Faculty of Natural and Applied Sciences
Computer Science Department

CSC 323: Object-Oriented Design

Project – Report

Submitted by:

Student ID	Name	Major
20222056	Paul Sawaya	Computer Science
20221772	Tiana Chebly	Computer Science
20212695	Charbel Khoury	Computer Science

Project Description (around 500 words)

Welcome to PCT Hospital, a place where we care for people who are in need of medical help. We are located in Zouk Mosbeh and we are here to help anyone who comes through our doors, whether they are just visiting or need to stay with us for adequate treatment.

We have made our hospital using special ways of designing things, through design patterns, to make everything work smoothly and help our patients better.

If you are an inpatient and you are also monitored, the amount that you will be paying is 100$ multiplied by the number of stay-in days. Otherwise, the amount will be 80$ multiplied by the number of stay-in days. If you are an outpatient, then the amount will be 25$ multipled by the number of tests.

Now concerning the Design Patterns, we have used the Singleton design pattern, ensuring that PCT Hospital remains limited to a single, cohesive instance. The architectural decision ensures a simplified management and guarantees a unified approach to patient care.

Then, we have also used the Builder design pattern which plays a pivotal role in customizing the configuration of hospital rooms to provide to diverse patient needs So, whether someone needs a regular room or a First Class room, we can build it for them and make sure they feel comfortable during their stay.

Moreover, in our hospital system, the Adapter design pattern precisely aligns different classes to ensure compatibility. This pattern strictly follows the principle of interface adaptation, facilitating seamless communication between our Hospital Management System Connector and the broader Hospital Management System in order to save and get data about the entrants to the hospital.

When it comes to paying for treatment, we offer different ways to make it easy for everyone. You can pay with cash, use a credit card like Visa, or even use popular payment services like WHISH or OMT. We make sure to give you a receipt so you know exactly what you've paid for.

At PCT Hospital, we believe in keeping things simple and making sure everyone gets the care they need.

Class Diagram (UML)

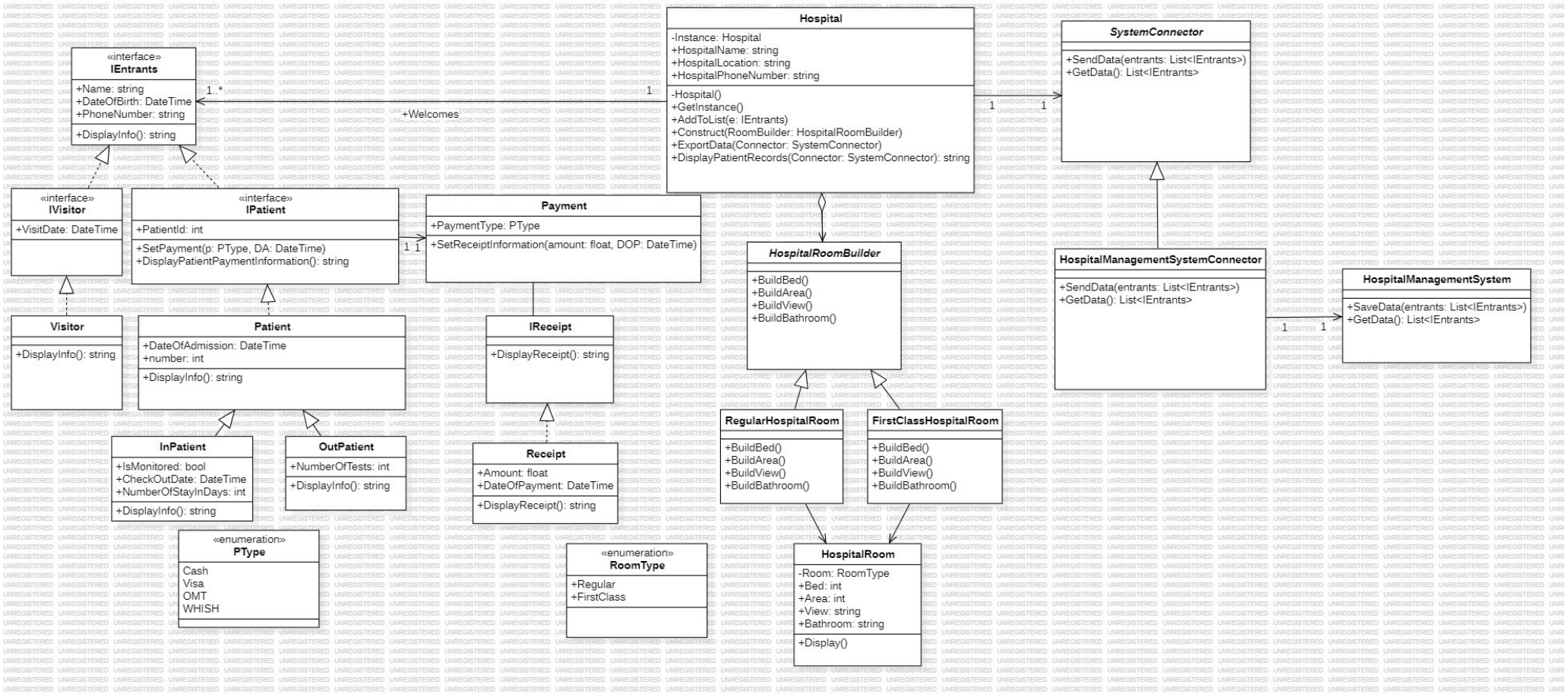

Figure 1. Hospital Class Diagram

Source Code (C#)

Form

```
using System;
using System.Text;

using OOD_Hospital_Project.AdapterPattern;
using OOD_Hospital_Project.SingletonPattern;
using OOD_Hospital_Project.BuilderPattern;
using OOD_Hospital_Project.Payments;
using OOD_Hospital_Project.Entrants;


namespace OOD_Proj_Final
{
    public partial class Form1 : Form
    {
        IEntrants Visitor1 = new Visitor("Paul S.", new DateTime(1999, 12, 23), "+961 3
333444", new DateTime(2024, 6, 5));
        IEntrants Visitor2 = new Visitor("Tiana C.", new DateTime(2004, 3, 1), "+961 3
322508", new DateTime(2019, 3, 2));
        IEntrants VisitorWihoutBirthDate = new Visitor("Charbel k.", new DateTime(2024, 5,
13));
        IPatient Patient1 = new InPatient("Jane BD.", new DateTime(1997, 9, 17), "+961 1
223377", new DateTime(2020, 4, 5), true, new DateTime(2020, 4, 15));
        IPatient Patient2 = new InPatient("Jad K.", new DateTime(2011, 6, 8), "+961 7 228899",
new DateTime(2019, 7, 9), false, new DateTime(2019, 8, 10));
        IPatient Patient3 = new OutPatient("Tony H.", new DateTime(2015, 3, 6), "+961 3
765456", new DateTime(2024, 4, 4), 3);

        public Form1()
        {
            InitializeComponent();
            Patient1.SetPayment(PType.Cash, new DateTime(2002, 9, 8));
            Patient2.SetPayment(PType.WHISH, new DateTime(2019, 8, 5));
            Patient3.SetPayment(PType.OMT, new DateTime(2024, 4, 4));
            var HospitalInstance = Hospital.GetInstance();

            HospitalInstance.AddToList(Visitor1);
            HospitalInstance.AddToList(Visitor2);
            HospitalInstance.AddToList(VisitorWihoutBirthDate);
            HospitalInstance.AddToList(Patient1);
            HospitalInstance.AddToList(Patient2);
            HospitalInstance.AddToList(Patient3);

        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "PCT Hospital NDU Branch By Your Side!";

        }

        private void button1_Click(object sender, EventArgs e)
        {

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
```

Fig 2. Namespaces Concept

```
        }

        private void label1_Click_1(object sender, EventArgs e)
        {

        }

        private void button1_Click_1(object sender, EventArgs e)
        {
            StringBuilder sb = new();
            sb.Append($"Name of the Hospital: {Hospital.GetInstance().HospitalName} \n" +
                $"Hospital Location: {Hospital.GetInstance().HospitalLocation} \n" +
                $"Hospital Phone Number: {Hospital.GetInstance().HospitalPhoneNumber}");
            MessageBox.Show(sb.ToString(), "Hospital Information");
        }

        private void button2_Click(object sender, EventArgs e)
        {

            SystemConnector systemConnector = new HospitalManagementSystemConnector();
            Hospital.GetInstance().ExportData(systemConnector);
            string result = Hospital.GetInstance().DisplayPatientRecords(systemConnector);
            MessageBox.Show(result, "Entrants Details");

        }

        private void button3_Click(object sender, EventArgs e)
        {

            HospitalRoomBuilder H1 = new RegularHospitalRoom();
            Hospital.GetInstance().Construct(H1);

            HospitalRoomBuilder H2 = new FirstClassHospitalRoom();
            Hospital.GetInstance().Construct(H2);

            string result = H1.Room.Display() +H2.Room.Display();
            MessageBox.Show(result, "Hospital Rooms");

        }

        private void button4_Click(object sender, EventArgs e)
        {
            StringBuilder PaymentResults = new();
            PaymentResults.Append(
             $"{Patient1.DisplayPatientPaymentInformation()}" +
              $"{Patient2.DisplayPatientPaymentInformation()}" +
             $"{Patient3.DisplayPatientPaymentInformation()}"
                           );
            MessageBox.Show(PaymentResults.ToString(), "Payment Details");
        }

        private void button5_Click(object sender, EventArgs e)
        {

        }
    }
}
```

Hospital

```
using System;
using System.Text;

using OOD_Hospital_Project.AdapterPattern;
using OOD_Hospital_Project.BuilderPattern;
using OOD_Hospital_Project.Entrants;

namespace OOD_Hospital_Project.SingletonPattern
{
    public partial class Hospital
    {
        private static Hospital Instance;
        public string HospitalName { get; set; }
        public string HospitalLocation { get; set; }
        public string HospitalPhoneNumber { get; set; }
        public List<IEntrants> PatientsRecords { get; set; } = new List<IEntrants>();
        List<IEntrants> Entrants { get; set; } = new List<IEntrants> { };


        private Hospital(string hospitalName, string hospitalLocation, string hospitalNumber)
        {

            HospitalName = hospitalName;
            HospitalLocation = hospitalLocation;
            HospitalPhoneNumber = hospitalNumber;
        }

        public static Hospital GetInstance()
        {
            if (Instance == null)
            {
                Instance = new Hospital("PCT Medical Center", "Zouk Mosbeh", "+961 9 111222");
            }

            return Instance;
        }

        public void AddToList(IEntrants e)
        {
            Entrants.Add(e);
        }

        public void Construct(HospitalRoomBuilder RoomBuilder)
        {
            RoomBuilder.BuildBed();
            RoomBuilder.BuildArea();
            RoomBuilder.BuildView();
            RoomBuilder.BuildBathroom();
        }

        public void ExportData(SystemConnector Connector)
        {
            Connector.SendData(Entrants);
        }

        public string DisplayPatientRecords(SystemConnector Connector)
        {
            List<IEntrants> temp = Connector.GetData();
            StringBuilder recordBuilder = new StringBuilder();

            foreach (IEntrants entrant in temp)
            {
                string patientId = entrant is Patient patient ? patient.PatientId.ToString() :
"N/A";
                string entrantType = entrant switch
                {
                    IVisitor => "Visitor",
```

Fig 3. Collection Concept (Lists)

Fig 4. Loops Concept (foreach)

```
                    InPatient => "InPatient",
                    OutPatient => "OutPatient",
                    _ => "Unknown"
                };

                recordBuilder.Append($"Entrant Number: {patientId}\n" +
                                     $"{entrant.DisplayInfo()}\n");
            }

            return recordBuilder.ToString();

        }
    }
}
```

IEntrants

```
namespace OOD_Hospital_Project.Entrants
{
        public interface IEntrants
        {
        public string Name { get; set; }
        public DateTime? DateOfBirth { get; set; }
        public string PhoneNumber { get; set; }
        string DisplayInfo();
        }


}
```

Fig 5. Abstraction Concept (Inteface)

IVisitor

```
namespace OOD_Hospital_Project.Entrants
{
    public interface IVisitor : IEntrants
    {
        DateTime VisitDate { get; set; }
        string DisplayInfo();
    }
}
```

Visitor

```
namespace OOD_Hospital_Project.Entrants
{
    public class Visitor : IVisitor
    {
        public string Name { get; set; }
        public DateTime? DateOfBirth { get; set; }
        public string PhoneNumber { get; set; }
        public DateTime VisitDate { get; set; }

        public Visitor(string n, DateTime DOB, string pn, DateTime vd)
        {
            Name = n;
            DateOfBirth = DOB;
            PhoneNumber = pn;
            VisitDate = vd;
        }

        public Visitor(string n, DateTime vd)
        {
            Name = n;
            DateOfBirth = null;
            PhoneNumber = "N/A";
            VisitDate = vd;
        }

        public string DisplayInfo()
```

Fig 6. Concrete Classes Concept

Fig 7. OverLoading Constructor Concept

```
        {
            string dateOfBirthString = DateOfBirth.HasValue ?
DateOfBirth.Value.ToShortDateString() : "N/A";
            return $"Entrant Type: Visitor\n" +
                   $"Name: {Name}\n" +
                   $"DateOfBirth: {dateOfBirthString}\n" +
                   $"PhoneNumber: {PhoneNumber}\n" +
                   $"Visit Date: {VisitDate.ToShortDateString()}\n";
        }
    }
 }
```

IPatient

```
using OOD_Hospital_Project.Payments;

namespace OOD_Hospital_Project.Entrants
{
    public interface IPatient : IEntrants
    {
        public int PatientId { get; set; }
        public Payment payment { get; set; }
        public void SetPayment(PType p, DateTime Dop);
        public string DisplayPatientPaymentInformation();
    }
}
```

Patient

```
using OOD_Hospital_Project.Payments;

namespace OOD_Hospital_Project.Entrants
{
    public class Patient : IPatient
    {
        public int PatientId { get; set; }
        public string Name { get; set; }
        public DateTime? DateOfBirth { get; set; }
        public string PhoneNumber { get; set; }
        public DateTime DateOfAdmission { get; set; }
        public Payment payment { get; set; }
        static int number = 1;

        public Patient(string name, DateTime dob, string pn, DateTime DA)
        {
            PatientId = number;
            Name = name;
            DateOfBirth = dob;
            PhoneNumber = pn;
            DateOfAdmission = DA;
            number++;
        }

        public virtual void SetPayment(PType p, DateTime Dop)
        {
        }

        public string DisplayPatientPaymentInformation()
        {
            string result = $"Patient Id: {PatientId}
\n{payment.DisplayPaymentInformation()}";
            return result;
        }

        public virtual string DisplayInfo()
        {
            string dateOfBirthString = DateOfBirth.HasValue ?
DateOfBirth.Value.ToShortDateString() : "N/A";
            return $"Name: {Name}\n" +
                $"DateOfBirth: {dateOfBirthString}\n" +
                $"PhoneNumber: {PhoneNumber}\n";        } } }
```

Fig 8. Properties Concept

Fig 9. Methods Concept

InPatient

```
using OOD_Hospital_Project.Payments;

namespace OOD_Hospital_Project.Entrants
{
    public class InPatient : Patient
    {
        public bool IsMonitored { get; set; }
        public int NumberOfStayInDays { get; set; }
        public DateTime CheckOutDate { get; set; }
        public InPatient(string name, DateTime dob, string pn, DateTime DA, bool Im, DateTime
checkOutDate) : base(name, dob, pn, DA)
        {
            IsMonitored = Im;
            CheckOutDate = checkOutDate;

            NumberOfStayInDays = (checkOutDate - DA).Days;
        }

        public override void SetPayment(PType p, DateTime Dop)
        {
            if (IsMonitored == true)
            {
                float amount = 100 * NumberOfStayInDays;
                payment = new(p, amount, Dop);
            }
            else {
                float amount = 80 * NumberOfStayInDays;
                payment = new(p, amount, Dop);
            }
        }

        public override string DisplayInfo()
        {
            return $"Entrant Type: InPatient\n" +
                   $"{base.DisplayInfo()}" +
                   $"Duration Of Stay: {NumberOfStayInDays} Days\n";
        }
    }
}
```

Fig 10. Overriding Methods Concept

OutPatient

```
using OOD_Hospital_Project.Payments;

namespace OOD_Hospital_Project.Entrants
{
    public class OutPatient : Patient
    {
        public int NumberOfTests { get; set; }

        public OutPatient(string name, DateTime dob, string pn, DateTime DA, int NOT) :
base(name, dob, pn, DA)
        {
            NumberOfTests = NOT;
        }

        public override void SetPayment(PType p, DateTime Dop)
        {
            float amount = 25 * NumberOfTests;
            payment = new(p, amount, Dop);
        }

        public override string DisplayInfo()
        {
            return $"Entrant Type: OutPatient\n" +
                   $"{base.DisplayInfo()}" +
                   $"Numbers Of Tests to be Done: {NumberOfTests}\n";
        } } }
```

Fig 11. Overriding Methods Concept

PType

```
namespace OOD_Hospital_Project.Payments
{
    public enum PType
    {
        Cash, Visa, OMT, WHISH }  }
```

Fig 12. Enumeration Concept

Payment

```
using System.Text;

namespace OOD_Hospital_Project.Payments
{
    public class Payment
    {
        IReceipt receipt;
        public PType PaymentType { get; set; }


        public Payment(PType p, float amount, DateTime Dop)
        {
            PaymentType = p;
            SetReceiptInformation(amount, Dop);

        }

        public void SetReceiptInformation(float amount, DateTime Dop)
        {
            receipt = new Receipt(amount, Dop);
        }

        public string DisplayPaymentInformation()
        {
            StringBuilder sb = new();
            string oldresult = receipt.DisplayReceipt();
            sb.Append($"{oldresult} \nThe Payment Type is {PaymentType}\n\n");
            return sb.ToString();
        }
    }
 }
```

IReceipt

```
namespace OOD_Hospital_Project.Payments
{
    public interface IReceipt
    {
        string DisplayReceipt();
    }
}
```

Receipt

```
using System.Text;
namespace OOD_Hospital_Project.Payments
{
    public class Receipt : IReceipt
    {
        public float Amount { get; set; }
        public DateTime DateOfPayment { get; set; }
        public Receipt(float amount, DateTime dateOfPayment)
        {
            Amount = amount;
            DateOfPayment = dateOfPayment;
        }

        public string DisplayReceipt()
        {
            StringBuilder sb = new StringBuilder();
            sb.Append($"The Total Amount is {Amount}$\n" +
                $"Date Of Payment: {DateOfPayment.ToShortDateString()}");
            return sb.ToString();
        }  } }
```

HospitalRoomBuilder

```
namespace OOD_Hospital_Project.BuilderPattern
{
    public abstract class HospitalRoomBuilder  {
        public HospitalRoom Room { get; set; }
        public abstract void BuildArea();
        public abstract void BuildView();
        public abstract void BuildBathroom();
        public abstract void BuildBed();

    }
}
```

Fig 13. Abstraction Concept

RoomType

```
namespace OOD_Hospital_Project.BuilderPattern
{
    public enum RoomType
    {
        Regular, FirstClass
 }
    }
```

Fig 14. Enumeration Concept

RegularHospitalRoom

```
namespace OOD_Hospital_Project.BuilderPattern
{
    public class RegularHospitalRoom : HospitalRoomBuilder
    {
        public RegularHospitalRoom()
        {
            Room = new HospitalRoom(RoomType.Regular);
        }
        public override void BuildArea()
        {
            Room.Area = 10;
        }
        public override void BuildView()
        {
            Room.View = "Regular View";
        }
        public override void BuildBathroom()
        {
            Room.Bathroom = "Simple Bathroom";
        }
        public override void BuildBed()
        {
            Room.Bed = 2;
        }
    }
}
```

FirstClassHospitalRoom

```
namespace OOD_Hospital_Project.BuilderPattern
{
    public class FirstClassHospitalRoom : HospitalRoomBuilder
    {
        public FirstClassHospitalRoom()
        {
            Room = new HospitalRoom(RoomType.FirstClass);
        }
        public override void BuildArea()
        {
            Room.Area = 20;
        }
        public override void BuildView()
        {
            Room.View = "Beautiful View";
        }
        public override void BuildBathroom()
        {
```

```
            Room.Bathroom = "Deluxe Bathroom";
        }
        public override void BuildBed()
        {
            Room.Bed = 1;
        }
    }

}
```

HospitalRoom

```
using System.Text;

namespace OOD_Hospital_Project.BuilderPattern
{
    public class HospitalRoom
    {
        private RoomType roomtype { get; set; }
        public int Bed { get; set; }
        public int Area { get; set; }
        public string Bathroom { get; set; }
        public string View { get; set; }

        public HospitalRoom(RoomType r)
        {
            roomtype = r;
        }

        public string Display()
        {
            StringBuilder sb = new StringBuilder();
            sb.Append($"The RoomType is {roomtype} \n" +
                $"The number of Bed is {Bed} \n" +
                $"The Area is {Area} m²\n" +
                $"The Bathroom is {Bathroom} \n" +
                $"The View is {View}\n\n");

            return sb.ToString();

        }
    }

}
```

SystemConnector

```
using OOD_Hospital_Project.Entrants;

namespace OOD_Hospital_Project.AdapterPattern
{
    public abstract class SystemConnector
    {
        public abstract void SendData(List<IEntrants> entrants);
        public abstract List<IEntrants> GetData();

    }
}
```

HospitalManagementSystemConnector

```
using OOD_Hospital_Project.Entrants;


namespace OOD_Hospital_Project.AdapterPattern
{
    public class HospitalManagementSystemConnector : SystemConnector
    {
        private HospitalManagementSystem HMS = new HospitalManagementSystem();
        public override void SendData(List<IEntrants> entrants)
        {

            HMS.SaveData(entrants);
        }

        public override List<IEntrants> GetData()
        {
            return HMS.GetData();
        }
    }

}
```

HospitalManagementSystem

```
using OOD_Hospital_Project.Entrants;

namespace OOD_Hospital_Project.AdapterPattern
{
    public class HospitalManagementSystem
    {
        public List<IEntrants> PatientRecords { get; set; }
        public void SaveData(List<IEntrants> entrants)
        {
            PatientRecords = entrants;
        }

        public List<IEntrants> GetData()
        {
            return PatientRecords;
        }
    }

 }
```

Descriptions of how and where you have used key concepts

Concept	**Description** (around 30 words each) refer to the class diagram and source code figures as you see fit
Abstraction (abstract classes and/or interfaces)	Abstract classes and interfaces are employed to decouple concrete classes, enabling flexibility in implementations. Examples include the "HospitalRoomBuilder" abstract class, which defines the blueprint for constructing hospital rooms and specifies methods for building the area, view, bathroom, and bed. In addition, within the "OOD_Hospital_Project.Entrants" namespace, the "IEntrants" interface outlines common properties and methods for entities entering the system, such as name, date of birth, phone number, and a method for displaying information. (Refer to Fig13 and Fig5)
Concrete classes	Concrete classes are used to create instances of objects and implement system functionality. Examples include "Visitor," representing individuals visiting the hospital, with attributes such as name, date of birth, phone number, and visit date.(Refer Fig6)
Namespaces	In the structure of the hospital management system, namespaces serve to categorize classes and interfaces according to the design patterns they adhere to. For instance, within the OOD_Hospital_Project namespace, we have distinct categories such as AdapterPattern, SingletonPattern, and BuilderPattern. Others, like Payments and Entrants, handle payment-related tasks and managing information about patients and staff. (Refer to Fig2)
Methods	Methods are used to implement functionalities related to the class's purpose. For instance, the "SetPayment" method is designed to set payment information, taking parameters for payment type and date of payment. The "DisplayPatientPaymentInformation" method displays information about patient payments, combining patient ID with payment details. Additionally, the "DisplayInfo" method is responsible for presenting general information about an entity, such as name, date of birth, and phone number.(Refer to Fig9)
Properties	They are used to represent the attributes of the classes. Examples include "PatientId" for identifying patients, "Name" for storing their names, "DateOfBirth" for their birthdates, "PhoneNumber" for contact information, "DateOfAdmission" for when they were admitted, "Payment" for handling payment details, and a static integer variable "number" for internal counting purposes. (Refer to Fig8)
Overriding of methods and/or properties	Overriding methods in object-oriented programming allow subclasses to provide a specific implementation for a method defined in their superclass, tailoring functionality to suit their unique requirements. For instance, in the context of a hospital management system, the "SetPayment" method can be overridden in subclasses like "OutPatient" and "InPatient" to calculate payment amounts differently based on the type of patient. (Refer to Fig10 and Fig11)
Constructor overloading	Constructor overloading occurs when a class has multiple constructors, each accepting different parameters to create an instance of the object. In the visitor class, the default constructor includes parameters for name, phone number, visit date, and date of birth. However, if the visitor prefers not to provide their phone number and date of birth, they can use the overloaded constructor,

	which only requires name and visit date parameters, setting date of birth and phone number to "N/A." (Refer to Fig7)
Enumeration (Enum)	Enumerations were defined to represent different payment types (Visa, Cash, WHISH, and OMT) and different room types (either regular or first class). (Refer to Fig12 and Fig14)
Collections (e.g., Lists)	Examples of collections in a hospital management system include Lists of entrants, such as PatientsRecords, which stores information about patients and visitors entering the hospital. These collections provide a structured way to manage and manipulate data related to hospital entrants. (Refer to Fig3)
Loops (e.g., foreach)	The loop goes through a list of people entering the hospital, gathering details like patient ID and type, and then adds them together to create records. (Refer to Fig4)

SOLID Principle	**Description of where the principle is realized and for what purpose** (around 50 Words)
1) Single Responsibility	Every class throughout our system adheres to the single responsibility principle, meaning that each class is responsible for one specific task. As such, the HospitalRoomBuilder class is solely responsible for creating rooms, which in turn minimizes the need for code modifications.
2) Open-Closed	Implemented via extensible classes such as Patient class. Additional patient types like Inpatient and Outpatient are seamlessly incorporated without altering the existing code, enhancing maintainability and avoiding the need for code modifications.
3) Liskov Substitution	This principle is all about well-designed inheritance. Every class that uses generalization follows the Liskov substitution principle. For instance, the classes HospitalRoomBuilder, FirstClassHospitalRoom, and RegularHospitalRoom can replace and substitute each other without things going wrong. They all inherit methods that work across all subclasses, ensuring a well-designed inheritance structure.
4) Interface Segregation	Throughout the program, client classes interact only with the methods they require due to the establishment of client-specific interfaces like IEntrants, IVisitor, and IPatient. For instance, the IVisitor interface allows the Visitor class to access methods specified in the IVistor interface, same goes for the IPatient and Patient classes. This design minimizes the need for recompilation and redeployment of other clients when changes are made to methods they don't utilize.
5) Dependency Inversion	There are no direct connections between any two concrete classes in our system. In our system, the hospital isn't directly connected to the entrants who visit it; instead, they're linked through an abstraction called IEntrants. This setup reuduces the coupling between different components and enables the introduction of new implementations without affecting the existing code. Also, the Hospital class is linked to the abstract SystemConnector class, and the Payment class isn't directly linked to the Receipt class but to an interface IReceipt class

GOF Design Pattern	**Description of where the pattern is realized and for what purpose** (around 50 Words)
1) Singleton	Implemented within the Hospital class, this design ensures that only one instance of Hospital exists throughout the program, granting a universal access to it.
2) Builder	Implemented by the RoomBuilder, including both the RegularHospitalRoom and FirstClassHospitalRoom classes, this approach facilitates the construction of different rooms with unique configurations, separating the process of constructing them from how they are represented.
3) Adapter Pattern	In this example, we implemented the adapter pattern because we are working with a pre-existing hospital management system that requires data storage and retrieval for system entrants. However, the existing class "HospitalManagementSystem" is not compatible with our "Hospital" class. To bridge this gap, we introduced "SystemConnector" as the parent class of "HospitalManagementSystemConnector". The latter serves as an adapter, facilitating data operations between the two classes, "Hospital" (with its "ExportData" function and "DisplayPatientRecords" function) and "HospitalManagementSystem" (with its "SaveData" function and "GetData" function). These functions were previously incompatible.

User Interface (Windows Forms)

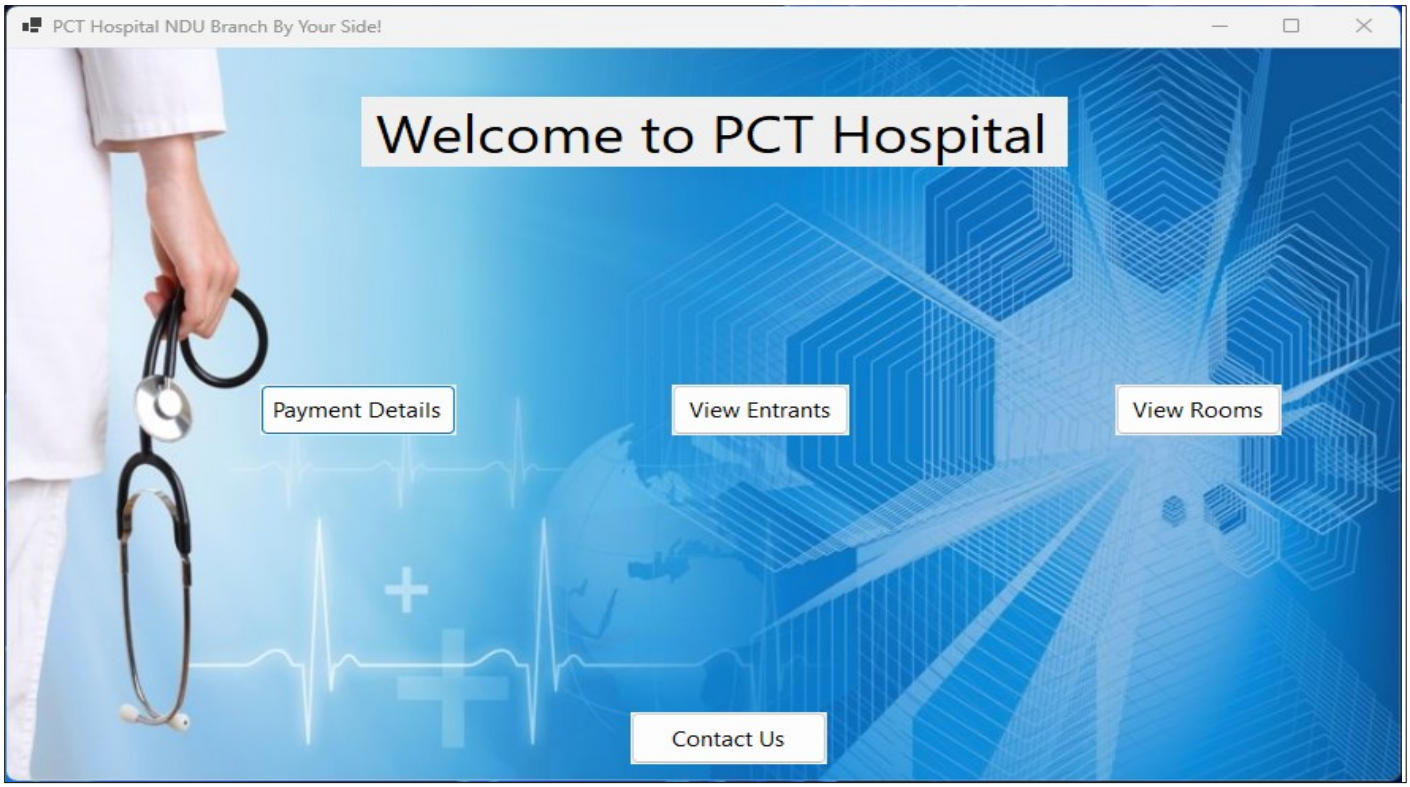

Figure 15. PCT Hospital System

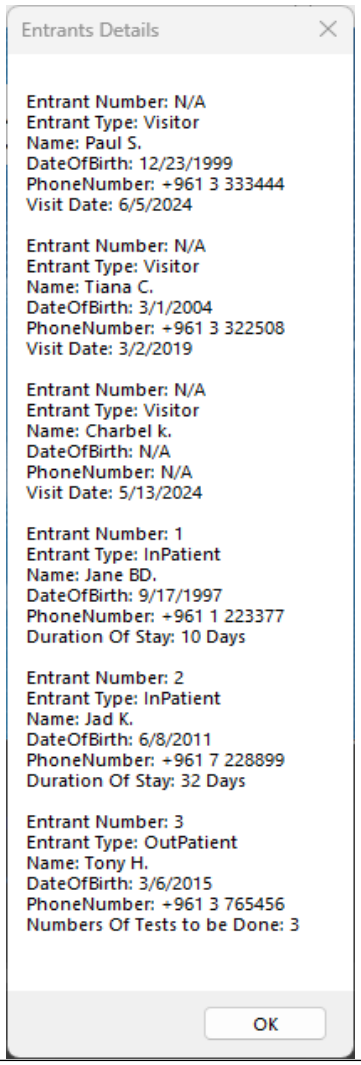

Figure 16. View Entrants Output

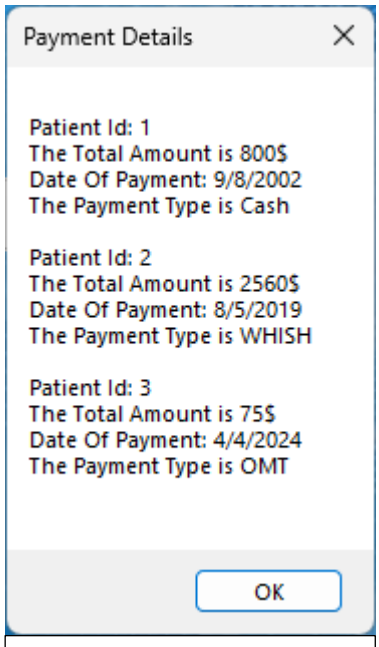

Figure 17. Payment Details Ouput

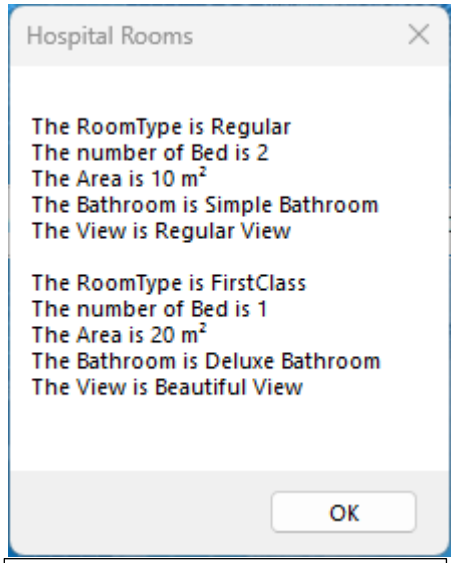

Figure 18. View Rooms Output

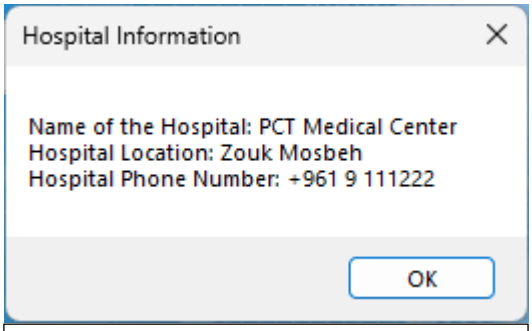

Figure 19. Contact Us Ouput

Brief Description of UI (around 50 words)
In our PCT Hospital UI, there's a section to view entrants, displaying information about both patients and visitors. For visitors, the entrant number isn't shown since it's not applicable. For patients, it indicates whether they're an inpatient or outpatient, along with all necessary information. Additionally, the UI presents payment details, differentiating between FirstClass and Regular rooms, and provides hospital information for contacting or locating the facility.