**1:** Create a Library Management System with the following namespace structure:

- `LibrarySystem.Books` - Book, Magazine, Journal classes
- `LibrarySystem.Members` - Member, Librarian classes
- `LibrarySystem.Transactions` - BorrowTransaction, ReturnTransaction classes

**Requirements:**

- Create at least 2 classes in each namespace
- Use `using` statements to simplify code
- Create a main program(Console App) that instantiates objects from all namespaces

**Deliverable:** Class library project + consuming console app.

### 2: Create two different libraries that have classes with the same name and resolve conflicts using namespace aliases.

**Scenario:**

- `CompanyA.Reporting.Report` class
- `CompanyB.Analytics.Report` class
- Both have different implementations of `Generate()` method

**Requirements:**

- Use namespace aliases to differentiate between the two Report classes
- Demonstrate using both classes in the same file

**3:** Create a `MathUtilities` class library with basic mathematical functions.

**Requirements:**

- Methods: `IsEven()`, `IsPrime()`, `Factorial()`
- Create a test console application that uses the library and call these methods

**Deliverable:** Class library project + consuming console app.

**4:** Create a solution with multiple projects that depend on each other.

**Structure:**

- `DataAccess.dll` - Database connection classes (simulated)
- `BusinessLogic.dll` - Referencescha DataAccess, contains business rules
- `UI.ConsoleApp` - References BusinessLogic, user interface

**Requirements:**

- Demonstrate project-to-project references
- Show how changes in DataAccess affect BusinessLogic
- Use dependency flow: UI Console → BusinessLogic → DataAccess

**Deliverable:** Multi-project solution demonstrating proper layered architecture and project dependencies.

**5:** Create a `TemperatureConverter` library with internal validation.

**Requirements:**

- Public class: `TemperatureConverter` with methods `CelsiusToFahrenheit()`, `FahrenheitToCelsius()`
- Internal class: `TemperatureValidator` that checks if temperature is within valid range (-273.15°C to 5500°C)
- Public methods should use the internal validator
- Create a client app that cannot access `TemperatureValidator` directly

**Deliverable:** Library + client app demonstrating that internal classes are inaccessible outside the assembly.

**6:** Create a simple calculator that handles exceptions and logs errors.

**Requirements:**

- Implement basic operations: Add, Subtract, Multiply, Divide
- Handle: `DivideByZeroException`, `FormatException`, `OverflowException`
- Log all exceptions into console
- Use try-catch-finally blocks appropriately
- Display user-friendly error messages

**Deliverable:** Console calculator app to insert numbers for operation, then show different exception scenarios.

**7: Task:** Refactor poorly named code to follow C# naming conventions.

**Provided Bad Code Example:**

```
public class emp {
        public string n; public int a;
        public double s;
        public void calc()
        {
                double x = s * 0.1;
                double y = s + x;
        }
}
```

**Requirements:**

- Refactor all names following C# conventions
- Meaningful descriptive names

**Deliverable:** Refactored code + comparison document explaining each change.