. Add to ShapeManager and perform:

- Draw the rectangle.
- Resize rectangle by `2.0x`.
- Move rectangle `(-1.0,1.0)`, then draw again.

QUESTION 5

Design a University Management System in PHP for managing departments, hostels, and students. The system should use interfaces for abstraction

Interface

`shape2D`

Attributes

`deptName` (string): Name of the department.
`deptHead` (string): Head of the department.

Methods:

`printDepartmentDetails()`: Displays department details.

Class

`Student - Implements department`

Class Atrributes

`studentName` (string): Name of the student.

`regdNo` (string): Registration number.

`electiveSubject` (string): Elective subject.

`avgMarks` (float): Average marks.

`hostelName` (string): Name of the hostel.

`hostelLocation` (string): Location of the hostel.

`numberOfRooms` (int): Number of rooms in the hostel.

Methods

`getStudentDetails()`: Inputs and assigns student details, including department and hostel.

`printStudentDetails()`: Displays all student information.

Implements `printDepartmentDetails()`: Prints department details.

`migrateHostel(string $newHostel, string $newLocation, int $newRooms)`: Updates hostel details.

Class

`UniversityManager`

Class Atrributes

`studentRecord` (Student): Holds a single student's information at a time.

Methods

`admitStudent(Student $student)`: Assigns the given student as the current record.

`displayStudentDetails()`: Prints details of the stored student.

`updateHostel(string $newHostel, string $newLocation, int $newRooms)`: Modifies the hostel details of the stored student

Implementation Tasks

A. Define Abstraction for Departments

B. Create a `Student` class that implements `Department`.

C. Create a `UniversityManager` class to handle student-related operations.

D. Admit a new student by providing all necessary details.

E. Migrate a student by updating their hostel details.

F. Display student details using `displayStudentDetails()`