# Project: Project Title Here


Session: 2021 – 2024

**Submitted by:**

Your Name Your Registration No

**Supervised by:**

Your OOP Teacher Name

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

## Important Instructions

**Here you can find the major parts of your Game documentation**

- Table of Contents
- Short Description of your Project
- Project Features List
- Wireframes
- CRC Diagram
- Complete Code

**Formatting Instructions**

1. Heading Size is 16
2. Sub heading size is 14
3. Further heading size is 13
4. Make your heading font bold
5. Text Font size is 12
6. Use Times New Roman Font Style
7. Text paragraphs should be justified. (Justify is feature of MS World)
8. Code Size should be 10 and 1.0 line Spacing to make it compact.
9. Follow proper coding Styles to make the classes and driver program

## Project Structure Guidelines

You must create three projects: a DLL Library project, a Windows Forms project, and a Console Application project (Total 3 projects). Both the Windows Forms and Console Application projects will utilize this library and access the data layer to perform operations on the database/file handling

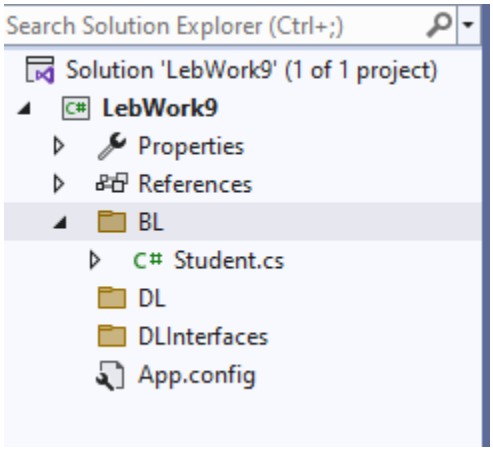
**Here is the Library Project Structure.**

In your DLL project, create a folder named 'DLInterfaces' and place all your interfaces in that folder. You must create a separate interface for each entity. For example, if you are developing a university management system and you want to manage teachers, students, and exams in your project, then you will create separate interfaces for teacher, student, and exam.

Next, create another folder named 'DL' where you will implement your interfaces for database and file handling code. For every single interface, you will have two concrete classes: one with a database implementation and the other with a file handling implementation.

Let's do everything for Student class.

Example Structure for Student BL Class.

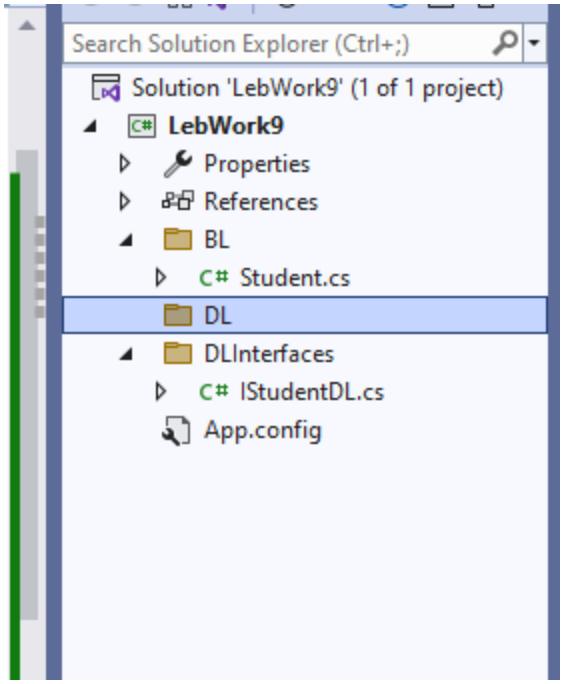



This is a student BL class. It contains some data members, constructor and getter setters.

```
namespace LebWork9.BL
{
    9 references
    public class Student
    {
        private string Roll;
        private string Name;
        private string Ecat;
        private string Fsc;
        private string Matric;
        private string Contact;

        0 references
        public Student(string roll, string name, string ecat,
            string fsc, string matric, string contact)
        {
            Roll=roll;
            Name=name;
            Ecat=ecat;
            Fsc=fsc;
            Matric=matric;
            Contact=contact;
        }
        //getter setter here
        0 references
        public string GetRoll()
        {
            return Roll;
        }
```

This interface contain some common functions, you can add and remove as per your project/class requirements.




```
namespace LebWork9.DL
{
    0 references
    internal interface IStudentDL
    {
        0 references
        Student FindStudentById(int id);
        0 references
        Student Save(Student student);
        0 references
        Student Update(Student student);
        0 references
        Student Delete(int id);
        0 references
        List<Student> GetAll();
        0 references
        List<Student> GetAllByName(string name); //it will get students by first name

    }
}
```

Add 2 sub folders in DL names DB(Database) and FH (File Handling)

In the 'DB' folder, we will create the 'StudentDB' class, which will inherit from the 'IStudentDL' interface. Now, you have to implement all the abstract functions inside this 'StudentDB' class,

which will contain the code for database operations.

```
using LebWork9.BL;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LebWork9.DL.DB
{

    public class StudentDB : IStudentDL
    {
        1 reference
        public Student Delete(int id)
        {
            //write your logic to delete data from database on basis of ID
            return null;
        }

        1 reference
        public Student FindStudentById(int id)
        {
            //Write your code to find a student from database
            return null;
        }

        1 reference
        public List<Student> GetAll()
        {
```

Now, you have to create another class named 'StudentFH'. It will also inherit the same interface as 'IStudentDL', and in this class, you have to implement all the functions with file handling code. **Note:** For file handling, you can create a static list in the StudentFH class, but you can still

achieve your whole functionality without creating a static global list.

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LebWork9.DL.FH
{
    0 references
    public class StudentFH : IStudentDL
    {
        1 reference
        public Student Delete(int id)
        {
            //write yoru code to Delete a data from the File
            return null;
        }

        1 reference
        public Student FindStudentById(int id)
        {
            //write a code to find a student by id from file
            return null;
        }

        1 reference
        public List<Student> GetAll()
        {
            //Write a code to load all students from file and return
            return null;
        }
```

The next step is to add this library into your Windows form project and finalize the functionality of your project.

## Console Project

Let's see how we can call this code in our driver application. The first step is to create a UI folder in your console project, and then create UI classes for your entities such as student, teacher, and exam, following our example. Implement your UI functions to take data from the user, display data, and perform other necessary operations.

**Read all comments carefully.**

```
0 references
public static void Main(string[] args)
{
    //Note: You will create UI layer in your console application to manage your all input/outputs

    // Here in the console application, you will create all of your objects of Data Layers.
    IStudentDL student = new StudentDB(); // You are creating an object to store student data in the database.
    // ITeacher teacher = new TeacherFH(); // For example, you want to store teacher data in a file.
    // Driver code
    int option = 0;
    while (true)
    {
        // int option = UI.Menu(); // You will create UI.Menu in your console project, not in the library.
        if (option == 1)
        {
            Student one = new Student("1", "ali", "333", "333", "333", "333");
            // Hardcoding student values; you can take student data from the UI implemented in your console project.
            student.Save(one);
        }
        else if (option == 2)
        {
            List<Student> list = student.GetAll();
            // UI.DisplayAll(list); // You will pass the list to your UI layer to display all students.
        }
        else if (option == 3)
        {
            // string roll = UI.GetStudentRoll();
            Student response = student.FindStudentById(1); // Here you will pass the roll number.
            // UI.DisplayStudent(response);
        }
```

If you closely observe this code

```
// Here in the console application, you will create all of your objects of Data Layers.
IStudentDL student = new StudentDB(); // You are creating an object to store student data in the database.
// ITeacher teacher = new TeacherFH(); // For example, you want to store teacher data in a file.
```

Here, we are using an interface to hold the reference of the object so that we can change this object created from the database concrete class to the file handling concrete class anytime, and it will not affect the code below.

If you see this code, we have changed the StudentDB to StudentFH and now it will use files to store student data.

```
// Here in the console application, you will create all of your objects of Data Layers.
IStudentDL student = new StudentFH(); // You are creating an object to store student data in the database.
// ITeacher teacher = new TeacherFH(); // For example, you want to store teacher data in a file.
```

## Window Form

Create a UI folder within your Windows Forms project, and within this folder, create all the necessary forms (Window Forms). Additionally, create an additional class named ObjectHandler in this project. This class will contain all the static reference variables to store the objects of the database/file handling data layer. Next, you'll need to write a function to return these objects, as depicted in the attached picture. With these objects readily available, you can now call the data layer functions within your Windows Forms to execute the respective functions.

```
namespace LebWork9
{
    0 references
    public class ObjectHandler
    {
        private static IStudentDL studentDL = new StudentDB();
      //private static ITeacherDL teacherDl = new TeacherFH();


        0 references
        public static IStudentDL GetStudentDL()
        {
            return studentDL;
        }

        // public static ITeacherDL GetTeacherDL()
        // {
        //     return teacherDl;
        //}

    }
}
```

This is the **StudentUI** window form code to save the student data on button click. Data is hard code so you can take it from Text Field in your form.

```
namespace LebWork9.UI
{
    2 references
    public partial class StudentUI : Form
    {
        0 references
        public StudentUI()
        {
            InitializeComponent();
        }

        1 reference
        private void saveButton_Click(object sender, EventArgs e)
        {
            Student one = new Student("1", "ali", "333", "333", "333", "333");
            ObjectHandler.GetStudentDL().Save(one);
        }
    }
}
```

If you examine the code, you'll notice that we're utilizing the Object Handler to obtain the Student Data Layer object, and subsequently, we invoke the respective functions to fulfill the functionality required. If GetStudentDL() returns the file handling class object, it will save the data in a file. Conversely, if it returns the database class object, it will save the data in the database. Therefore, we have the flexibility to switch the configuration from database to file and vice versa at any time within our Object Handler class.

**Best of Luck**

**Evaluator Reg. No. :** **Evaluator Name:**

| | A-Extensive Evidence | B-Convincing Evidence | C-Limited Evidence | D-No Evidence |
|---|---|---|---|---|
| **Documentation** | | | | |
| **Class Diagram**<br><br>**Grade:** | The project contains all the required classes, and the document includes a well-designed CRC diagram of the project. | Met 80% of the criteria given in extensive evidence | Met 50% of the criteria given in extensive evidence | Met less than 50% of th given in extensive evid |
| **Example Usage Document**<br><br>**Grade:** | Example Usage documentation was readable and easy to understand | Example Usage documentation was readable but took some time to read | Documentation requires a lot of improvement and takes a lot of effort to read. | The documentation is n understandable. |
| **Framework** | | | | |
| **Classes** | The project contains all the required classes, and the document contains a well-designed CRC diagram of the project. | Met 80% of the criteria given in extensive evidence | Met 50% of the criteria given in extensive evidence | Met less than 50% of th given in extensive evid |
| **Encapsulation** | Every class must include private data members, getters, and setters. Additionally, in the library, the classes must be public. | Met 80% of the criteria given in extensive evidence | Met 50% of the criteria given in extensive evidence | Met less than 50% of th given in extensive evid |
| **Has and Is a relationship** | The project incorporates both "Is A" (inheritance) and "Has A" (association) relationships. | Met 80% of the criteria given in extensive evidence | Met 50% of the criteria given in extensive evidence | Met less than 50% of th given in extensive evid |
| **Polymorphism** | The project should include interfaces for the DL layer with all the necessary abstract functions, and there must be fully operational concrete classes for both database and file handling. | Met 80% of the criteria given in extensive evidence | Met 50% of the criteria given in extensive evidence | Met less than 50% of th given in extensive evid |