**Banking System**

**Problem Statement:** Student Management System

**Objective**

As part of a training assignment, you are tasked with developing a console-based student management system in C# that utilizes collections (`Dictionary` and `List`) to store and manage student records. The system should allow adding student information, retrieving student details by ID, and displaying all students. The assignment focuses on using collections effectively to organize data, handle lookups, and implement basic error handling for invalid operations.

**Requirements**

You are required to create a C# console application that implements a student management system with the following components and functionality:

**1. Student Class:**

- Create a `Student` class to represent a student record.

- Properties:

- `Id` (int): A unique identifier for the student (read-only after initialization).

- `Name` (string): The student's name (read-only after initialization).

- `Grade` (double): The student's average grade (0.0 to 100.0, read-only after initialization).

- Ensure properties are encapsulated (e.g., use private setters or equivalent to prevent external modification after object creation).

**2. StudentManager Class:**

- Create a `StudentManager` class to manage student records.

- Collections:

- Use a `Dictionary<int, Student>` to store student records, where the key is the student’s ID and the value is a `Student` object.

- Optionally, use a `List<Student>` internally (or rely on the `Dictionary`’s `Values` collection) for iterating over students.

- Methods:

- `AddStudent(int id, string name, double grade)`: Adds a new student to the collection.

- `GetStudentById(int id)`: Retrieves a student by their ID.

- `DisplayAllStudents()`: Displays all students in the system, including their ID, name, and grade.

- Validation:

- Ensure student IDs are unique; throw an `ArgumentException` if an ID already exists.

- Validate that the student’s name is not null or empty; throw an `ArgumentException` if invalid.

- Ensure the grade is between 0.0 and 100.0; throw an `ArgumentException` if out of range.

- Throw a `KeyNotFoundException` if a requested student ID does not exist in the `GetStudentById` method.

**3. Main Program:**

- In the `Main` method, create a `StudentManager` instance and demonstrate the following test cases:

- Add at least three students with valid IDs, names, and grades.

- Attempt to add a student with a duplicate ID (should throw an exception).

- Attempt to add a student with an invalid grade (e.g., -10 or 150).

- Retrieve and display a student by ID (both valid and invalid cases).

- Display all students in the system.

- Use try-catch blocks to handle exceptions and display user-friendly error messages.

**Constraints**

- Use a `Dictionary<int, Student>` as the primary collection for storing student records to enable fast lookups by ID.

- Ensure the `Student` class properties are immutable after initialization (e.g., use private setters or readonly fields).

- Validate inputs:

- Student IDs must be positive integers.

- Names must not be null, empty, or consist only of whitespace.

- Grades must be in the range [0.0, 100.0].

- Handle exceptions gracefully to prevent the application from crashing.

- Keep the code modular and simple, suitable for beginner to intermediate learners.

- Do not use external libraries or databases; store data in memory using collections.

Expected Deliverables

1. Source Code:

- A C# console application with a `Student` class, a `StudentManager` class, and a `Program` class containing the `Main` method.

- The code should be well-organized, with clear comments explaining the purpose of each class, method, and collection usage.

2. Console Output:

- Display a list of all students with their ID, name, and grade.

- Show the result of retrieving a student by ID (both successful and failed attempts).

- Display appropriate error messages for invalid operations (e.g., duplicate ID, invalid grade, or missing student).

- Example output:

Student List:

ID: 1, Name: Alice, Grade: 85.5

ID: 2, Name: Bob, Grade: 90.0

ID: 3, Name: Charlie, Grade: 78.5

Found: Bob, Grade: 90.0

Error: Student with ID 4 not found.

Error: Student with ID 1 already exists.

Error: Grade must be between 0 and 100.

**Example** **Workflow**

1. Create a `StudentManager` and add students with IDs 1, 2, and 3 (e.g., Alice with grade 85.5, Bob with grade 90.0, Charlie with grade 78.5).

2. Attempt to add a student with ID 1 again (should throw `ArgumentException`).

3. Attempt to add a student with a grade of -10 (should throw `ArgumentException`).

4. Retrieve the student with ID 2 and display their details.

5. Attempt to retrieve a student with ID 4 (should throw `KeyNotFoundException`).

6. Display all students in the system.