МІНІСТЕРСТВО  ОСВІТИ  І  НАУКИ  УКРАЇНИ

НАЦІОНАЛЬНИЙ   ТЕХНІЧНИЙ   УНІВЕРСИТЕТ   УКРАЇНИ

“КИЇВСЬКИЙ  ПОЛІТЕХНІЧНИЙ  ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 3

з дисципліни “Програмування”

тема “Структурні шаблони”

|  |  |  |
| --- | --- | --- |
| Виконав(ла)  студент(ка) II курсу  групи КП-01  Грабовська Анастасія Дмитрівна  (*прізвище, ім’я, по батькові*)  Варіант - 5 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Заболотня Тетяна Миколаївна  (*прізвище, ім’я, по батькові*) |

Київ 2021

**Мета роботи**

Ознайомитися зі структурними шаблонами у C# та реалізувати дві задачі за допомогою цих паттернів.

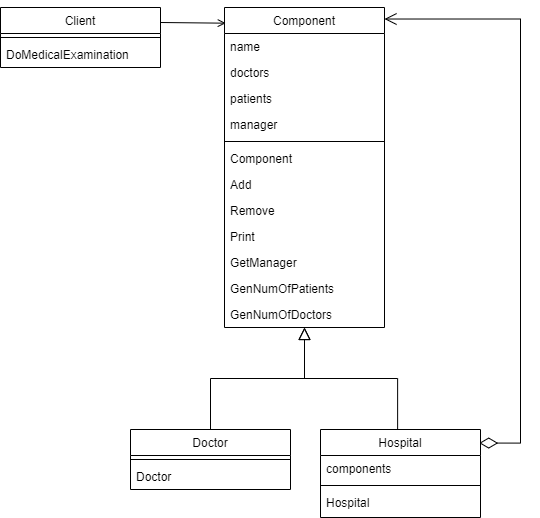
**Постановка задачі**

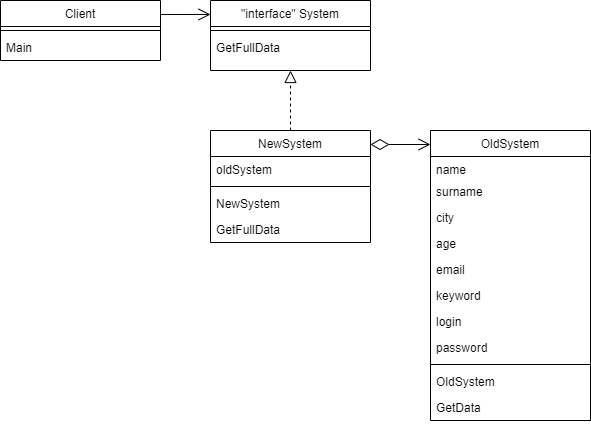
1. За допомогою шаблону проєктування створити структуру класів, до якої будуть входити класи, які відповідають таким об’єктам як Лікар, Відділення та Поліклініка. Кожний такий об’єкт має своє ім’я. Поліклініка та відділення характеризуються кількістю лікарів та хворих, а також мають власних завідувачів. У програмі передбачити реалізацію спеціального методу – «Пройти медичний огляд» у одного, декількох або всіх лікарів.
2. У старій системі реєстрації користувача необхідно було обов’язково вказувати ім’я, прізвище, місто проживання, вік, адресу електронної пошти, ключове слово, а також логін та пароль. Для прискорення процесу реєстрації необхідно вдосконалити систему: зробити таку надбудову над системою, щоб користувач повинен був вводити тільки ім’я, прізвище, логін, пароль та ключове слово. Всі інші незаповнені поля повинні отримати значення за замовчанням від системи.

**Обгрунтування вибору шаблону**

1. Очевидно, що для цієї структури класів потрібно реалізувати шаблон **компонувальник (Composite)**, оскільки ми маємо складну деревовидну структуру. Паттерн Компонувальник пропонує зберігати у складових об'єктах посилання на інші прості чи складні об'єкти. Ті, у свою чергу, теж можуть зберігати свої вкладені об'єкти тощо. У результаті можна будувати складну деревоподібну структуру даних, використовуючи лише два основні різновиди об'єктів. Завдяки тому, що прості та складові об'єкти реалізують спільний інтерфейс, клієнту байдуже, з яким саме об'єктом йому доведеться працювати.
2. Очевидно, що для цієї структури класів потрібно реалізувати шаблон **адаптер,** оскільки нам треба адаптувати старий інтерфейс до нового. Адаптер виступає прошарком між двома об'єктами, перетворюючи виклики одного на виклики зрозумілі іншому.

**UML діаграма класів**





**Код програми:**

1.

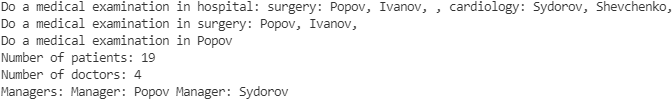
|  |
| --- |
| abstract class Component  {      protected int doctors;      protected int patients;      protected string name;      protected string manager;      public Component(string name)      {          this.name = name;      }      public virtual void Add(Component component){}      public virtual void Remove(Component component) { }      public virtual string Print()      {          return name;      }      public virtual string GetManager()      {          return manager;      }      public virtual int GetNumOfPatients()      {          return patients;      }      public virtual int GetNumOfDoctors()      {          return doctors;      }  }  class Hospital : Component  {      private List<Component> components = new List<Component>();      public Hospital(string name) : base(name)      {}      public override void Add(Component component)      {          components.Add(component);      }      public override void Remove(Component component)      {          components.Remove(component);      }      public override string Print()      {          string departments = name + ": ";          for(int i=0; i<components.Count;i++)          {              departments += components[i].Print() + ", ";          }          return departments;      }      public override int GetNumOfPatients()      {          for(int i=0; i<components.Count;i++)          {              patients += components[i].GetNumOfPatients();          }          return patients;      }      public override int GetNumOfDoctors()      {          for(int i=0; i<components.Count;i++)          {              doctors += components[i].GetNumOfDoctors();          }          return doctors;      }      public override string GetManager()      {          if (components != null)          {              if (components[components.Count-1].GetType().Name == "Doctor")              {                  return "Manager: " + components[0].Print();              }              manager = "Managers: ";              for(int i=0; i<components.Count;i++)              {                  manager += components[i].GetManager() + " ";              }              return manager;          }          return "";      }  }  class Doctor : Component  {      public Doctor(string name, int patients) : base(name)      {          this.doctors = 1;          this.patients = patients;      }  }  class Client  {      public void DoMedicalExamination(Component component)      {          Console.WriteLine("Do a medical examination in " + component.Print());      }  } |
| Client patient = new Client();  Component hospital = new Hospital("hospital");  Component surgery = new Department("surgery");  Component cardiology = new Department("cardiology");  hospital.Add(surgery);  hospital.Add(cardiology);  Component doc1 = new Doctor("Popov", 4);  Component doc2 = new Doctor("Ivanov", 8);  surgery.Add(doc1);  surgery.Add(doc2);  Component doc3 = new Doctor("Sydorov", 2);  Component doc4 = new Doctor("Shevchenko", 5);  cardiology.Add(doc3);  cardiology.Add(doc4);  patient.DoMedicalExamination(hospital);  patient.DoMedicalExamination(surgery);  patient.DoMedicalExamination(doc1);  Console.WriteLine("Number of patients: " + hospital.GetNumOfPatients());  Console.WriteLine("Number of doctors: " + hospital.GetNumOfDoctors());  Console.WriteLine(hospital.GetManager()); |

2.

|  |
| --- |
| interface System  {      void GetFullData();  }  class OldSystem  {      private string name;      private string surname;      private string city;      private int age;      private string email;      private string keyword;      private string login;      private string password;      public OldSystem(string name, string surname, string city, int age, string email, string keyword, string login, string password)      {          this.name = name;          this.surname = surname;          this.city = city;          this.age = age;          this.email = email;          this.keyword = keyword;          this.login = login;          this.password = password;      }      public void GetData()      {          Console.WriteLine($"User: {name} {surname} {age} years old ({email}). City: {city}. Login: {login}. Password: {password}. Keyword: {keyword}. ");      }  }  class NewSystem : System  {      private OldSystem oldSystem;      public NewSystem(string name, string surname, string keyword, string login, string password)      {          this.oldSystem = new OldSystem(name, surname, "Kyiv", 18, "mail@gmail.com", keyword, login, password);      }      public void GetFullData()      {          oldSystem.GetData();      }  } |
| System newSystem = new NewSystem("Anna", "Ivanova", "bluberry", "anna123", "1234");  newSystem.GetFullData(); |

**Приклади результатів:**

1. Console:



1. Console:



**Висновки:**

Під час виконання лабораторної роботи я ознайомилася зі структурними шаблонами у C# та реалізувала дві задачі за допомогою цих паттернів.