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 “Шаблони проєктування”

Виконав студент
ІІ курсу групи КП-01
Тітов Єгор Павлович
Варіант 16

Перевірив
“____” “_____” 20___р
викладач
Заболотня Тетяна Миколаївна

Київ 2021

Постановка задач

1. Домашні тварини (кішки та собаки) характеризуються віком та іменем. Вони живуть у квартирах, які знаходяться у під'їздах, що розташовані у будинках. За допомогою шаблону проєктування забезпечити можливість виведення списку тварин по квартирам, під'їздам, будинкам з вказівкою імені та віку. Крім цього розробити метод визначення середнього віку кішок та собак, які живуть на заданій території.

2. Всі квіти мають таку характеристику як відношення до вологи. Якщо квітка любить вологу, її треба поливати більше звичайної норми на 30%, а якщо не любить вологу – на 50% менше за звичайну норму. Для автоматичного поливання трьох видів квітів раніше існувало 3 окремих прилади. Потім їх об'єднали в один та надали новому пристрою можливість розпізнавати тип квітки. За допомогою шаблону проєктування реалізувати програмний прототип нового приладу для поливання квітів.

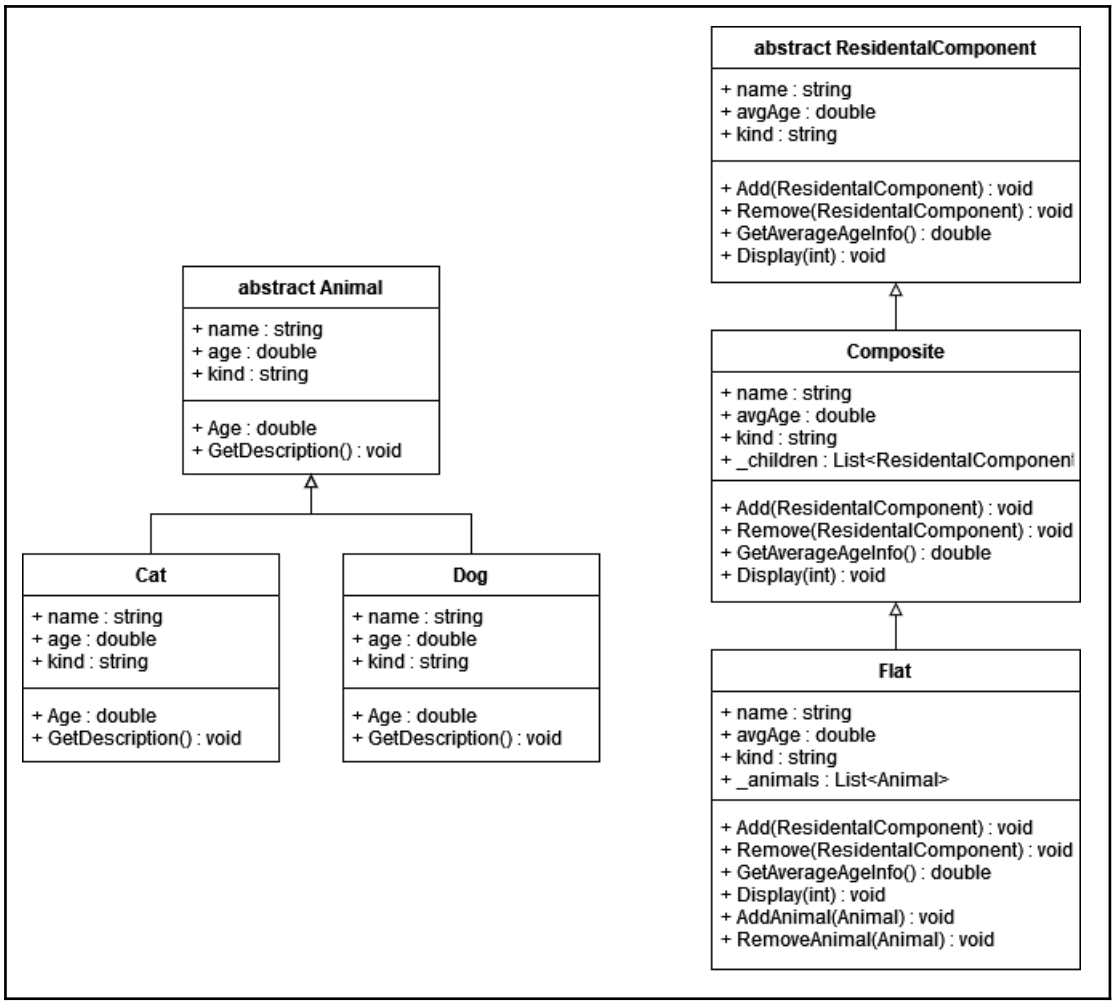
Обгрунтування вибору шаблонів

1. Оскільки маємо ієрархічну структуру "будівля - під'їзд - квартира" найкраще буде використати шаблон "Компонувальник".

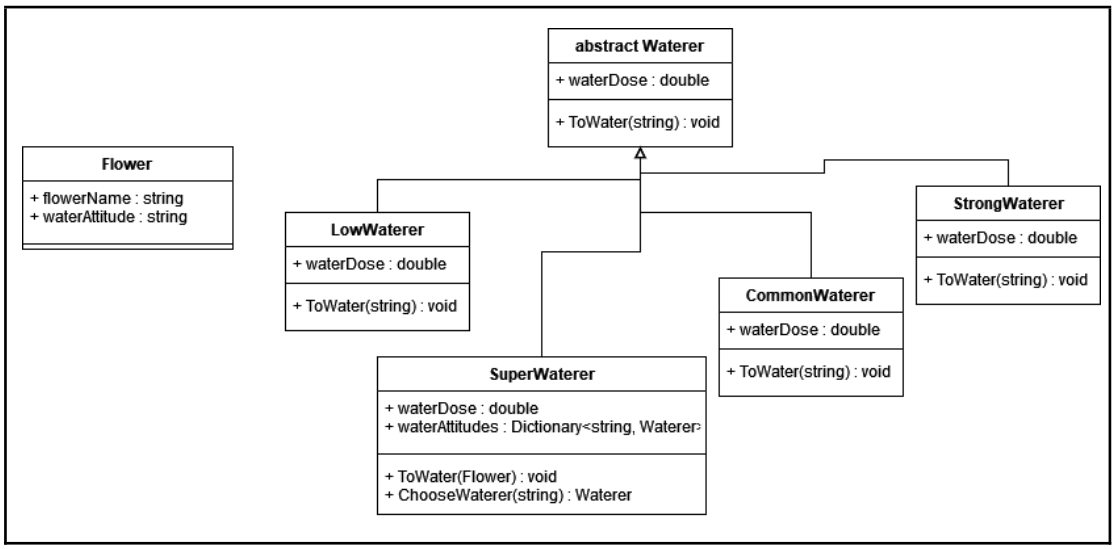
2. За умовою маємо три прилади, що об'єднали в один пристрій. Тобто маємо окремий функціонал для кожного приладу, що зручно буде об'єднати в одне ціле і приховати "зайве", використовуючи шаблон "Фасад".

UML діаграма класів

1. Завдання 1



2. Завдання 2



Текст програми

Завдання 1

```
using System;

namespace No_1
{
    abstract class Animal
    {
        protected string name;
        protected double age;
        protected string kind = "unknown kind";

        public Animal(string name, int age)
        {
            this.name = name;
            this.age = age;
        }

        public double Age
        {
            get
            {
                return age;
            }
            set
            {
                if (age > 0)
                {
                    this.age = value;
                }
                else
                {
                    Console.WriteLine("Incorrect age value entered");
                }
            }
        }

        public string GetDescription()
        {
```

```
            return $"{kind} '{name}'. Age: '{age}'";
        }
    }
}
```

```
namespace No_1
{
    class Cat : Animal
    {
        public Cat(string name, int age) : base(name, age)
        {
            this.kind = "Cat";
        }
    }
}
```

```
namespace No_1
{
    class Dog : Animal
    {
        public Dog(string name, int age) : base(name, age)
        {
            this.kind = "Dog";
        }
    }
}
```

```
namespace No_1
{
    abstract class ResidentalComponent
    {
        protected string name;
        protected double avgAge;

        public ResidentalComponent(string name)
        {
            this.name = name;
        }

        public abstract void Add(ResidentalComponent c);
        public abstract void Remove(ResidentalComponent c);
        public abstract double GetAverageAgeInfo();
```

```
        public abstract void Display(int depth);
    }
}
```

```
using System;
using System.Collections.Generic;

namespace No_1
{
    class Composite : ResidentalComponent
    {
        private List<ResidentalComponent> _children = new
List<ResidentalComponent>();

        public Composite(string name) : base(name)
        {
            this.avgAge = 0;
        }

        public override void Add(ResidentalComponent component)
        {
            _children.Add(component);
        }

        public override void Remove(ResidentalComponent component)
        {
            _children.Remove(component);
        }

        public override void Display(int depth)
        {
            Console.WriteLine(new String('-', depth) + " " + name + ".
Average animals age: " + this.avgAge.ToString());

            foreach (ResidentalComponent component in _children)
            {
                component.Display(depth + 2);
            }
        }

        public override double GetAverageAgeInfo()
```

```
        {
            this.avgAge = 0;
            double totalAge = 0;
            int totalCount = 0;

            foreach (ResidentalComponent component in _children)
            {
                double compAvgAge = component.GetAverageAgeInfo();

                if (compAvgAge != 0)
                {
                    totalAge += compAvgAge;

                    totalCount++;
                }
            }

            if (totalCount != 0)
            {
                this.avgAge = totalAge / totalCount;
            }

            return this.avgAge;
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace No_1
{
    class Flat : ResidentalComponent
    {
        private List<Animal> _animals = new List<Animal>();

        public Flat(string name) : base(name)
        {

        }
```

```
        public override void Add(ResidentalComponent c)
        {
            Console.WriteLine("Impossible operation");
        }
        public override void Remove(ResidentalComponent c)
        {
            Console.WriteLine("Impossible operation");
        }

        public void AddAnimal(Animal an)
        {
            _animals.Add(an);
        }
        public void RemoveAnimal(Animal an)
        {
            _animals.Remove(an);
        }


        public override void Display(int depth)
        {
            Console.WriteLine(new String('-', depth) + " " + name + ".
Average animals age: " + avgAge.ToString());
        }

        public override double GetAverageAgeInfo()
        {
            this.avgAge = 0;
            double totalAge = 0;
            int totalCount = 0;

            foreach (Animal animal in _animals)
            {
                double animalAge = animal.Age;

                if (animalAge != 0)
                {
                    totalAge += animalAge;

                    totalCount++;
                }
```

```
            }

            if (totalCount != 0)
            {
                this.avgAge = totalAge / totalCount;
            }

            return this.avgAge;
        }
    }
}
```

```
namespace No_1
{
    class Program
    {
        static void Main(string[] args)
        {
            Composite buildings = new Composite("All buildings");

            Composite build1 = new Composite("Building #1");
            Composite build2 = new Composite("Building #2");
            buildings.Add(build1);
            buildings.Add(build2);

            Composite entr1 = new Composite("Entrance #1");
            Composite entr2 = new Composite("Entrance #2");
            Composite entr3 = new Composite("Entrance #3");
            build1.Add(entr1);
            build1.Add(entr2);
            build2.Add(entr3);

            Flat fl1 = new Flat("Flat #1");
            Flat fl2 = new Flat("Flat #2");
            Flat fl3 = new Flat("Flat #3");
            Flat fl4 = new Flat("Flat #4");
            entr1.Add(fl1);
            entr2.Add(fl2);
            entr2.Add(fl3);
            entr3.Add(fl4);
```

```
            Dog dog1 = new Dog("Reks", 10);
            Dog dog2 = new Dog("Akbar", 5);
            Dog dog3 = new Dog("Sharik", 3);
            Dog dog4 = new Dog("Mops", 15);
            Cat cat1 = new Cat("Nami", 2);
            Cat cat2 = new Cat("Pirate", 6);
            Cat cat3 = new Cat("Murka", 5);
            Cat cat4 = new Cat("Abel", 10);

            fl1.AddAnimal(dog1);
            fl1.AddAnimal(dog2);
            fl1.AddAnimal(cat1);

            fl2.AddAnimal(dog3);
            fl2.AddAnimal(cat2);
            fl2.AddAnimal(cat3);

            fl3.AddAnimal(dog4);
            fl3.AddAnimal(cat4);

            buildings.GetAverageAgeInfo();
            buildings.Display(2);
        }
    }
}
```

2. Завдання 2

```
namespace No_2
{
    class Flower
    {
        public string flowerName;
        public string waterAttitude;

        public Flower(string name, string waterAttitude)
        {
            this.flowerName = name;
```

```
            this.waterAttitude = waterAttitude;
        }
    }
}
```

```
using System;

namespace No_2
{
    abstract class Waterer
    {
        protected double waterDose;

        public void ToWater(string flowerName)
        {
            Console.WriteLine($"Waterer pour flower '{flowerName}' with dose
'{waterDose}'");
        }
    }
}
```

```
namespace No_2
{
    class LowWaterer : Waterer
    {
        public LowWaterer(double commonDose)
        {
            this.waterDose = commonDose * 0.5;
        }
    }
}
```

```
namespace No_2
{
    class CommonWaterer : Waterer
    {
        public CommonWaterer(double commonDose)
        {
            this.waterDose = commonDose;
        }
    }
}
```

```
namespace No_2
{
    class StrongWaterer : Waterer
    {
        public StrongWaterer(double commonDose)
        {
            this.waterDose = commonDose + commonDose * 0.3;
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace No_2
{
    class SuperWaterer : Waterer
    {
        Dictionary<string, Waterer> waterAttitudes = new Dictionary<string,
Waterer>();

        public SuperWaterer(Waterer low, Waterer common, Waterer strong)
        {
            waterAttitudes.Add("negative", low);
            waterAttitudes.Add("common", common);
            waterAttitudes.Add("positive", strong);
        }

        public virtual void ToWater(Flower flower)
        {
            Waterer choosenWaterer = ChooseWaterer(flower.waterAttitude);

            if (choosenWaterer != null)
            {
                choosenWaterer.ToWater(flower.flowerName);
            }
            else
            {
                Console.WriteLine($"Program error. '{flower.waterAttitude}'
attitude to water does not exist");
            }
```

```
        }

        private Waterer ChooseWaterer(string waterAttitude)
        {
            Waterer choosenWaterer;

            waterAttitudes.TryGetValue(waterAttitude, out choosenWaterer);

            return choosenWaterer;
        }
    }
}
```

```
namespace No_2
{
    class Program
    {
        static void Main(string[] args)
        {
            double commonWaterDose = 1;

            Flower fl1 = new Flower("Rose", "positive");
            Flower fl2 = new Flower("Cactus", "negative");
            Flower fl3 = new Flower("Romashka", "common");

            Waterer low = new LowWaterer(commonWaterDose);
            Waterer common = new CommonWaterer(commonWaterDose);
            Waterer strong = new StrongWaterer(commonWaterDose);

            SuperWaterer superWaterer = new SuperWaterer(low, common,
strong);

            superWaterer.ToWater(fl1);
            superWaterer.ToWater(fl2);
            superWaterer.ToWater(fl3);
        }
    }
}
```

Приклади результатів

Завдання 1:

```
-- All buildings. Average animals age: 7,125
---- Building #1. Average animals age: 7,125
------ Entrance #1. Average animals age: 5,666666666666667
-------- Flat #1. Average animals age: 5,666666666666667
------ Entrance #2. Average animals age: 8,583333333333334
-------- Flat #2. Average animals age: 4,666666666666667
-------- Flat #3. Average animals age: 12,5
---- Building #2. Average animals age: 0
------ Entrance #3. Average animals age: 0
-------- Flat #4. Average animals age: 0
```

Завдання 2:

```
Waterer pour flower 'Rose' with dose '1,3'
Waterer pour flower 'Cactus' with dose '0,5'
Waterer pour flower 'Romashka' with dose '1'
```

Висновки

В результаті виконання лабораторної роботи я ознайомився і використав для вирішення поставлених завдань такі шаблони як “Фасад” і “Компонувальник”. Внаслідок цього я зрозумів переваги використання шаблонів проєктування і як важливо правильно їх застосовувати.