

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 1\_\_

з дисципліни “Об’єктно орієнтоване програмування ”

тема “**Реалізація структурних шаблонів проектування**”

|  |  |  |
| --- | --- | --- |
| Виконала  студентка II курсу  групи КП-82  Лахман Ксенія Вікторівна  (*прізвище, ім’я, по батькові*) |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Заболотня Тетяна Миколаївна  (*прізвище, ім’я, по батькові*) |

Київ 2020

**Мета роботи**

Ознайомлення з основними характеристиками шаблонів «Декоратор», «Компонувальник» та «Міст», запам’ятовування поширених ситуацій, коли використання цих шаблонів є доцільним, набуття вмінь та навичок реалізації шаблонів під час створення програмного коду.

**Постановка задачі:**

1. Реалізувати різноманітні методи проведення віртуального медичного огляду лікарями як мінімум трьох спеціалізацій (стоматолог, хірург, ортопед і т.ін.). Забезпечити підтримку віртуального медичного огляду певною множиною лікарів.

2. У банку вакансій кожна позиція характеризується назвою, компанією, що її пропонує, категорією вакансії та регіоном. Забезпечити виведення на екран інформації про вакансії будь-якого типу групування. Також реалізувати виведення тільки кількості вакансій з різними типами групування (за компанією, за категорією, за регіоном тощо).

**Обгрунтування вибору шаблонів:**

1. Декоратор

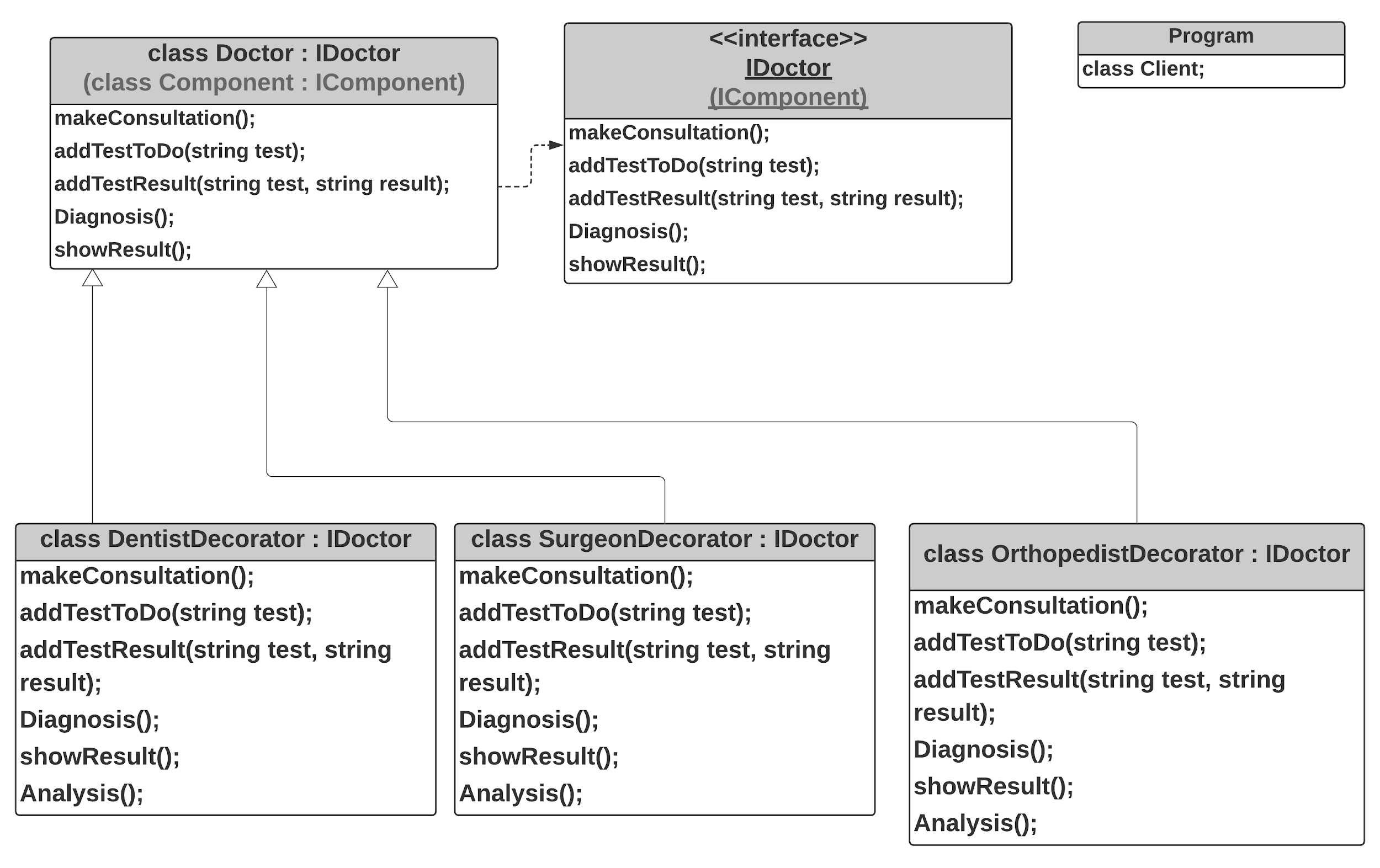
* Необхідно динамічно додавати нові об’єкти
* Конкретні функції належать конкретним класам, не перебуваючи на верхніх рівнях ієрархії. (Кожен лікар має свої функції для виконання)
* Велика кількість малих об’єктів зі схожою реалізацією (багато лікарів - багато схожих класів)

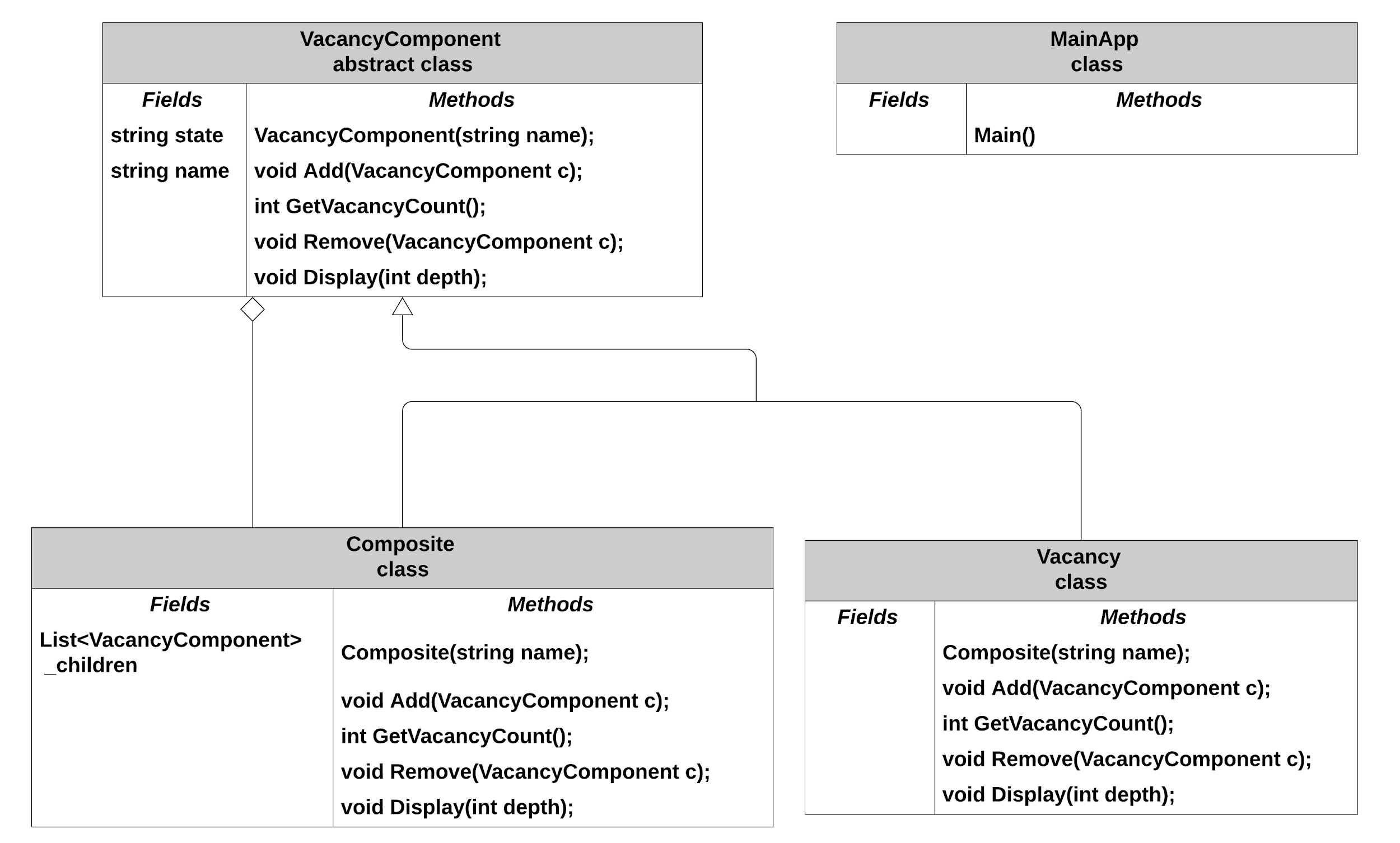
1. Компонувальник

* Добре підходить для деревовидної структури об’єктів
* Об’єкти не мають суттєвих відмінностей

**Діаграма класів**

**Задача1**

****

**Задача2**

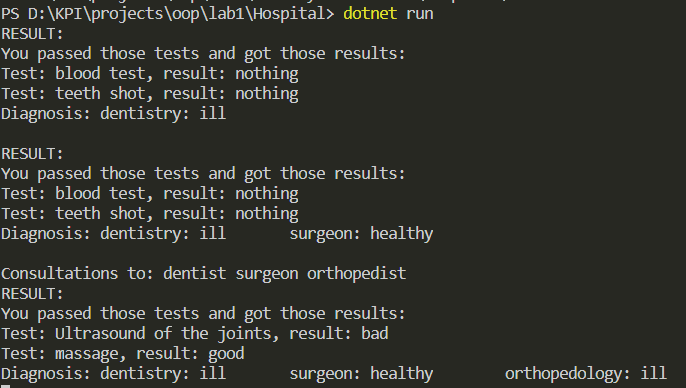
**Тексти коду програм**

|  |
| --- |
| **Задача 1** |
| class Hospital  {  interface IDoctor //IComponent  {  string makeConsultation();  bool addTestToDo(string test);  bool addTestResult(string test, string result);  string Diagnosis();  void showResult();  }  class Doctor : IDoctor // class Component : IComponent  {  public string makeConsultation() {  return "Consultations to: ";  }  public string Diagnosis()  {  return "Diagnosis: ";  }  public bool addTestToDo(string test)  {  return true;  }  public bool addTestResult(string test, string result)  {  return true;  }  public void showResult()  {  Console.WriteLine("RESULT:");  }  }  class DentistDecorator : IDoctor //class DecoratorA : IComponent  {  IDoctor doctor;  public Dictionary<string, string> tests = new Dictionary<string, string>();  public DentistDecorator(IDoctor d)  {  doctor = d;  }  public string makeConsultation()  {  string s = doctor.makeConsultation();  s += "dentist ";  return s;  }  public bool addTestToDo(string test)  {  tests.Add(test, "nothing");  return true;  }  public bool addTestResult(string test, string result)  {  tests[test] = result;  return true;  }  public string Diagnosis()  {  string s = doctor.Diagnosis();  s += "dentistry: ill\t";  return s;  }  public void showResult()  {  Console.WriteLine("RESULT:");  if (tests.Count != 0)  {  Console.WriteLine("You passed those tests and got those results: ");  foreach (KeyValuePair<string, string> kvp in tests)  {  Console.WriteLine("Test: " + kvp.Key + ", " + "result: " + kvp.Value);  }  Console.WriteLine(Diagnosis());  } else {  Console.WriteLine("You didn't pass the tests, so I don't know your diadnosis");  }  }  }  class SurgeonDecorator : IDoctor //class DecoratorB : IComponent  {  IDoctor doctor;  string state = "healthy";  public Dictionary<string, string> tests = new Dictionary<string, string>();  public SurgeonDecorator(IDoctor d)  {  doctor = d;  }  public string makeConsultation()  {  string s = doctor.makeConsultation();  s += "surgeon ";  return s;  }  public bool addTestToDo(string test)  {  tests.Add(test, "nothing");  return true;  }  public bool addTestResult(string test, string result)  {  tests[test] = result;  return true;  }  void Analysis()  {  if (tests.ContainsValue("very bad"))  {  state = "operation";  } else if (tests.ContainsValue("bad")) {  state = "ill";  }  }  public string Diagnosis()  {  Analysis();  string s = doctor.Diagnosis();  s += "surgeon: " + state + "\t";  return s;  }  public void showResult()  {  Console.WriteLine("RESULT:");  if (tests.Count != 0)  {  Console.WriteLine("You passed those tests and got those results: ");  foreach (KeyValuePair<string, string> kvp in tests)  {  Console.WriteLine("Test: " + kvp.Key + ", " + "result: " + kvp.Value);  }  Console.WriteLine(Diagnosis());  } else {  Console.WriteLine("You didn't pass the tests, so I don't know your diadnosis");  }  }  }  class OrthopedistDecorator : IDoctor //class DecoratorС : IComponent  {  IDoctor doctor;  string state = "healthy";  public Dictionary<string, string> tests = new Dictionary<string, string>();  public OrthopedistDecorator(IDoctor d)  {  doctor = d;  }  public string makeConsultation()  {  string s = doctor.makeConsultation();  s += "orthopedist ";  return s;  }  public bool addTestToDo(string test)  {  tests.Add(test, "nothing");  return true;  }  public bool addTestResult(string test, string result)  {  tests[test] = result;  return true;  }  void Analysis()  {  if (tests.ContainsValue("very bad"))  {  state = "operation";  } else if (tests.ContainsValue("bad")) {  state = "ill";  }  }  public string Diagnosis()  {  Analysis();  string s = doctor.Diagnosis();  s += "orthopedology: " + state + "\t";  return s;  }  public void showResult()  {  Console.WriteLine("RESULT:");  if (tests.Count != 0)  {  Console.WriteLine("You passed those tests and got those results: ");  foreach (KeyValuePair<string, string> kvp in tests)  {  Console.WriteLine("Test: " + kvp.Key + ", " + "result: " + kvp.Value);  }  Console.WriteLine(Diagnosis());  } else {  Console.WriteLine("You didn't pass the tests, so I don't know your diadnosis");  }  }  }  class Client  {  static void Main()  {  IDoctor doctor = new Doctor();  doctor = new DentistDecorator(doctor);  doctor.makeConsultation();  doctor.addTestToDo("blood test");  doctor.addTestToDo("teeth shot");  doctor.showResult();    doctor = new SurgeonDecorator(doctor);  Console.WriteLine("");  doctor.makeConsultation();  doctor.addTestToDo("blood test");  doctor.addTestToDo("teeth shot");  doctor.showResult();  doctor = new OrthopedistDecorator(doctor);  Console.WriteLine("");  Console.WriteLine(doctor.makeConsultation());  doctor.addTestToDo("Ultrasound of the joints");  doctor.addTestToDo("massage");  doctor.addTestResult("massage", "good");  doctor.addTestResult("Ultrasound of the joints", "bad");  doctor.showResult();  Console.ReadKey();  }  }  } |

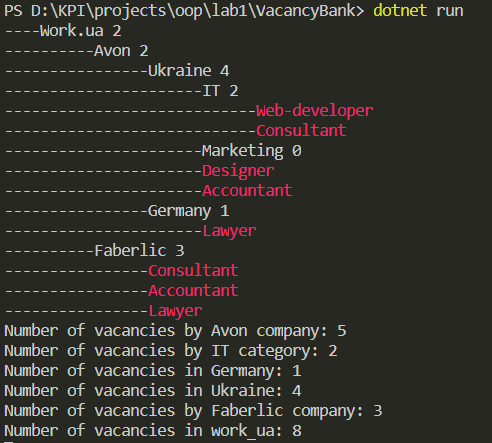
|  |
| --- |
| **Задача 2** |
| namespace VacancyBank  {  class MainApp  {  static void Main()  {  // company, region, category, name  Composite work\_ua = new Composite("Work.ua");  Composite company\_a = new Composite("Avon");  Composite company\_f = new Composite("Faberlic");  Composite region\_u = new Composite("Ukraine");  Composite region\_g = new Composite("Germany");  Composite category\_i = new Composite("IT");  Composite category\_m = new Composite("Marketing");  Vacancy name\_w = new Vacancy("Web-developer");  Vacancy name\_c = new Vacancy("Consultant");  Vacancy name\_l = new Vacancy("Lawyer");  Vacancy name\_a = new Vacancy("Accountant");  Vacancy name\_d = new Vacancy("Designer");  company\_a.Add(region\_u);  company\_a.Add(region\_g);  region\_u.Add(category\_i);  region\_u.Add(category\_m);  region\_u.Add(name\_d);  region\_u.Add(name\_a);  category\_i.Add(name\_w);  category\_i.Add(name\_c);  region\_g.Add(name\_l);  company\_f.Add(name\_c);  company\_f.Add(name\_a);  company\_f.Add(name\_l);    work\_ua.Add(company\_a);  work\_ua.Add(company\_f);  work\_ua.Display(4);  Console.WriteLine("Number of vacancies by Avon company: " + company\_a.GetVacancyCount());  Console.WriteLine("Number of vacancies by IT category: " + category\_i.GetVacancyCount());  Console.WriteLine("Number of vacancies in Germany: " + region\_g.GetVacancyCount());  Console.WriteLine("Number of vacancies in Ukraine: " + region\_u.GetVacancyCount());  Console.WriteLine("Number of vacancies by Faberlic company: " + company\_f.GetVacancyCount());  Console.WriteLine("Number of vacancies in work\_ua: " + work\_ua.GetVacancyCount());  Console.ReadKey();  }  }  abstract class VacancyComponent  {  public string name;  public int count;  public string state;  public VacancyComponent(string name)  {  this.name = name;  }  public abstract void Add(VacancyComponent c);  public abstract int GetVacancyCount();  public abstract void Remove(VacancyComponent c);  public abstract void Display(int depth);  }  class Composite : VacancyComponent  {  private List<VacancyComponent> \_children = new List<VacancyComponent>();  public Composite(string name) : base(name)  {  this.count = 0;  this.state = "composite";  }  public override void Add(VacancyComponent component)  {  \_children.Add(component);  this.count += 1;  }  public override void Remove(VacancyComponent component)  {  \_children.Remove(component);  this.count -= 1;  }  public override void Display(int depth)  {  Console.WriteLine(new String('-', depth) + name + " " + this.count.ToString());  foreach (VacancyComponent component in \_children)  {  component.Display(depth + 6);  }  }  public override int GetVacancyCount()  {  int num = 0;  foreach (VacancyComponent component in \_children)  {  num += component.GetVacancyCount();    }  return num;  }  }  class Vacancy : VacancyComponent  {    public Vacancy(string name) : base(name)  {  this.state = "vacancy";  this.count = 1;  }  public override void Add(VacancyComponent c)  {  Console.WriteLine("Impossible operation");  }  public override void Remove(VacancyComponent c)  {  Console.WriteLine("Impossible operation");  }  public override void Display(int depth)  {  Console.Write(new String('-', depth));  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine(name);  Console.ResetColor();  }  public override int GetVacancyCount() {  return 1;  }  }  } |
|  |

**Приклади роботи програми:**

**Задача1**



**Задача2**



**Висновки**

Виконавши дану лабораторну роботу, я ознайомилася з основними характеристиками шаблонів «Декоратор» та «Компонувальник», запам’ятала ситуації, коли використання цих шаблонів є доцільним.