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 №** 4

з дисципліни “Програмування”

тема “Породжуючі 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 з породжуючими шаблонами у C# та реалізувати дві задачі за допомогою цих паттернів.

**Постановка задачі**

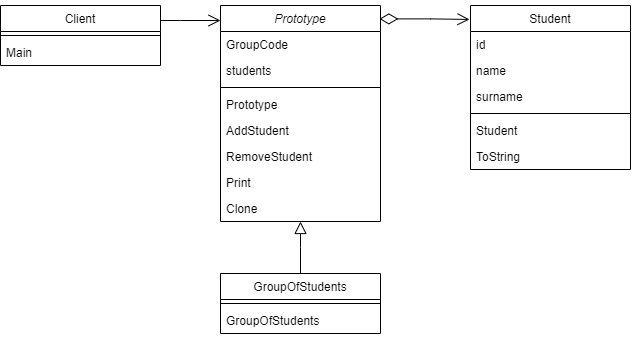
1. За допомогою шаблону проєктування розробити механізм клонування групи студентів, яка складається з окремих об’єктів класу «Студент». Дане клонування необхідне для обробки масиву даних про студентів різними службами університету: поліклінікою, деканатом, студентською радою і т.д.
2. За допомогою шаблону проєктування реалізувати процес випуску автомобілів різної комплектації. Так на різних етапах створення автівки на неї можна встановити двигуни різної потужності, різні типи кузова та по-різному оформити салон.

**Обгрунтування вибору шаблону**

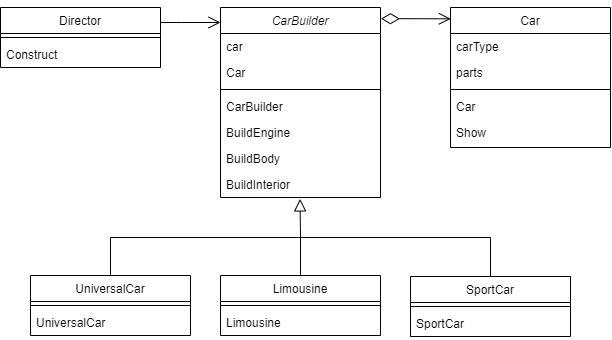
1. Очевидно, що для цієї структури класів потрібно реалізувати шаблон **прототип**, оскільки в завданні потрібно робити клонування. Паттерн прототип доручає створення копій самим об'єктам, що копіюються. Він вводить спільний інтерфейс для всіх об'єктів, які підтримують клонування. Це дозволяє копіювати об'єкти, не прив'язуючись до їх класів.
2. Очевидно, що для цієї структури класів потрібно реалізувати шаблон **будівельник,** оскільки нам треба реалізувати поступовий процес створення об’єкту. Будівельник дає можливість використовувати той самий код будівництва для створення різних уявлень об'єктів.

**UML діаграма класів**

3.



4.



**Код програми:**

3.

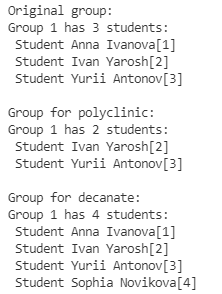
|  |
| --- |
| class Student  {      [JsonInclude]      public int id;      [JsonInclude]      public string name;      [JsonInclude]      public string surname;      public Student() {}      public Student(int id, string name, string surname)      {          this.id = id;          this.name = name;          this.surname = surname;      }      public override string ToString()      {          return $"Student {name} {surname}[{id}]";      }  }  abstract class Prototype  {      [JsonInclude]      public int GroupCode { get; private set; }      [JsonInclude]      public List<Student> students;      public Prototype() {}      public Prototype(int groupCode)      {          this.GroupCode = groupCode;          this.students = new List<Student>();      }      public abstract void AddStudent(Student student);      public abstract void RemoveStudent(Student student);      public abstract void Print();      public abstract Prototype Clone();  }  class GroupOfStudents : Prototype  {      public GroupOfStudents() {}      public GroupOfStudents(int groupCode) : base(groupCode)      {}      public override void AddStudent(Student student)      {          students.Add(student);      }      public override void RemoveStudent(Student student)      {          students.Remove(students[student.id - 1]);      }      public override void Print()      {          Console.WriteLine($"Group {GroupCode} has {students.Count} students:");          foreach(Student s in students)          {              Console.WriteLine(" " + s.ToString());          }      }      public override Prototype Clone()      {          var options = new JsonSerializerOptions          {              IncludeFields = true,          };          string jsonString = JsonSerializer.Serialize(this, options);          return JsonSerializer.Deserialize<GroupOfStudents>(jsonString, options);      }  } |
| GroupOfStudents groupOfStudents = new GroupOfStudents(1);  Student anna = new Student(1, "Anna", "Ivanova");  groupOfStudents.AddStudent(anna);  Student ivan = new Student(2, "Ivan", "Yarosh");  groupOfStudents.AddStudent(ivan);  Student yurii = new Student(3, "Yurii", "Antonov");  groupOfStudents.AddStudent(yurii);  GroupOfStudents groupForPolyclinic = (GroupOfStudents) groupOfStudents.Clone();  groupForPolyclinic.RemoveStudent(anna);  GroupOfStudents groupForDecanate = (GroupOfStudents) groupOfStudents.Clone();  groupForDecanate.AddStudent(new Student(4, "Sophia", "Novikova"));  Console.WriteLine("Original list equels to list for polyclinic " + (groupOfStudents==groupForPolyclinic));  Console.WriteLine("Original group:");  groupOfStudents.Print();  Console.WriteLine("\nGroup for polyclinic:");  groupForPolyclinic.Print();  Console.WriteLine("\nGroup for decanate:");  groupForDecanate.Print(); |

4.

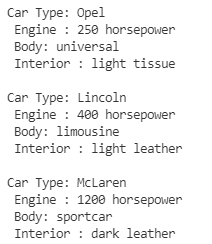
|  |
| --- |
| class Director  {      public void Construct(CarBuilder carBuilder)      {          carBuilder.BuildEngine();          carBuilder.BuildBody();          carBuilder.BuildInterior();      }  }  abstract class CarBuilder  {      protected Car car;      public Car Car      {          get { return car; }      }      public abstract void BuildEngine();      public abstract void BuildBody();      public abstract void BuildInterior();  }  class UniversalCar : CarBuilder  {      public UniversalCar()      {          car = new Car("Opel");      }      public override void BuildEngine()      {          car["engine"] = "250 horsepower";      }      public override void BuildBody()      {          car["body"] = "universal";      }      public override void BuildInterior()      {          car["interior"] = "light tissue";      }  }  class Limousine : CarBuilder  {      public Limousine()      {          car = new Car("Lincoln");      }      public override void BuildEngine()      {          car["engine"] = "400 horsepower";      }      public override void BuildBody()      {          car["body"] = "limousine";      }      public override void BuildInterior()      {          car["interior"] = "light leather";      }  }  class SportCar : CarBuilder  {      public SportCar()      {          car = new Car("McLaren");      }      public override void BuildEngine()      {          car["engine"] = "1200 horsepower";      }      public override void BuildBody()      {          car["body"] = "sportcar";      }      public override void BuildInterior()      {          car["interior"] = "dark leather";      }  }  class Car  {      private string carType;      private Dictionary<string,string> parts =      new Dictionary<string,string>();      public Car(string carType)      {          this.carType = carType;      }      public string this[string key]      {          get { return parts[key]; }          set { parts[key] = value; }      }      public void Show()      {          Console.WriteLine("\nCar Type: {0}", carType);          Console.WriteLine(" Engine : {0}", parts["engine"]);          Console.WriteLine(" Body: {0}", parts["body"]);          Console.WriteLine(" Interior : {0}", parts["interior"]);      }  } |
| Director director = new Director();  CarBuilder builder1 = new UniversalCar();  director.Construct(builder1);  builder1.Car.Show();  CarBuilder builder2 = new Limousine();  director.Construct(builder2);  builder2.Car.Show();  CarBuilder builder3 = new SportCar();  director.Construct(builder3);  builder3.Car.Show(); |

**Приклади результатів:**

1. Console:



1. Console::



**Висновки:**

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