**ASSIGNMENT 2 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 20: Advanced Programming | | |
| **Submission date** | 10/05/2021 | **Date Received 1st submission** |  |
| **Re-submission Date** | 13/05/2021 | **Date Received 2nd submission** |  |
| **Student Name** | PHAN MINH TRI | **Student ID** | GCC18015 |
| **Class** | GCC07F1 | **Assessor name** | TRUNG-VIET’S NGUYEN |
| **Student declaration**  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
|  |  | **Student’s signature** |  |

**Grading grid**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P3 | P4 | M3 | M4 | D3 | D4 |
|  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **❒ Summative Feedback: ❒ Resubmission Feedback:** | | |
| **Grade:** | **Assessor Signature:** | **Date:** |
| **Lecturer Signature:** | | |

|  |  |
| --- | --- |
| **Assignment title** | **Assignment 2: Implement and test an Object Oriented programming solution.** |
| In this assignment, you will have opportunities to provide evidence against the following criteria.  Indicate the page numbers where the evidence can be found. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Assessment criteria** | **Expected evidence** | **Task no.** | **Assessor’s Feedback** |
| **LO3: Implement code applying design patterns** | | | |
| 3.1: Build an application derived from UML class diagrams.. | Code the application based on UML diagrams  Coded UML Class diagram and explanation about Relationships among classes .  How did you implement main functionalities (Add, update, delete) with main flow and code snippet | 3 |  |
| **LO4: Investigate scenarios with respect to design patterns.** | | | |
| 4.1  Discuss a range of design patterns with relevant examples of creational, structural and behavioural pattern types | Describe the use of design patters with relevant examples of Singleton, Builder, Adapter, Iterator, Observer of Design Pattern | 4 |  |

|  |  |  |
| --- | --- | --- |
| **Assessment criteria** | **Expected Evidence** | **Feedback**  **(note on Merit/Distinction if applicable)** |
| Merit descriptor No. (M3) |  |  |
| Merit descriptor No. (M4) |  |  |
| Distinction descriptor No. (D3) |  |  |
| Distinction descriptor No. (D4) |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Summative feedback: | | | |
| **Assessor’s Signature:** |  | **Date:** |  |

**<ATTACHED EVIDENCE**

**ASSIGNMENT 2 BRIEF**

|  |  |
| --- | --- |
| **Unit Number and Title** | **20: Advance Programming** |
| Academic Year | 2018 |
| Unit Tutor | Hoàng Đức Quang |
| **Assignment Title** | **Assignment 2** |
| **Issue Date** |  |
| Submission Date |  |
| IV Name & Date |  |

|  |  |  |
| --- | --- | --- |
| **Pass** | **Merit** | **Distinction** |
| **LO3** Implement code applying design patterns | | **D3** Evaluate the use of design patterns for the given purpose specified in M3. |
| **P3** Build an application derived from UML class diagrams. | **M3** Develop code that imple ments a design pattern for a given purpose. |
| **LO4** Investigate scenarios with respect to design patterns | | **D4** Critically evaluate a range of design patterns against the range of given scenarios with justification of your choices. |
| **P4** Discuss a range of design patterns with relevant examples of creational, structural  and behavioural pattern types. | **M4** Reconcile the most appropriate design pattern from a range with a series of given scenarios. |

|  |  |
| --- | --- |
| **Aim of the assignment** | This assignment satisfies the following learning outcomes and assessment criteria:  **LO3: Implement code applying design patterns**  P3. Build an application derived from UML class diagrams.  **LO4: Investigate scenarios with respect to design patterns**  P4. Discuss a range of design patterns with relevant examples of creational, structural and behavioural pattern types. |
| **Specific requirements**  *(see Appendix for assessment criteria and grade descriptors)* | **Scenario:**  Please see the scenario from Assignment Brief 1.  **Task 1**  In this task you will need to:   * Code the application based on UML diagrams   **Task 2**  Please prepare a presentation with the following points   * Coded UML Class diagram and explanation about Relationships among classes . * How did you implement main functionalities (Add, update, delete) with main flow and code snippet   **Task 3**  **Discuss a range of design patterns**   * Describe the use of design patters with relevant examples of Singleton, Builder, Adapter, Iterator, Observer of Design Pattern |
| **Student guidelines** | For the assignment assessments, you are required to:   * Produce a presentation to explain the code’s structure, IDE’s features such as code generation, debugging and show test cases and test result evaluations. * Write the program to fulfill the requirements |
| **Submission requirements** | Students are expected to submit hard copy of assignment |

**Appendix A- Grade Descriptor**

|  |  |  |
| --- | --- | --- |
| In addition to the above PASS criteria, this assignment gives you the opportunity to submit evidence in order to achieve the following MERIT and DISTINCTION grades | | |
| Grade Descriptor | Indicative characteristic/s | Contextualization |
| M3 | Develop code that imple ments a design pattern for a given purpose. |  |
| M4 | Reconcile the most appropriate design pattern from a range with a series of given scenarios. |  |
| D3 | Evaluate the use of design patterns for the given purpose specified in M3. |  |
| D4 | Critically evaluate a range of design patterns against the range of given scenarios with justification of your choices. |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| This brief has been verified as being fit for purpose | | | | | |
| **Internal Verifier 1** |  | Signature |  | Date |  |
| **Internal Verifier 2** |  | Signature |  | Date |  |

Table of Contents

[**ASSIGNMENT 2 BRIEF** 7](#_Toc71839821)

[**Appendix A- Grade Descriptor** 9](#_Toc71839822)

[**I.** **Build an application derived from UML class diagrams.** 12](#_Toc71839823)

[**1.** **Main Menu** 13](#_Toc71839824)

[**a)** **Figure** 13](#_Toc71839825)

[**b)** **Description** 13](#_Toc71839826)

[**c)** **Code** 13](#_Toc71839827)

[**2.** **Management Students** 14](#_Toc71839828)

[**a)** **Figure** 14](#_Toc71839829)

[**b)** **Description** 14](#_Toc71839830)

[**c)** **Code** 14](#_Toc71839831)

[**3.** **Management Lecturers** 15](#_Toc71839832)

[**a)** **Figure** 15](#_Toc71839833)

[**b)** **Description** 15](#_Toc71839834)

[**c)** **Code** 15](#_Toc71839835)

[**4.** **Add** 16](#_Toc71839836)

[**a)** **Add Student** 16](#_Toc71839837)

[**b)** **Add lecturer** 17](#_Toc71839838)

[**5.** **View** 18](#_Toc71839839)

[**a)** **View all student** 18](#_Toc71839840)

[**b)** **View all lecturer** 19](#_Toc71839841)

[**6.** **Search** 19](#_Toc71839842)

[**a)** **Student** 19](#_Toc71839843)

[**b)** **Lecturer** 21](#_Toc71839844)

[**7.** **Delete** 22](#_Toc71839845)

[**a)** **Student** 22](#_Toc71839846)

[**b)** **Lecturer** 23](#_Toc71839847)

[**8.** **Update** 25](#_Toc71839848)

[**a)** **Student** 25](#_Toc71839849)

[**b)** **Lecturer** 27](#_Toc71839850)

[**9.** **Test case** 29](#_Toc71839851)

[**II.** **Discuss a range of design patterns with relevant examples of creational, structural and behavioural pattern types.** 37](#_Toc71839852)

[**1.** **Singleton** 37](#_Toc71839853)

[**a)** **What is Singleton Design Pattern?** 37](#_Toc71839854)

[**b)** **Real-Life Example** 37](#_Toc71839855)

[**c)** **What are the Advantages of using the Singleton Pattern in C#?** 37](#_Toc71839856)

[**d)** **Code example** 37](#_Toc71839857)

[**e)** **Singleton UML diagram example** 39](#_Toc71839858)

[**2.** **Builder** 39](#_Toc71839859)

[**a)** **What is Builder Pattern** 39](#_Toc71839860)

[**b)** **Real-Life Example:** 39](#_Toc71839861)

[**c)** **What are the Advantages of using the Builder Pattern in C#?** 39](#_Toc71839862)

[**d)** **Code example** 40](#_Toc71839863)

[**e)** **Builder UML diagram example** 43](#_Toc71839864)

[**3.** **Adapter** 44](#_Toc71839865)

[**a)** **What is Adapter Design Pattern?** 44](#_Toc71839866)

[**b)** **Example** 44](#_Toc71839867)

[**c)** **What are the Advantages of using the Adapter Pattern in C#?** 44](#_Toc71839868)

[**d)** **Code example** 44](#_Toc71839869)

[**e)** **Adapter UML diagram example** 46](#_Toc71839870)

[**4.** **Iterator** 47](#_Toc71839871)

[**a)** **What is Iterator Design Pattern?** 47](#_Toc71839872)

[**b)** **Real-Life example:** 47](#_Toc71839873)

[**c)** **What are the Advantages of using the Iterator Pattern in C#?** 47](#_Toc71839874)

[**d)** **Code example** 47](#_Toc71839875)

[**e)** **Iterator UML diagram example:** 49](#_Toc71839876)

[**f)** **Advantages Iterator Design Pattern** 50](#_Toc71839877)

[**5.** **Observer** 50](#_Toc71839878)

[**a)** **What is Observer** 50](#_Toc71839879)

[**b)** **Real-Life example:** 50](#_Toc71839880)

[**c)** **What are the Advantages of using the Observer Pattern in C#?** 50](#_Toc71839881)

[**d)** **Code example** 51](#_Toc71839882)

[**e)** **Advantages Observer Design Pattern** 54](#_Toc71839883)

[**f)** **Observer UML diagram example** 54](#_Toc71839884)

[References 55](#_Toc71839885)

Table of Figure

[*Figure 1: Main Menu* 13](#_Toc71839887)

[*Figure 2: Management Student* 14](#_Toc71839888)

[*Figure 3: Management Lectures* 15](#_Toc71839889)

[*Figure 4: Add student* 16](#_Toc71839890)

[*Figure 5: Add lecturer* 17](#_Toc71839891)

[*Figure 6: View all student* 18](#_Toc71839892)

[*Figure 7: View all lecturer* 19](#_Toc71839893)

[*Figure 8: Search student* 20](#_Toc71839894)

[*Figure 9: Search lecturer* 21](#_Toc71839895)

[*Figure 10: Delete student 1.1* 22](#_Toc71839896)

[*Figure 11: Delete student 1.2* 22](#_Toc71839897)

[Figure 12: Delete lecturer 1.1 23](#_Toc71839898)

[Figure 13: Delete lecturer 1.2 24](#_Toc71839899)

[*Figure 14: Update student* 25](#_Toc71839900)

[*Figure 15: Update lecturer 1.1* 27](#_Toc71839901)

[*Figure 16: Update lecturer 1.2* 27](#_Toc71839902)

[*Figure 17: Singleton output* 38](#_Toc71839903)

[*Figure 18: Singleton UML diagram example* 39](#_Toc71839904)

[*Figure 19: Output code* 43](#_Toc71839905)

[*Figure 20: UML Diagram of Builder Design Pattern* 43](#_Toc71839906)

[*Figure 21: Output code example* 46](#_Toc71839907)

[*Figure 22: Adapter UML diagram example* 46](#_Toc71839908)

[*Figure 23: Result - Iterator Design Pattern* 49](#_Toc71839909)

[*Figure 24: Class diagram - Iterator Design Pattern* 49](#_Toc71839910)

[*Figure 25: Output Observer code Example.* 53](#_Toc71839911)

[*Figure 26: Observer UML diagram example* 54](#_Toc71839912)

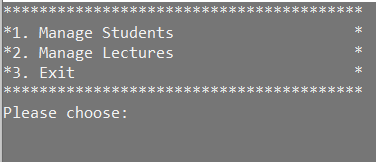
Table of Table

[*Table 1: Test case* 36](#_Toc71839913)

# **Build an application derived from UML class diagrams.**

# **Main Menu**

# **Figure**



*Figure 1: Main Menu*

# **Description**

* When the user selects number 1, the program will display the student’s management item.
* When the user chooses number 2, the program will display the lecturer’s management item.
* When the user selects 3, the program will exit.

# **Code**

ChangeColor();

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\*1. Manage Students \*");

Console.WriteLine("\*2. Manage Lectures \*");

Console.WriteLine("\*3. Exit \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

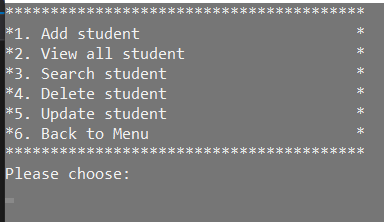
Console.WriteLine("Please choose:");

string strResult = Console.ReadLine();

return strResult;

# **Management Students**

# **Figure**



*Figure 2: Management Student*

# **Description**

* When the user selects option 1, the software allows the user to add student details to the list.
* When the user selects option 2, the program displays all of the students on the list.
* When the user chooses 3, the program will allow the user to search information student on the list.
* When the user selects 4, the software would encourage the user to edit the student details on the list.
* When the user selects 5, the software would encourage the user to erase details from the list of students.
* When the user chooses 6, back to main menu.

# **Code**

private static string ManageStudents()

{

ChangeColor();

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\*1. Add student \*");

Console.WriteLine("\*2. View all student \*");

Console.WriteLine("\*3. Search student \*");

Console.WriteLine("\*4. Delete student \*");

Console.WriteLine("\*5. Update student \*");

Console.WriteLine("\*6. Back to Menu \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("Please choose:");

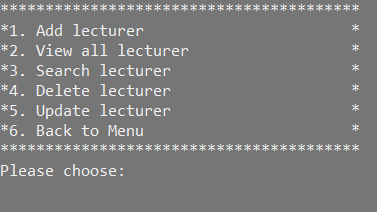
string strResult = Console.ReadLine();

return strResult;

}

# **Management Lecturers**

# **Figure**



*Figure 3: Management Lectures*

# **Description**

* When the user selects option 1, the software allows the user to add lecture details to the list.
* When the user selects option 2, the program displays all of the lectures on the list.
* When the user chooses 3, the program will allow the user to search information lectures on the list.
* When the user selects 4, the software would encourage the user to edit the details of the lecture on the list.
* When the user selects 5, the software would encourage the user to erase details from the list of lectures.
* When the user chooses 6, back to main menu.

# **Code**

private static string ManageLecturers()

{

ChangeColor();

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\*1. Add lecturer \*");

Console.WriteLine("\*2. View all lecturer \*");

Console.WriteLine("\*3. Search lecturer \*");

Console.WriteLine("\*4. Delete lecturer \*");

Console.WriteLine("\*5. Update lecturer \*");

Console.WriteLine("\*6. Back to Menu \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("Please choose:");

string strResult = Console.ReadLine();

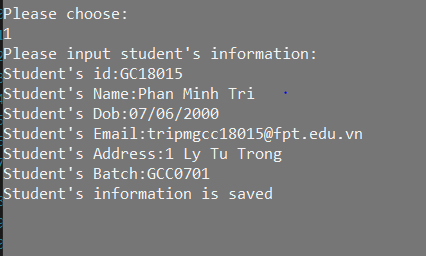
return strResult;

}

# **Add**

# **Add Student**

* Figure



*Figure 4: Add student*

* Description
* When the user selects 1, the software will prompt the user to enter student details (specified previously). The script will then check the input data and if anything is right, will add a new student to the existing student list. The app should notify the user of any related messages.
* Code

class Student : Person

{

public string IdStudent { get; set; }

public string stdBatch { get; set; }

public static void ShowInputForm(List<Student> lstStudent)

{

Student objstd = new Student();

Console.WriteLine("Please input student's information:");

//id

Console.Write("Student's id:");

do

{

objstd.IdStudent = Console.ReadLine();

} while (vailidate(objstd.IdStudent, lstStudent));

//name

Console.Write("Student's Name:");

objstd.Name = Console.ReadLine();

//Dob

Console.Write("Student's Dob:");

objstd.Dob = Convert.ToDateTime(Console.ReadLine());

//email

Console.Write("Student's Email:");

objstd.Email = Console.ReadLine();

//address

Console.Write("Student's Address:");

objstd.Address = Console.ReadLine();

//Batch

Console.Write("Student's Batch:");

objstd.stdBatch = Console.ReadLine();

lstStudent.Add(objstd);

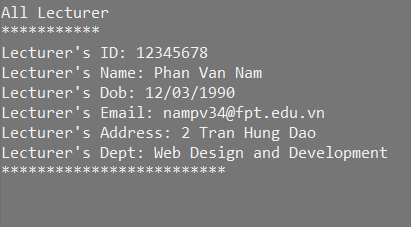
Console.WriteLine("Student's information is saved");

Console.ReadKey();

}

# **Add lecturer**

* Figure



*Figure 5: Add lecturer*

* Description
* When the user selects 1, the software will ask the user to enter information about the lecturer (specified previously). The software will then validate the input data, and if it is all right, it will add a new lecturer to the current lecturer list. The user should be informed of any relevant messages by the software.
* Code

class Lecturer : Person

{

public string IdLecturer { get; set; }

public string lecDept { get; set; }

public static void ShowInputForm(List<Lecturer> lstLecturer)

{

Lecturer objlec = new Lecturer();

Console.WriteLine("Please input lecturer's information:");

//id

Console.Write("Lecturer's id:");

do

{

objlec.IdLecturer = Console.ReadLine();

} while (vailidate(objlec.IdLecturer, lstLecturer));

//name

Console.Write("Lecturer's Name:");

objlec.Name = Console.ReadLine();

//Dob

Console.Write("Lecturer's Dob:");

objlec.Dob = Convert.ToDateTime(Console.ReadLine());

//email

Console.Write("Lecturer's Email:");

objlec.Email = Console.ReadLine();

//address

Console.Write("Lecturer's Address:");

objlec.Address = Console.ReadLine();

//Dept

Console.Write("Lecturer's lecDept:");

objlec.lecDept = Console.ReadLine();

lstLecturer.Add(objlec);

Console.WriteLine("Lecturer's information is saved");

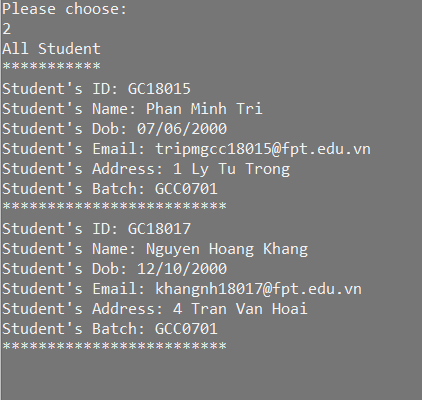
Console.ReadKey();

}

# **View**

# **View all student**

* Figure

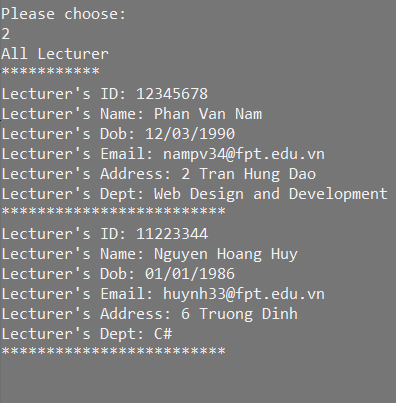


*Figure 6: View all student*

* Description
* When the user selects 2, the software will display a list of all students who have been added to the software.

# **View all lecturer**

* Figure



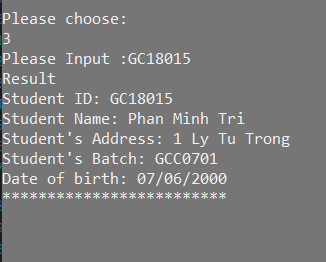
*Figure 7: View all lecturer*

* Description
* When the user selects 2, the software will display a list of all lecturer who have been added to the software.

# **Search**

# **Student**

* Figure



*Figure 8: Search student*

* Description
* When the user selects 3, the software will prompt the user to enter relevant information of the student to find such as ID, Name, Address, Email of the student to search. Users can simply type part of the name, ID, Email, Batch, or Address to find complete information of the student.
* Code

public static void search(List<Student> lstStudent)

{

Console.Write("Please Input :");

string strSearch = Console.ReadLine();

Console.WriteLine("Result");

foreach (var student in lstStudent)

{

if (student.IdStudent.ToLower().Contains(strSearch.ToLower()) || student.Name.ToLower().Contains(strSearch.ToLower())

|| student.Address.ToLower().Contains(strSearch.ToLower()) || student.Email.ToLower().Contains(strSearch.ToLower()) || student.stdBatch.ToLower().Contains(strSearch.ToLower()))

{

Console.WriteLine("Student ID: " + student.IdStudent);

Console.WriteLine("Student Name: " + student.Name);

Console.WriteLine("Student's Address: " + student.Address);

Console.WriteLine("Student's Batch: " + student.stdBatch);

Console.WriteLine("Date of birth: " + student.Dob.ToString("MM/dd/yyyy"));

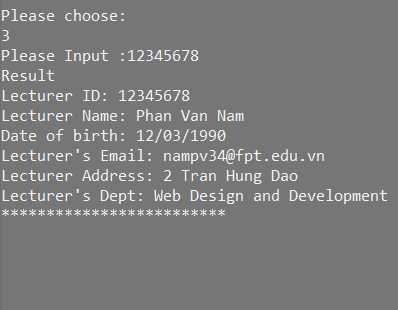
Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

# **Lecturer**

* Figure



*Figure 9: Search lecturer*

* Description
* When the user selects 3, the software will prompt the user to enter relevant information of the lecturer to find such as ID, Name, Address, Email of the lecturer to search. Users can simply type part of the name, ID, Email, Dept, or Address to find complete information about the lecturer.
* Code

public static void search(List<Lecturer> lstLecturer)

{

Console.Write("Please Input :");

string strSearch = Console.ReadLine();

Console.WriteLine("Result");

foreach (var lecturer in lstLecturer)

{

if (lecturer.IdLecturer.ToLower().Contains(strSearch.ToLower()) || lecturer.Name.ToLower().Contains(strSearch.ToLower())

|| lecturer.Address.ToLower().Contains(strSearch.ToLower()) || lecturer.Email.ToLower().Contains(strSearch.ToLower()) || lecturer.lecDept.ToLower().Contains(strSearch.ToLower()))

{

Console.WriteLine("Lecturer ID: " + lecturer.IdLecturer);

Console.WriteLine("Lecturer Name: " + lecturer.Name);

Console.WriteLine("Date of birth: " + lecturer.Dob.ToString("MM/dd/yyyy"));

Console.WriteLine("Lecturer's Email: " + lecturer.Email);

Console.WriteLine("Lecturer Address: " + lecturer.Address);

Console.WriteLine("Lecturer's Dept: " + lecturer.lecDept);

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

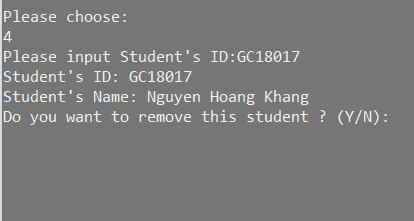
Console.ReadKey();

}

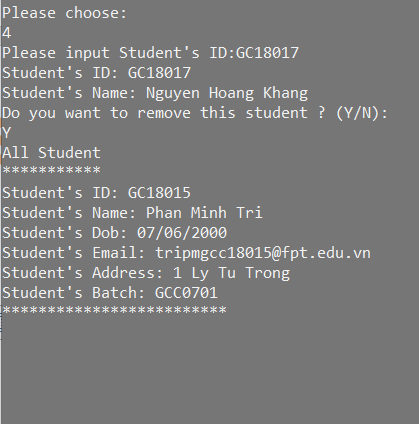
# **Delete**

# **Student**

* Figure



*Figure 10: Delete student 1.1*



*Figure 11: Delete student 1.2*

* Description
* When the user selects 4, the software will ask to enter the ID of the student to be deleted, then there will be 2 options Y (yes) / N (no) to confirm deletion of all information of a student.
* Code

public static void delete(List<Student> lstStudent)

{

Console.Write("Please input Student's ID:");

string stID = Console.ReadLine();

foreach(Student st in lstStudent)

{

if(st.IdStudent == stID)

{

Console.WriteLine("Student's ID: " + st.IdStudent);

Console.WriteLine("Student's Name: " + st.Name);

Console.WriteLine("Do you want to remove this student ? (Y/N): ");

string strResult = Console.ReadLine();

if(strResult.ToUpper() =="Y")

{

lstStudent.Remove(st);

break;

}

else

{

break;

}

}

}

Student.ShowInformation(lstStudent);

Console.ReadKey();

}

# **Lecturer**

* Figure

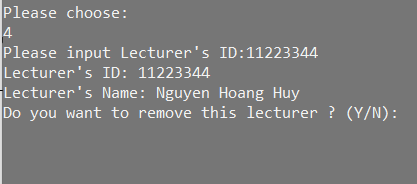


Figure : Delete lecturer 1.1

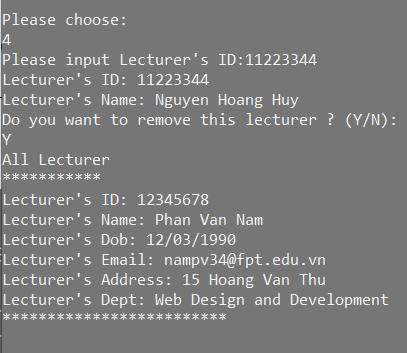


Figure : Delete lecturer 1.2

* Description
* When the user selects 4, the software will ask to enter the ID of the lecturer to be deleted, then there will be 2 options Y (yes) / N (no) to confirm deletion of all information of a lecturer.
* Code

public static void delete(List<Lecturer> lstLecturer)

{

Console.Write("Please input Lecturer's ID:");

string stID = Console.ReadLine();

foreach (Lecturer st in lstLecturer)

{

if (st.IdLecturer == stID)

{

Console.WriteLine("Lecturer's ID: " + st.IdLecturer);

Console.WriteLine("Lecturer's Name: " + st.Name);

Console.WriteLine("Do you want to remove this lecturer ? (Y/N): ");

string strResult = Console.ReadLine();

if (strResult.ToUpper() == "Y")

{

lstLecturer.Remove(st);

break;

}

else

{

break;

}

}

}

Lecturer.ShowInformation(lstLecturer);

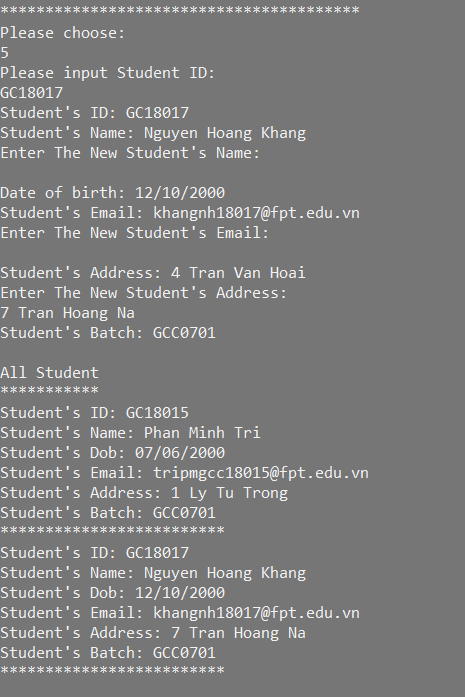
Console.ReadKey();

}

# **Update**

# **Student**

* Figure



*Figure 14: Update student*

* Description
* When the user selects 5, the software will ask to enter the student ID that needs to be updated. Then will display the order of each information for the user to update in turn name, dob, email, address, batch.
* Code

public static void update(List<Student> lstStudent)

{

Console.WriteLine("Please input Student ID:");

string stID = Console.ReadLine();

string strInput = "";

var st = lstStudent.Where(s => s.IdStudent == stID).FirstOrDefault<Student>();

Console.WriteLine("Student's ID: " + st.IdStudent);

//name

Console.WriteLine("Student's Name: " + st.Name);

Console.WriteLine("Enter The New Student's Name: ");

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.Name = strInput;

strInput = "";

}

//dob

Console.WriteLine("Date of birth: " + st.Dob.ToString("MM/dd/yyyy"));

//email

Console.WriteLine("Student's Email: " + st.Email);

Console.WriteLine("Enter The New Student's Email: ");

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.Email = strInput;

strInput = "";

}

//address

Console.WriteLine("Student's Address: " + st.Address);

Console.WriteLine("Enter The New Student's Address: ");

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.Address = strInput;

strInput = "";

}

//batch

Console.WriteLine("Student's Batch: " + st.stdBatch);

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.stdBatch = strInput;

strInput = "";

}

Student.ShowInformation(lstStudent);

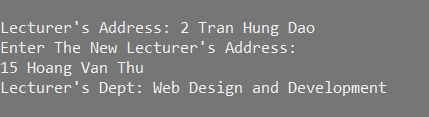
Console.ReadKey();

}

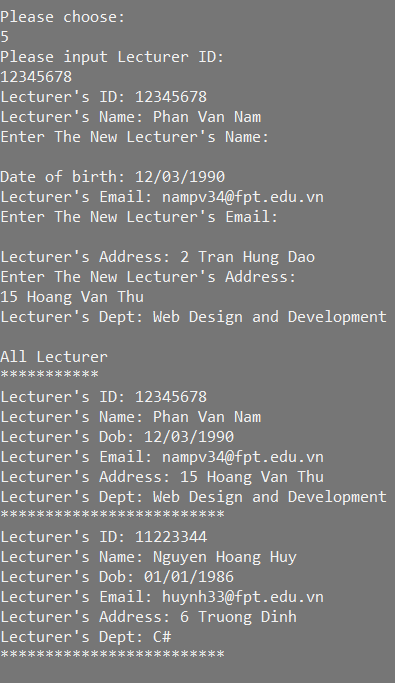
}

# **Lecturer**

* Figure



*Figure 15: Update lecturer 1.1*



*Figure 16: Update lecturer 1.2*

* Description
* When the user selects 5, the software will ask to enter the lecturer ID that needs to be updated. Then will display the order of each information for the user to update in turn name, dob, email, address, dept.
* Code

public static void update(List<Lecturer> lstLecturer)

{

Console.WriteLine("Please input Lecturer ID:");

string stID = Console.ReadLine();

string strInput = "";

var st = lstLecturer.Where(s => s.IdLecturer == stID).FirstOrDefault<Lecturer>();

Console.WriteLine("Lecturer's ID: " + st.IdLecturer);

//name

Console.WriteLine("Lecturer's Name: " + st.Name);

Console.WriteLine("Enter The New Lecturer's Name: ");

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.Name = strInput;

strInput = "";

}

//dob

Console.WriteLine("Date of birth: " + st.Dob.ToString("MM/dd/yyyy"));

//email

Console.WriteLine("Lecturer's Email: " + st.Email);

Console.WriteLine("Enter The New Lecturer's Email: ");

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.Email = strInput;

strInput = "";

}

//address

Console.WriteLine("Lecturer's Address: " + st.Address);

Console.WriteLine("Enter The New Lecturer's Address: ");

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.Address = strInput;

strInput = "";

}

//dept

Console.WriteLine("Lecturer's Dept: " + st.lecDept);

strInput = Console.ReadLine();

if (strInput.Length > 0)

{

st.lecDept = strInput;

strInput = "";

}

Lecturer.ShowInformation(lstLecturer);

Console.ReadKey();

}

}

# **Test case**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Test case** | **Function** | **Input Data** | **Expected output** | **Actual output** | **Evaluation** |
| 1. | Management Student | Show Student Menu | The user enters 1 to access the "ManageStudents" screen. | Display menu “Manage  Students” | Display menu “ManageStudents” | Pass |
| 2. | Management Lecturers | Show Lecturer Menu | The user enters 2 to access the "ManageLecturers" screen. | Display menu "Manage  Lecturers” | Display menu "ManageLecturers” | Pass |
| 3. | Exit | Exit the system | The user enters 3 to “Exit” the system | Exit management system | Exit management system | Pass |
| 4. | The user enters 4 to access the "ManageStudents" screen. | Select “Menu” | User enter a number other than 1 | Come back main  menu | Come back main menu | Pass |
| 5. | The user enters 4 to access the "ManageLecturers" screen. | Select “Menu” | User enter a number other than 2 | Come back main  menu | Come back main menu | Pass |
| 6. | The user enters the symbol to access the "ManageStudents" screen. | Select “Menu” | The user enters the symbol to system | Come back main  menu | Come back main  menu | Pass |
| 7. | The user enters the symbol to access the "ManageLecturers" screen. | Select “Menu” | The user enters the symbol to system | Come back main  menu | Come back main  menu | Pass |
| 8. | Add student | System display student add interface screen | User selects 1  to perform the  function "Add  student" | The system shows the add student interface | The system shows the add student interface | Pass |
| 9. | Add student | System display student add interface screen | User enter a number other than 1 | The system does not display the interface for adding students | The system does not display the interface for adding students | Pass |
| 10. | Add student | Check Student ID:  Input ID The student ID of the form like GTxxxxx or GCxxxxx (x: is a digit) | ID: GC18015  Name: Phan Minh Tri  Email: tripmgcc18015@  fpt.edu.vn  Address: 1 Ly Tu Trong  Day of birth: 07/06/2000  Batch: GCC0701 | Student information is added to the list. Display message. | Student information is added to the list. Display message. | Pass |
| 11. | Add student | Check Student ID:  Input ID The student ID of the form like GTxxxxx or GCxxxxx (x: is a digit) | ID: GX18015 | The system notifies you to re-enter the ID. | The system notifies you to re-enter the ID | Pass |
| 12. | View all students | Show all student (in the Student menu) | User selects 2  to perform the  function  "View all  student's | Display student information view interface | Display student information view interface | Pass |
| 13. | Search Student | System display student search interface screen | Enter number 3 | The system shows display student search interface screen | The system shows display student search interface screen | Pass |
| 14. | Search Student | System display student search interface screen | User enter a number other than 3 | The system does not display the interface for search students | The system does not display the interface for search students | Pass |
| 15. | Search Student | Test the search function (Users can search for student information by entering 1 of the following information: ID, Name Address, Batch) | Enter ID: GC18015 (ID already exists in the list) | The system displays all student information with the ID: GC18015 | The system displays all student information with the ID: GC18015 | Pass |
| 16. | Search Student | Test the search function | Enter ID: GC1234 (ID does not exist in the list) | Student information is not displayed, the system shows an announcement | Student information is not displayed, the system shows an announcement | Pass |
| 17. | Search Student | Test the search function | Enter Name: Phan Minh Tri (student name already exists in the list) | The system displays all student information with the name Phan Minh Tri | The system displays all student information with the name Phan Minh Tri | Pass |
| 18. | Search Student | Test the search function | Enter Name: Phan Minh Nam (student name does not exist in the list) | Student information is not displayed, the system shows an announcement | Student information is not displayed, the system shows an announcement | Pass |
| 19. | Delete Student | System display student Delete interface screen | Enter number 4 | System shows display student delete interface screen | System shows display student delete interface screen | Pass |
| 20. | Delete Student | System display student Delete interface screen | User enter a number other than 4 | Cannot access the student delete function interface | Cannot access the student delete function interface | Pass |
| 21. | Delete Student | Check input Student ID (in the delete function) | Enter ID: GC18017 (ID already exists in the list) | System shows display student (ID, Name, Day of birth, Email).  The system shows the notice "Do you want to remove this student?" | System shows display student (ID, Name, Day of birth, Email).  The system shows the notice "Do you want to remove this student?" | Pass |
| 22. | Delete Student | Check input Student ID (in the delete function) | Enter ID: GC18019 (ID does not exist in the list) | Student information is not displayed, the system shows an announcement | Student information is not displayed, the system shows an announcement | Pass |
| 23. | Delete Student | Check Delete function | Enter: Y | Deleted successfully | Deleted successfully | Pass |
| 24. | Delete Student | Check Delete function | Enter: N | The system will notify | The system will notify | Pass |
| 25. | Delete Student | Check Delete function | Enter another | The system will notify | The system will notify | Pass |
| 26. | Update Student | System display student update interface screen | Enter number 5 | System shows display student update interface screen | System shows display student update interface screen | Pass |
| 27. | Update Student | System display student update interface screen | User enter a number other than 5 | Cannot access the student update function interface | Cannot access the student update function interface | Pass |
| 28. | Update Student | Check input Student ID  (in the update function) | Enter ID: GC18015 (ID already exists in the list) | System shows display student update interface screen.  User can change Name, Email, Address, Date of birth, Batch. | System shows display student update interface screen.  User can change Name, Email, Address, Date of birth, Batch. | Pass |
| 29. | Update Student | Check input Student ID  (in the update function) | Enter ID: GC18019 (ID does not exists in the list) | Student information is not displayed, the system shows an announcement | Student information is not displayed, the system shows an announcement | Pass |
| 30. | Add lecturer | System display lecturer add interface screen | User selects 1  to perform the  function "Add  lecturer" | The system shows the add lecturer interface | The system shows the add lecturer interface | Pass |
| 31. | Add lecturer | System display lecturer add interface screen | User enter a number other than 1 | The system does not display the interface for adding lecturers | The system does not display the interface for adding lecturers | Pass |
| 32. | Add lecturer | Check Lecturer ID: Input ID The lecturer ID of the form like GTxxxxx or GCxxxxx (x: is a digit) | ID: 12345678  Name: Phan Van Nam  Email: nampv34@  fpt.edu.vn  Address: 15 Hoang Van Thu  Day of birth: 12/03/1990  Dept: Web Design and Development | Lecturer information is added to the list. Display message. | Lecturer information is added to the list. Display message. | Pass |
| 33. | Add lecturer | Check Lecturer ID: Input ID The lecturer ID of the form like GTxxxxx or GCxxxxx (x: is a digit) | ID: GX18045 | The system notifies you to re-enter the ID. | The system notifies you to re-enter the ID | Pass |
| 34. | View all lecturers | Show all lecturers (in the Lecturer menu) | User selects 2  to perform the  function  "View all  lecturer's | Display lecturer information view interface | Display lecturer information view interface | Pass |
| 35. | Search Lecturer | System display lecturer search interface screen | Enter number 3 | The system shows display lecturer search interface screen | The system shows display lecturer search interface screen | Pass |
| 36. | Search Lecturer | System display lecturer search interface screen | User enter a number other than 3 | The system does not display the interface for search lecturers | The system does not display the interface for search lecturers | Pass |
| 37. | Search Lecturer | Test the search function (Users can search for lecturer information by entering 1 of the following information: ID, Name Address, Dept) | Enter ID: 12345678 (ID already exists in the list) | The system displays all lecturer information with the ID: 12345678 | The system displays all lecturer information with the ID: 12345678 | Pass |
| 38. | Search Lecturer | Test the search function | Enter ID: GC1256 (ID does not exist in the list) | Lecturer information is not displayed, the system shows an announcement | Lecturer information is not displayed, the system shows an announcement | Pass |
| 39. | Search Lecturer | Test the search function | Enter Name: Phan Van Nam (lecturer name already exists in the list) | The system displays all lecturer information with the name Phan Van Nam | The system displays all lecturer information with the name Phan Van Nam | Pass |
| 40. | Search Lecturer | Test the search function | Enter Name: Phan Kim Nam (lecturer name does not exist in the list) | Lecturer information is not displayed, the system shows an announcement | Lecturer information is not displayed, the system shows an announcement | Pass |
| 41. | Delete Lecturer | System display lecturer Delete interface screen | Enter number 4 | System shows display lecturer delete interface screen | System shows display lecturer delete interface screen | Pass |
| 42. | Delete Lecturer | System display lecturer Delete interface screen | User enter a number other than 4 | Cannot access the lecturer delete function interface | Cannot access the lecturer delete function interface | Pass |
| 43. | Delete Lecturer | Check input Lecturer ID (in the delete function) | Enter ID: 11223344 (ID already exists in the list) | System shows display lecturer (ID, Name, Day of birth, Email).  The system shows the notice "Do you want to remove this lecturer?" | System shows display lecturer (ID, Name, Day of birth, Email).  The system shows the notice "Do you want to remove this lecturer?" | Pass |
| 44. | Delete Lecturer | Check input Lecturer ID (in the delete function | Enter ID: G12356 (ID does not exist in the list) | Lecturer information is not displayed, the system shows an announcement | Lecturer information is not displayed, the system shows an announcement | Pass |
| 45. | Delete Lecturer | Check Delete function | Enter: Y | Deleted successfully | Deleted successfully | Pass |
| 46. | Delete Lecturer | Check Delete function | Enter: N | The system will notify | The system will notify | Pass |
| 47 | Delete Lecturer | Check Delete function | Enter another | The system will notify | The system will notify | Pass |
| 48. | Update Lecturer | System display lecturer update interface screen | Enter number 5 | System shows display lecturer update interface screen | System shows display lecturer update interface screen | Pass |
| 49. | Update Lecturer | System display lecturer update interface screen | User enter a number other than 5 | Cannot access the lecturer update function interface | Cannot access the lecturer update function interface | Pass |
| 50. | Update Lecturer | Check input Lecturer ID  (in the update function) | Enter ID: 12345678 (ID already exists in the list) | System shows display lecturer update interface screen.  User can change Name, Email, Address, Date of birth, Dept. | System shows display lecturer update interface screen.  User can change Name, Email, Address, Date of birth, Dept. | Pass |
| 51. | Update Lecturer | Check input Lecturer ID  (in the update function) | Enter ID: G12356 (ID does not exists in the list) | Lecturer information is not displayed, the system shows an announcement | Lecturer information is not displayed, the system shows an announcement | Pass |

*Table 1: Test case*

# **Discuss a range of design patterns with relevant examples of creational, structural and behavioural pattern types.**

# **Singleton**

# **What is Singleton Design Pattern?**

* Provides a global entry point that ensures that a class has only one case.
* A singleton class is one that allows only one instance to be created by itself and usually provides simple access to that instance.
* Most importantly, singletons do not permit instance construction to specify certain parameters so a second request for an instance with a different parameter could be troublesome! (The factory pattern is more suitable if the same instance can be reached with the same parameter for all requests.) (Alle, n.d.)

# **Real-Life Example**

* Assume you're a member of a sporting team that's competing in a tournament. When the team plays another team, the captains of both teams must conduct a coin flip, according to the laws of the game. If your team requires a captain, you must first appoint someone. Your team must have only one Captain.

# **What are the Advantages of using the Singleton Pattern in C#?**

* The singleton interface pattern in C# provides the first and most important advantage of handling concurrent access to a shared resource. In other words, if a resource is shared by several clients at the same time, the singleton architecture pattern efficiently handles concurrent access to that resource.
* It has Static Initialization and can be lazy-loaded.
* To exchange non-changing common data in an application, such as master data and configuration data. The objects must then be cached in memory.
* Since it offers a single global point of entry to a particular instance, it is easy to handle.
* To reduce the overhead of repeatedly instantiating a large entity. (dotnettutorials.net, n.d.)

# **Code example**

* Singleton.cs:

namespace SingletonDemo

{

public sealed class Singleton

{

private static int counter = 0;

private static Singleton instance = null;

public static Singleton GetInstance

{

get

{

if (instance == null)

instance = new Singleton();

return instance;

}

}

private Singleton()

{

counter++;

Console.WriteLine("Counter Value " + counter.ToString());

}

public void PrintDetails(string message)

{

Console.WriteLine(message);

}

}

}

* Program.cs:

namespace SingletonDemo

{

class Program

{

static void Main(string[] args)

{

Singleton fromTeachaer = Singleton.GetInstance;

fromTeachaer.PrintDetails("From Teacher");

Singleton fromStudent = Singleton.GetInstance;

fromStudent.PrintDetails("From Student");

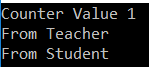
Console.ReadLine();

}

}

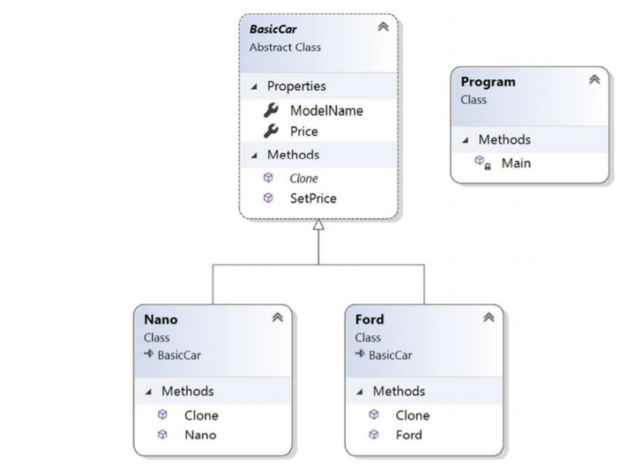
}

* Output:



*Figure 17: Singleton output*

# **Singleton UML diagram example**



*Figure 18: Singleton UML diagram example*

# **Builder**

# **What is Builder Pattern**

* When building complex structures with several elements, the Builder pattern comes in handy. The construction phase of an entity should be independent of these components; in other words, how these components are built should have little bearing on the production process. Furthermore, you should be able to generate various versions of the objects using the same construction technique. (refactoring.guru., n.d.)

# **Real-Life Example:**

* To complete an order for a computer, various hardware components are configured depending on customer specifications. For example, a customer with an Intel processor can choose a 1 TB hard disk, while a customer with an AMD processor can choose a 500 GB hard disk.

# **What are the Advantages of using the Builder Pattern in C#?**

* The parameters of the constructor have been reduced and are now given in highly readable method calls.
* The builder architecture pattern also decreases the number of constructor parameters, reducing the need to pass null for optional constructor parameters.
* The whole entity is still instantiated.
* Immutable objects can be generated without requiring a great deal of complex logic during the object development process. (geeksforgeeks.org, n.d.)

# **Code example**

interface HousePlan

{

public void setBasement(String basement);

public void setStructure(String structure);

public void setRoof(String roof);

public void setInterior(String interior);

}

class House implements HousePlan

{

private String basement;

private String structure;

private String roof;

private String interior;

public void setBasement(String basement)

{

this.basement = basement;

}

public void setStructure(String structure)

{

this.structure = structure;

}

public void setRoof(String roof)

{

this.roof = roof;

}

public void setInterior(String interior)

{

this.interior = interior;

}

}

interface HouseBuilder

{

public void buildBasement();

public void buildStructure();

public void bulidRoof();

public void buildInterior();

public House getHouse();

}

class IglooHouseBuilder implements HouseBuilder

{

private House house;

public IglooHouseBuilder()

{

this.house = new House();

}

public void buildBasement()

{

house.setBasement("Ice Bars");

}

public void buildStructure()

{

house.setStructure("Ice Blocks");

}

public void buildInterior()

{

house.setInterior("Ice Carvings");

}

public void bulidRoof()

{

house.setRoof("Ice Dome");

}

public House getHouse()

{

return this.house;

}

}

class TipiHouseBuilder implements HouseBuilder

{

private House house;

public TipiHouseBuilder()

{

this.house = new House();

}

public void buildBasement()

{

house.setBasement("Wooden Poles");

}

public void buildStructure()

{

house.setStructure("Wood and Ice");

}

public void buildInterior()

{

house.setInterior("Fire Wood");

}

public void bulidRoof()

{

house.setRoof("Wood, caribou and seal skins");

}

public House getHouse()

{

return this.house;

}

}

class CivilEngineer

{

private HouseBuilder houseBuilder;

public CivilEngineer(HouseBuilder houseBuilder)

{

this.houseBuilder = houseBuilder;

}

public House getHouse()

{

return this.houseBuilder.getHouse();

}

public void constructHouse()

{

this.houseBuilder.buildBasement();

this.houseBuilder.buildStructure();

this.houseBuilder.bulidRoof();

this.houseBuilder.buildInterior();

}

}

class Builder

{

public static void main(String[] args)

{

HouseBuilder iglooBuilder = new IglooHouseBuilder();

CivilEngineer engineer = new CivilEngineer(iglooBuilder);

engineer.constructHouse();

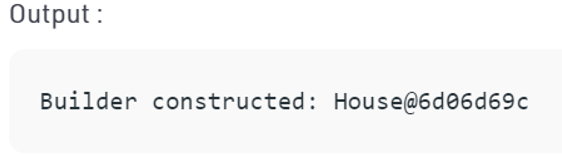
House house = engineer.getHouse();

System.out.println("Builder constructed: " + house);

}

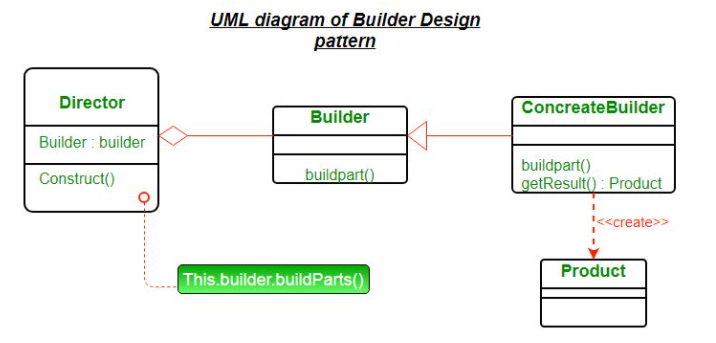
}

* Output code example



*Figure 19: Output code*

# **Builder UML diagram example**



*Figure 20: UML Diagram of Builder Design Pattern*

* **Product** – The product class defines the type of complex object that would be created by the builder pattern.
* **Builder** – This abstract base class defines all of the steps that must be taken in order to correctly create a product. Since the builder's actual functionality is carried out in specific subclasses, each step is normally abstract. The GetProduct method returns the final product. In lieu of the builder class, a basic interface is often used.
* **ConcreteBuilder** – Builder can be inherited by as many concrete builder groups as there are concrete builder groups. These groups provide the features needed to construct a particular complex product.
* **Director** - The algorithm that generates the final product object is in control of the director-class. A director object's Construct method is named after it is developed. The method includes a parameter that is used to grab the concrete builder object that will be used to construct the product. The director then calls the concrete builder's methods in the proper order to create the product object. Once the process is over, the GetProduct method of the builder object can be used to return the product.

# **Adapter**

# **What is Adapter Design Pattern?**

* The adapter is a structural architecture pattern that enables incompatible objects to work together. The Adapter serves as a container for two items. It intercepts calls for one object and converts them to a format and interface that the second object recognizes.

# **Example**

* The Adapter pattern is pretty common in C # code. It's very often used in systems based on some legacy code. In such cases, Adapters make legacy code work with modern classes.

# **What are the Advantages of using the Adapter Pattern in C#?**

* Aids in achieving reusability and flexibility.
* Since the client class is not required to use a separate interface, it may use polymorphism to differentiate between adapter implementations.

# **Code example**

// Java implementation of Adapter pattern

interface Bird

{

// birds implement Bird interface that allows

// them to fly and make sounds adaptee interface

public void fly();

public void makeSound();

}

class Sparrow implements Bird

{

// a concrete implementation of bird

public void fly()

{

System.out.println("Flying");

}

public void makeSound()

{

System.out.println("Chirp Chirp");

}

}

interface ToyDuck

{

// target interface

// toyducks dont fly they just make

// squeaking sound

public void squeak();

}

class PlasticToyDuck implements ToyDuck

{

public void squeak()

{

System.out.println("Squeak");

}

}

class BirdAdapter implements ToyDuck

{

// You need to implement the interface your

// client expects to use.

Bird bird;

public BirdAdapter(Bird bird)

{

// we need reference to the object we

// are adapting

this.bird = bird;

}

public void squeak()

{

// translate the methods appropriately

bird.makeSound();

}

}

class Main

{

public static void main(String args[])

{

Sparrow sparrow = new Sparrow();

ToyDuck toyDuck = new PlasticToyDuck();

// Wrap a bird in a birdAdapter so that it

// behaves like toy duck

ToyDuck birdAdapter = new BirdAdapter(sparrow);

System.out.println("Sparrow...");

sparrow.fly();

sparrow.makeSound();

System.out.println("ToyDuck...");

toyDuck.squeak();

// toy duck behaving like a bird

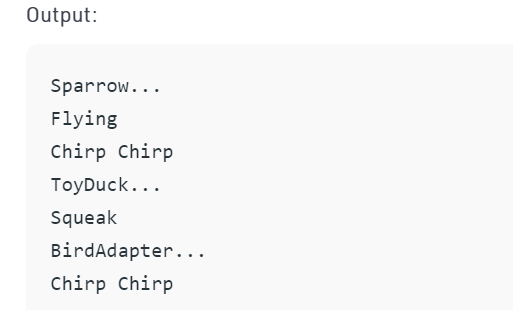
System.out.println("BirdAdapter...");

birdAdapter.squeak();

}

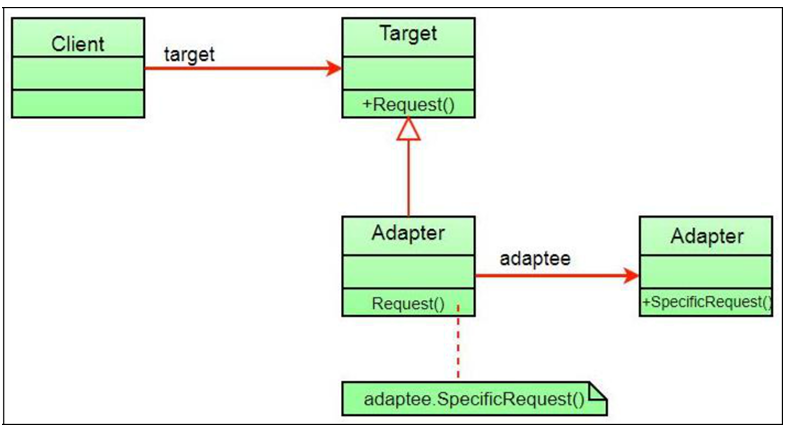
}

* Output:



*Figure 21: Output code example*

# **Adapter UML diagram example**



*Figure 22: Adapter UML diagram example*

* The client just sees the target interface, not the converter. The adapter implements the aim interface. Adapter delegated all demands to Adaptee.

# **Iterator**

# **What is Iterator Design Pattern?**

* The iterative design pattern is a common pattern that offers a useful abstraction. The iterator pattern is used to view and navigate the items of a collection without having to consider or reveal the collection's underlying structure.
* Iterator is a behavioral interface pattern in which you use an iterator to sequentially enumerate the items in the container, i.e., the set. Behavioral design patterns are ones that control object collaboration and responsibility delegation among objects.

# **Real-Life example:**

* Assume there are two companies, Company A and Company B. Company A stores their employee information (such as each employee's name, address, and compensation details) in a linked list data system, while Company B stores their employee data in a collection. When the two companies plan to combine.
* In this case, the Iterator pattern comes in handy so you don't have to write any code from scratch. In this case, you can provide a shared interface from which both companies can access the data. You can call such methods without having to rewrite the code.

# **What are the Advantages of using the Iterator Pattern in C#?**

* By offering tried-and-true architecture options, design patterns will help developers save time. The designers use the model prototype again and again since it helps them solve a lot of small problems. These design trends allow script legibility for coding developers and engineers.
* Design patterns provide short and simple ideas in a generic style that does not necessitate any specific expertise. These solutions can be used to address both major and small problems of software creation.
* These designs are often used by developers to express basic and well-known transmission names. The old and common style trends can be upgraded and updated on a regular basis. (Pedamkar, n.d.)

# **Code example**

class Program

{

public class Weeks //Aggregate object

{

private string[] weeks = new string[]{

"Monday",

"Tuesday",

"Wednesday",

"Thursday",

"Friday",

"Saturday",

"Sunday"

};

public IWeeksIterator GetWeeksIterator()

{

//creates an Iterator object

return new WeeksIterator(weeks);

}

}

public interface IWeeksIterator //Iterator interface

{

string Current { get; }

bool MoveNext();

}

public class WeeksIterator : IWeeksIterator //Iterator object

{

private readonly string[] weeks;

private int position;

public WeeksIterator(string[] weeks)

{

this.weeks = weeks;

this.position = -1;

}

public string Current => weeks[position];

public bool MoveNext()

{

if (++position == weeks.Length) return false;

return true;

}

}

static void Main(string[] args)

{

var weeks = new Weeks();

var iterator = weeks.GetWeeksIterator();

while (iterator.MoveNext())

{

Console.WriteLine(iterator.Current);

}

Console.ReadLine();

}

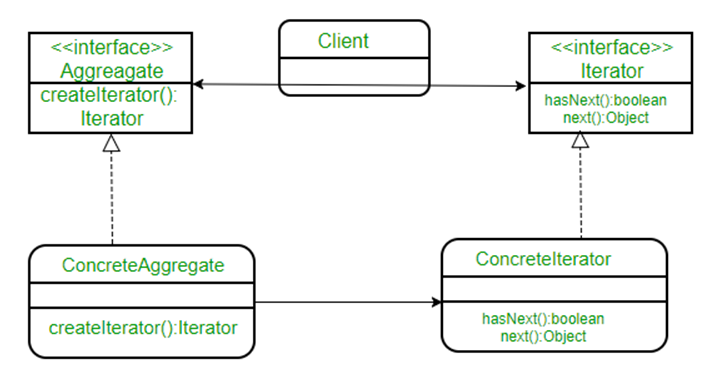
}

* Output:



*Figure 23: Result - Iterator Design Pattern*

# **Iterator UML diagram example:**



*Figure 24: Class diagram - Iterator Design Pattern*

* Aggregate is a popular interface for the client that distinguishes it from the execution of your collection of objects. ConcreteAggregate implements CreateIterator(), which returns an iterator for its collection. Each ConcreteAggregate is responsible for creating a ConcreteIterator that can iterate over its collection of properties. In addition to next() / hasNext(), the iterator interface offers a range of methods for traversing or changing the array, such as scan, delete, and so on.
* Have a look at a scenario and see how this happens. Assume we're creating a notification bar for our app that displays all of the alerts in a notification collection. NotificationCollection provides an iterator for traversing the elements without disclosing how the collection (array in this case) was added to the Client.

# **Advantages Iterator Design Pattern**

* Since the iterator employs the Same Responsibility and Open / Closed SOLID concepts, the code is simpler to use, comprehend, and evaluate.
* The Single Responsibility Principle enables one to clean up the traversal algorithm's client and sets.
* The Open / Closed Principle allows for the introduction of new kinds of collections and iterators without causing any problems.
* Since each iterator object has its own iteration condition, it is possible to iterate over the same set in parallel.
* Since the client/context does not use a complicated interface, the programming is cleaner, and the framework is more portable and reusable.

# **Observer**

# **What is Observer**

* The Observer pattern specifies a one-to-many dependence between objects such that when one object changes state, all of its dependents are immediately informed and modified.
* The Observer pattern allows any object that implements a subscriber interface to subscribe and unsubscribe to and from these events.
* This pattern contains a large number of observers (objects) who are observing a specific subject (also an object). When an adjustment is made inside the subject, observers would like to be updated. As a result, they sign up for that topic. When they lose interest in a topic, they literally de-register from it. (refactoring.guru, n.d.)

# **Real-Life example:**

* Consider a star with a large number of social media followers. With of these fans needs to be kept up to date on the latest news from their favorite artists. Then they accompany the celebrity until the attention wanes. When they lose confidence in a celebrity, they actually stop watching them. Consider yourself as an observer of one of these fans or supporters, with the celebrity as the focus.

# **What are the Advantages of using the Observer Pattern in C#?**

* Gives interacting structures a loosely coupled GUI. Loosely coupled objects are adaptable to changing specifications. Loose coupling means that the interacting objects will know little about one another in this situation.
* The observer sequence describes this loose coupling as follows:
* The subject is only mindful that the observer implements the Observer interface. It isn't much else.
* There is no need to change the Topic to introduce or exclude observers.
* Object and observer classes are separate entities that can be reused.

# **Code example**

**using** System;

**using** System.Collections.Generic;

**using** System.Threading;

**namespace** **RefactoringGuru**.DesignPatterns.Observer.Conceptual

{

**public** **interface** **IObserver**

{

// Receive update from subject

**void** Update(ISubject subject);

}

**public** **interface** **ISubject**

{

// Attach an observer to the subject.

**void** Attach(IObserver observer);

// Detach an observer from the subject.

**void** Detach(IObserver observer);

// Notify all observers about an event.

**void** Notify();

}

// The Subject owns some important state and notifies observers when the

// state changes.

**public** **class** **Subject** : ISubject

{

// For the sake of simplicity, the Subject's state, essential to all

// subscribers, is stored in this variable.

**public** **int** State { **get**; **set**; } = -0;

// List of subscribers. In real life, the list of subscribers can be

// stored more comprehensively (categorized by event type, etc.).

**private** List<IObserver> \_observers = **new** List<IObserver>();

// The subscription management methods.

**public** **void** Attach(IObserver observer)

{

Console.WriteLine("Subject: Attached an observer.");

**this**.\_observers.Add(observer);

}

**public** **void** Detach(IObserver observer)

{

**this**.\_observers.Remove(observer);

Console.WriteLine("Subject: Detached an observer.");

}

// Trigger an update in each subscriber.

**public** **void** Notify()

{

Console.WriteLine("Subject: Notifying observers...");

**foreach** (**var** **observer** **in** \_observers)

{

observer.Update(**this**);

}

}

// Usually, the subscription logic is only a fraction of what a Subject

// can really do. Subjects commonly hold some important business logic,

// that triggers a notification method whenever something important is

// about to happen (or after it).

**public** **void** SomeBusinessLogic()

{

Console.WriteLine("\nSubject: I'm doing something important.");

**this**.State = **new** Random().Next(0, 10);

Thread.Sleep(15);

Console.WriteLine("Subject: My state has just changed to: " + **this**.State);

**this**.Notify();

}

}

// Concrete Observers react to the updates issued by the Subject they had

// been attached to.

**class** **ConcreteObserverA** : IObserver

{

**public** **void** Update(ISubject subject)

{

**if** ((subject **as** Subject).State < 3)

{

Console.WriteLine("ConcreteObserverA: Reacted to the event.");

}

}

}

**class** **ConcreteObserverB** : IObserver

{

**public** **void** Update(ISubject subject)

{

**if** ((subject **as** Subject).State == 0 || (subject **as** Subject).State >= 2)

{

Console.WriteLine("ConcreteObserverB: Reacted to the event.");

}

}

}

**class** **Program**

{

**static** **void** Main(**string**[] args)

{

// The client code.

**var** **subject** = **new** Subject();

**var** **observerA** = **new** ConcreteObserverA();

subject.Attach(observerA);

**var** **observerB** = **new** ConcreteObserverB();

subject.Attach(observerB);

subject.SomeBusinessLogic();

subject.SomeBusinessLogic();

subject.Detach(observerB);

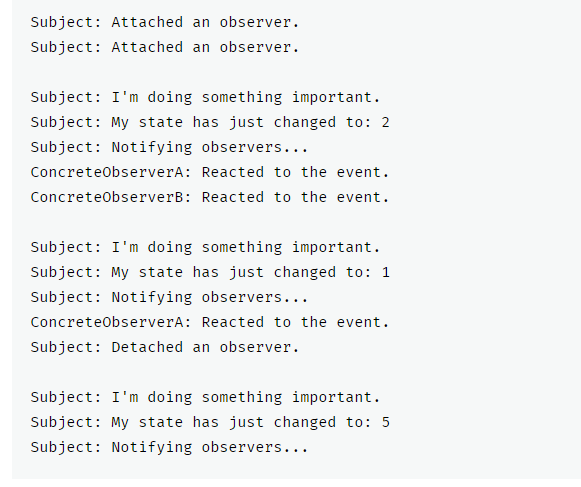
subject.SomeBusinessLogic();

}

}

}

* Output

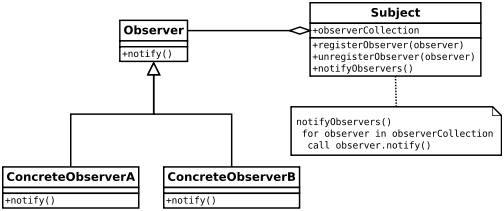


*Figure 25: Output Observer code Example.*

# **Advantages Observer Design Pattern**

* It adheres to the concept of loose coupling between objects that communicate with one another. It enables successful data transmission to other objects without requiring any changes to the Subject or Observer classes.
* Observers may be inserted or deleted at any time.

# **Observer UML diagram example**



*Figure 26: Observer UML diagram example*

* In this case, the interfaces are the Observer and the Subject (can be any abstract supertype not necessarily a java interface).
* Both observers that need data must apply the observer GUI.
* The notify() method of the observer interface determines the action to be taken when the subject provides details.
* The subject maintains a list of registered observers in an observed array, which is essentially a list of those that have been observed (subscribed).
* To add and delete observers, use the registered observer(observer) and unregisterObserver(observer) methods.
* Notify observers() is called when the data shifts and the observers need new information.

# References

1111111, n.d. [Online]   
Available at: https://www.educba.com/what-is-design-pattern-in-c-sharp/  
[Accessed 13 5 2021].

Alle, M., n.d. [Online]   
Available at: https://www.c-sharpcorner.com/UploadFile/8911c4/singleton-design-pattern-in-C-Sharp/#:~:text=There%20are%20various%20ways%20to%20implement%20the%20Singleton,reference%20to%20the%20single%20created%20in  
[Accessed 10 5 2021].

dotnettutorials.net, n.d. [Online]   
Available at: https://dotnettutorials.net/lesson/singleton-design-pattern/  
[Accessed 12 5 2021].

geeksforgeeks.org, n.d. [Online]   
Available at: https://www.geeksforgeeks.org/builder-design-pattern/#:~:text=Advantages%20of%20Builder%20Design%20Pattern&text=Builder%20design%20pattern%20also%20helps,optional%20parameters%20to%20the%20constructor.&text=Immutable%20objects%20can%20be%20build,logic%20i  
[Accessed 12 5 2021].

Pedamkar, P., n.d. [Online]   
Available at: https://www.educba.com/what-is-design-pattern-in-c-sharp/  
[Accessed 13 5 2021].

refactoring.guru., n.d. [Online]   
Available at: https://refactoring.guru/design-patterns/builder/csharp/example#:~:text=Builder%20is%20a%20creational%20design,using%20the%20same%20construction%20process.  
[Accessed 12 5 2021].

refactoring.guru, n.d. [Online]   
Available at: https://refactoring.guru/design-patterns/observer/csharp/example  
[Accessed 13 5 2021].