WRITE UP
Create an OOP Based System for Storing School Data Using Design Patterns
Description
The application will then fill the objects with dummy data to test its correctness.

STEPS:

Step1:Create Visual Studio Project

Step 2: Singleton Pattern - SchoolDataStorage

```
csharp
public class SchoolDataStorage
{
    private static SchoolDataStorage instance;
    public List<Student> Students { get; set; }
    public List<Teacher> Teachers { get; set; }
    public List<Subject> Subjects { get; set; }

    private SchoolDataStorage()
    {
        Students = new List<Student>();
        Teachers = new List<Teacher>();
        Subjects = new List<Subject>();
    }

    public static SchoolDataStorage Instance
    {
        get
        {
            if (instance == null)
            {
                instance = new SchoolDataStorage();
            }
            return instance;
        }
    }
}
```

Explanation:*
- SchoolDataStorage is designed as a singleton to ensure only one instance exists.
- It has lists to store Student, Teacher, and Subject objects.
- The constructor is private to prevent direct instantiation.
- The Instance property provides a global point of access to the single instance, creating it if it doesn't exist.

Step 3: Entity Classes - Student, Teacher, Subject

```
csharp
public class Subject
```

```
{
    public string Name { get; set; }
    public string SubjectCode { get; set; }
    public Teacher Teacher { get; set; }
}

public class Teacher
{
    public string Name { get; set; }
    public string ClassAndSection { get; set; }
}

public class Student
{
    public string Name { get; set; }
    public string ClassAndSection { get; set; }
}
```

Explanation:*
  - Subject, Teacher, and Student classes represent entities with specific properties.

Step 4: Repository Pattern - SchoolRepository

```
csharp
public class SchoolRepository
{
    private SchoolDataStorage dataStorage;

    public SchoolRepository()
    {
        dataStorage = SchoolDataStorage.Instance;
    }

    // Methods for adding data
    public void AddStudent(Student student) { /* ... */ }
    public void AddTeacher(Teacher teacher) { /* ... */ }
    public void AddSubject(Subject subject) { /* ... */ }

    // Methods for retrieving data
    public List<Student> GetStudentsInClass(string classAndSection) { /* ... */ }
    public List<Subject> GetSubjectsTaughtByTeacher(string teacherName) { /* ... */ }

    // Additional methods for updating and removing data
    public void UpdateStudent(Student existingStudent, Student updatedStudent) { /* ... */ }
    public void RemoveSubject(Subject subjectToRemove) { /* ... */ }
}
```

- ***Explanation:***
  - SchoolRepository acts as a mediator between the program and data storage

(SchoolDataStorage).
- It provides methods to add, retrieve, update, and remove data.
- It uses the singleton instance of SchoolDataStorage.

Step 5: Main Program - Program , Populate the dummy data

```
csharp
class Program
{
    static void Main()
    {
        SchoolRepository repository = new SchoolRepository();

        // Adding dummy data
        // ...

        // Displaying lists
        // ...
    }
}
```

- ***Explanation:***
  - In the Main method, a SchoolRepository instance is created.
  - Dummy data (students, teachers, subjects) is added using repository methods.
  - Lists of students and subjects are displayed using repository methods.

Key Points:

1. *Singleton Pattern:*
   - Ensures a single instance of SchoolDataStorage, facilitating centralized data storage.

2. *Entity Classes:*
   - Subject, Teacher, and Student represent the main entities in the system.

3. *Repository Pattern:*
   - SchoolRepository provides an interface to interact with data storage, encapsulating data manipulation logic.

4. *Main Program:*
   - Demonstrates how to use the repository to add data and retrieve lists of students and subjects.

5. *Flexibility:*
   - The code can be extended to handle more complex scenarios with additional repository methods.

Step 6: Crete Git Repository and documentations .