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 CLASS – DATA SECURITY TASK

Functionality

This class represents a patient and securely stores patient information.

Instructions

- The `PatientId` must be initialized through the constructor and must be **read-only**.
- The `Name` and `Age` must be public properties with getter and setter.
- The `medicalHistory` must be declared as a **private field**.
- A default constructor **must not be created**.

`SetMedicalHistory(string history)`

- This method stores the medical history of the patient.
- The medical history must not be directly accessible from outside the class.

`GetMedicalHistory()`

- This method returns the stored medical history.
- No modification to the medical history should be possible using this method.

TASK 3: DOCTOR CLASS – STATIC BEHAVIOR TASK

Functionality

This class represents a doctor and tracks the total number of doctors in the hospital.

Instructions

- `TotalDoctors` must be declared as a **static variable**.
- `LicenseNumber` must be declared as **readonly**.
- The static constructor must initialize `TotalDoctors`.
- The instance constructor must increment `TotalDoctors` by 1.

Validation

- The license number must not be modified after object creation.

TASK 4: APPOINTMENT CLASS – METHOD OVERLOADING TASK

Functionality

This class schedules appointments between patients and doctors.

Instructions

```
Schedule(Patient p, Doctor d)
```

- This method schedules a basic appointment.
- It must display the patient name and doctor name.

```
Schedule(Patient p, Doctor d, DateTime date, string mode =
   "Offline")
```

- This method schedules an appointment with date and consultation mode.
- The default value of `mode` must be `"Offline"`.
- The date must be displayed in a readable format.

No additional overloaded methods are allowed.

TASK 5: DIAGNOSIS SERVICE – ADVANCED PARAMETER TASK

Functionality

This method evaluates the patient's health condition and risk level.

Instructions

```
Evaluate(in int age, ref string condition, out string riskLevel,
   params int[] testScores)
```

- The `age` parameter must not be modified inside the method.
- All values in `testScores` must be summed.
- A **static local function** named `IsCritical(int sum)` must be created.
- If the total score is greater than 250 **or** age is greater than 60:
    - Set `condition` to `"Serious"`
    - Set `riskLevel` to `"High"`
- Otherwise:
    - Set `riskLevel` to `"Moderate"`
- The `riskLevel` must be assigned in all execution paths.

TASK 6: BILLING CLASS – OBJECT INITIALIZER TASK

Functionality

This class calculates the hospital bill.

Instructions

- The class must contain the following public properties:
    - `ConsultationFee`
    - `TestCharges`
    - `RoomCharges`

```
double Total()
```

- This method must calculate and return the sum of all charges.
- The `Bill` object must be created using **object initializer syntax only**.

- No constructor parameters are allowed.

TASK 7: INSURANCE SERVICE – FINANCIAL LOGIC TASK

Functionality

This method applies insurance coverage to the bill amount.

Instructions

```
ApplyCoverage(double billAmount, int coveragePercent)
```

- Calculate the discount based on coverage percentage.
- Subtract the discount from the bill amount.
- Return the final payable amount.
- The method must not store or modify any class-level data.

TASK 8: RECURSION TASK – HOSPITAL STAY

Functionality

This method calculates the number of hospital stay days using recursion.

Instructions

```
CalculateStayDays(int days)
```

- If `days` is less than or equal to 0, return 0.
- Otherwise, return `1 + CalculateStayDays(days - 1)`.
- No loops (`for`, `while`, `foreach`) are allowed.

TASK 9: INPUT VALIDATION TASK

Functionality

This method validates and converts user input.

Instructions

`ReadAge(string input)`

- Use `int.TryParse()` to convert the input to integer.
- If conversion fails, throw an exception with an appropriate message.
- The method must not return a default or incorrect value.

TASK 10: SYSTEM INITIALIZATION TASK

Functionality

This class initializes the hospital system.

Instructions

- Declare `HospitalName` as a `const string`.
- Create a static constructor.
- The static constructor must display a system boot message.
- The static constructor must execute **before Main()**.

TASK 11: MAIN PROGRAM EXECUTION TASK

Functionality

This method controls the complete hospital workflow.

Instructions

The `Main()` method must:

1. Initialize the hospital system.
2. Read patient and doctor details from the user.
3. Create a `Patient` object.
4. Assign medical history using method call.
5. Create a `Doctor` object.
6. Schedule an appointment.

7. Perform diagnosis evaluation.
8. Generate hospital bill.
9. Apply insurance coverage.
10. Display the final payable amount.

If **any validation fails**, display:

- Invalid Input
