Advanced Project Tasks – Integrated Hospital Care Management System (IHCMS)

MODULE STRUCTURE

```
HospitalSystem
│
├── Core
│   ├── Patient
│   ├── Doctor
│   ├── MedicalRecord
│   ├── Appointment
│
├── Services
│   ├── DiagnosisService
│   ├── BillingService
│   ├── InsuranceService
│
├── Utilities
│   ├── InputHelper
│   ├── CalculationHelper
│
└── Program.cs
```

Task 1: Core OOP Design (Conceptual Understanding)

Objective

Understand and map real-world healthcare entities to OOP concepts.

Tasks for Students

1. Identify and justify the use of:
    - Classes
    - Objects
    - Fields
    - Methods
2. Explain how **encapsulation** is applied to protect patient medical data.
3. Explain why OOP is suitable for healthcare systems compared to procedural programming.

Task 2: Patient Module (Core Layer)

Objective

Implement patient lifecycle management using constructors and encapsulation.

Tasks

1. Create a `Patient` class with:
    - PatientId (read-only)
    - Name, Age
    - Private medical history
2. Implement:
    - Default constructor
    - Parameterized constructor
    - Overloaded constructors
3. Provide methods to:
    - Set medical history
    - Retrieve medical history securely

Task 3: Doctor Module (Static & Readonly Members)

Objective

Understand static and readonly fields in real scenarios.

Tasks

1. Create a `Doctor` class with:
    - Name, Specialization
    - Readonly License Number
    - Static counter for total doctors
2. Use a **static constructor** to initialize static data.
3. Demonstrate that static constructor runs only once.

Task 4: Appointment Scheduling (Methods & Overloading)

Objective

Use method overloading and default arguments.

Tasks

1. Create an `Appointment` class.
2. Overload `ScheduleAppointment()` to:
    - Schedule basic appointment
    - Schedule appointment with date
    - Schedule appointment with date and mode (Online/Offline)
3. Use **default and named arguments** while calling methods.

Task 5: Medical Record Management (Encapsulation & Access Modifiers)

Objective

Apply access modifiers to protect sensitive healthcare data.

Tasks

1. Create a `MedicalRecord` class.
2. Store diagnosis and history as private members.
3. Provide controlled access using public methods.

4. Explain why direct access to medical data is restricted.

Task 6: Diagnosis Service (Advanced Parameter Modifiers)

Objective

Apply `ref`, `out`, `in`, `params`, and local functions.

Tasks

1. Create a `DiagnosisService` class.
2. Implement a method that:
    - Accepts patient age using `in`
    - Updates patient condition using `ref`
    - Returns risk level using `out`
    - Accepts multiple test scores using `params`
3. Use:
    - One local function
    - One static local function
4. Explain why static local functions cannot access outer variables.

Task 7: Billing Service (Object Initializers & Overloading)

Objective

Design flexible billing using object initializers.

Tasks

1. Create a `Bill` or `BillingService` class with:
    - Consultation fee
    - Test charges
    - Room charges
2. Use object initializer syntax to create billing objects.
3. Implement method overloading for different billing scenarios.
4. Calculate total bill amount.

Task 8: Insurance Processing (Type Conversion)

Objective

Handle insurance coverage using type conversion techniques.

Tasks

1. Implement insurance coverage calculation.
2. Use:
    - Implicit casting
    - Explicit casting
    - `Parse`
    - `TryParse`
    - `Convert`
3. Handle invalid user input safely using `TryParse`.

Task 9: Hospital Stay Calculation (Recursion)

Objective

Apply recursion in a healthcare context.

Tasks

1. Implement a recursive method to calculate total hospital stay days.
2. Explain why recursion is suitable for this calculation.
3. Compare recursion with loop-based solution.

Task 10: Utilities Module (Validation & Helper Functions)

Objective

Centralize validation logic using utility classes.

Tasks

1. Create `InputHelper` class.
2. Validate:
    - Patient age
    - Billing amount
3. Throw meaningful error messages for invalid input.

Task 11: System Initialization (Static Constructor & Program Flow)

Objective

Understand system-level initialization.

Tasks

1. Use a static constructor to initialize:
    - Hospital name
    - System configuration
2. Show that static constructor executes before any object creation.
3. Explain the order of execution.

Task 12: Integration & Program Execution

Objective

Integrate all modules into a complete workflow.

Tasks

1. In `Program.cs`, demonstrate:
    - Patient registration
    - Doctor assignment
    - Diagnosis evaluation
    - Billing and insurance processing
2. Ensure proper flow and output readability.
3. Use meaningful console messages.

Task 13: Theory & Reflection

Write short notes on:

1. Advantages of OOP in healthcare systems
2. Role of constructors in system design
3. Difference between static and instance members
4. Why encapsulation is critical in medical applications

Expected Student Deliverables

- Well-structured C# source code
- Proper use of OOP concepts
- Clear comments and explanations
- Clean and readable console output