Практическая работа

По Проектному практикуму

Выполнил студент

Группы К17.1

Шишелов Владимир

Введение

Шаблон либо паттерн проектирования — это общепринятое решение к обыденной, повседневной, часто возникающей задаче. Таких шаблонов довольно много для решения многих возникающих задач.

При помощи паттернов возможно довольно ощутимое удешевление проекта, за счет уменьшения времени разработки и применения готовых решений.

Также использование шаблонов может произвести негативное влияние на продукт. Например, привести к сложности программы в целом.

В данной практической работе будут описаны пять известных шаблонов проектирования, их характеристики и способы применения. А также круговая диаграмма эффективности использования шаблонов и её описание на основе сторонних программных продуктов.

Примеры и листинг представлен на языке С#.

Одиночка (Singleton, Синглтон) – позволяет создать только один пример класса и обеспечение к нему доступа. Одиночка - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

Когда надо использовать Синглтон? Когда необходимо, чтобы для класса существовал только один экземпляр. Пример логики таков - создать объект если он не был создан ранее иначе использовать существующий.

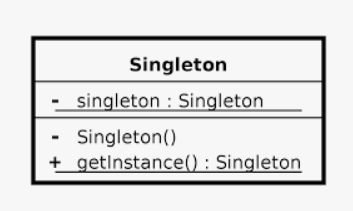


Рисунок 1 – иллюстрация класса Синглтон.

Пример на языке С#:

class Singleton

{

    private static Singleton instance;

    private Singleton()

    {}

    public static Singleton getInstance()

    {

        if (instance == null)

            instance = new Singleton();

        return instance;

    }

}

Приспособленец (Flyweight) – структурный шаблон проектирования, который позволяет использовать разделяемые объекты сразу в нескольких контекстах. Данный паттерн используется преимущественно для оптимизации работы с памятью.

Нужен для увеличения производительности программного продукта за счёт дублирования, копирования похожих, а иногда и одних и тех же по структуре объектов.

Простым примером является применение многочисленные объекты в играх например деревья в лесах ,дома в городах и планеты в космических билдерах. Оптимизация ресурсов здесь происходит за счет создания одной единой модели с общими параметрами каждого многочисленного объекта.

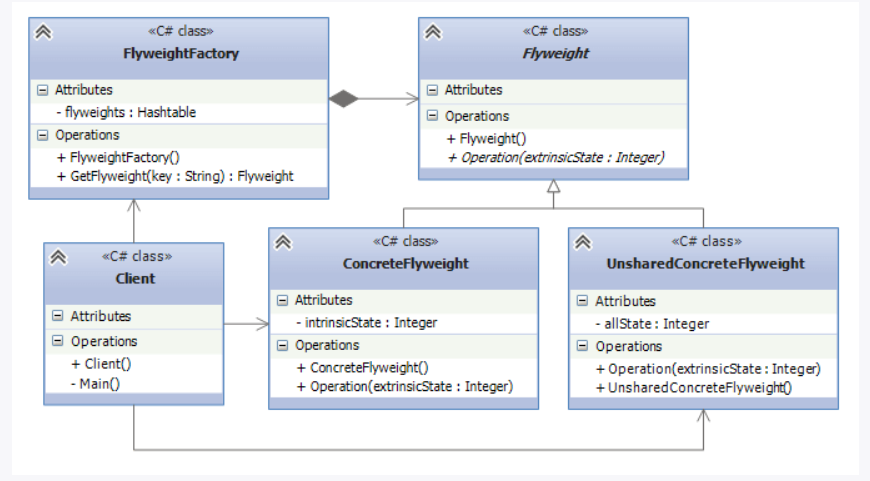


Рисунок 2 – Работа с шаблоном Flyweight

Пример Flyweight на языке С#:

class FlyweightFactory

{ Hashtable flyweights = new Hashtable();

    public FlyweightFactory()

    {

        flyweights.Add("X", new ConcreteFlyweight());

        flyweights.Add("Y", new ConcreteFlyweight());

        flyweights.Add("Z", new ConcreteFlyweight());

    }

    public Flyweight GetFlyweight(string key)

    {  if (!flyweights.ContainsKey(key))

            flyweights.Add(key, new ConcreteFlyweight());

        return flyweights[key] as Flyweight;

    }

}

abstract class Flyweight

{

    public abstract void Operation(int extrinsicState);

}

class ConcreteFlyweight : Flyweight

{

    int intrinsicState;

    public override void Operation(int extrinsicState){  }

}

class UnsharedConcreteFlyweight : Flyweight

{

    int allState;

    public override void Operation(int extrinsicState)

    {

        allState = extrinsicState;

    }

}

class Client

{

    void Main()

    {

        int extrinsicstate = 22;

        FlyweightFactory f = new FlyweightFactory();

        Flyweight fx = f.GetFlyweight("X");

        fx.Operation(--extrinsicstate);

        Flyweight fy = f.GetFlyweight("Y");

        fy.Operation(--extrinsicstate);

        Flyweight fd = f.GetFlyweight("D");

        fd.Operation(--extrinsicstate);

        UnsharedConcreteFlyweight uf = new UnsharedConcreteFlyweight();

        uf.Operation(--extrinsicstate);

    }

}

Прототип (Prototype) – шаблон основаный на идее полиморфизма родственных объектов. Позволяет создавать объекты на основе уже ранее созданных объектов-прототипов. То есть по сути данный паттерн предлагает технику клонирования объектов.

Использование Прототипа:

* Когда конкретный тип создаваемого объекта должен определяться динамически во время выполнения
* Когда нежелательно создание отдельной иерархии классов фабрик для создания объектов-продуктов из параллельной иерархии классов (как это делается, например, при использовании паттерна Абстрактная фабрика)
* Когда клонирование объекта является более предпочтительным вариантом нежели его создание и инициализация с помощью конструктора. Особенно когда известно, что объект может принимать небольшое ограниченное число возможных состояний.

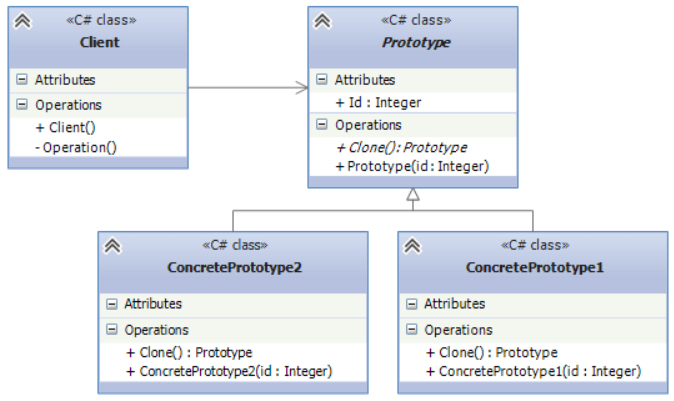


Рисунок 3 – Описание шаблона Prototype.

Пример Prototype на языке С#:

class Client

{

    void Operation()

    {

        Prototype prototype = new ConcretePrototype1(1);

        Prototype clone = prototype.Clone();

        prototype = new ConcretePrototype2(2);

        clone = prototype.Clone();

    }

}

abstract class Prototype

{

    public int Id { get; private set; }

    public Prototype(int id)

    {

        this.Id = id;

    }

    public abstract Prototype Clone();

}

class ConcretePrototype1 : Prototype

{

    public ConcretePrototype1(int id)

        : base(id)

    { }

    public override Prototype Clone()

    {

        return new ConcretePrototype1(Id);

    }

}

class ConcretePrototype2 : Prototype

{

    public ConcretePrototype2(int id)

        : base(id)

    { }

    public override Prototype Clone()

    {

        return new ConcretePrototype2(Id);

    }

}

Состояние (State) - шаблон проектирования, который позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. Похож с шаблоном Srtategy но имеет весомые отличия. Применяется когда поведение объекта должно зависеть от его состояния и может изменяться динамически во время выполнения и когда в коде методов объекта используются многочисленные условные конструкции, выбор которых зависит от текущего состояния объекта.

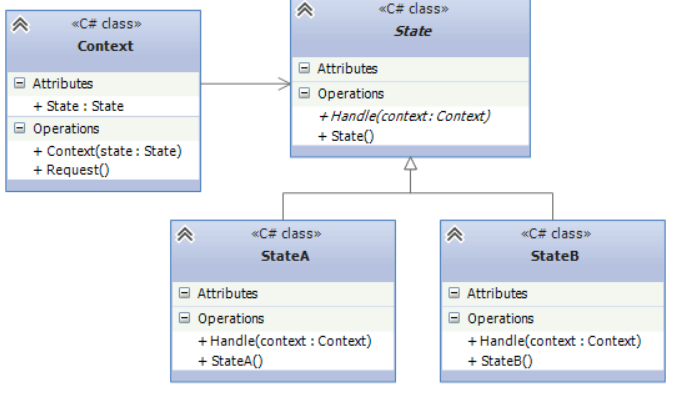


Рисунок 4 – Описание шаблона State.

Пример State на языке С#:

abstract class State

{

    public abstract void Handle(Context context);

}

class StateA : State

{

    public override void Handle(Context context)

    {

        context.State = new StateB();

    }

}

class StateB : State

{

    public override void Handle(Context context)

    {

        context.State = new StateA();

    }

}

class Context

{   public State State { get; set; }

    public Context(State state)

    {

        this.State = state;

    }

    public void Request()

    { this.State.Handle(this); }

}

Стратегия (Strategy) - представляет шаблон проектирования, который определяет набор алгоритмов, инкапсулирует каждый из них и обеспечивает их взаимозаменяемость. В зависимости от ситуации мы можем легко заменить один используемый алгоритм другим. При этом замена алгоритма происходит независимо от объекта, который использует данный алгоритм как и в шаблоне State.

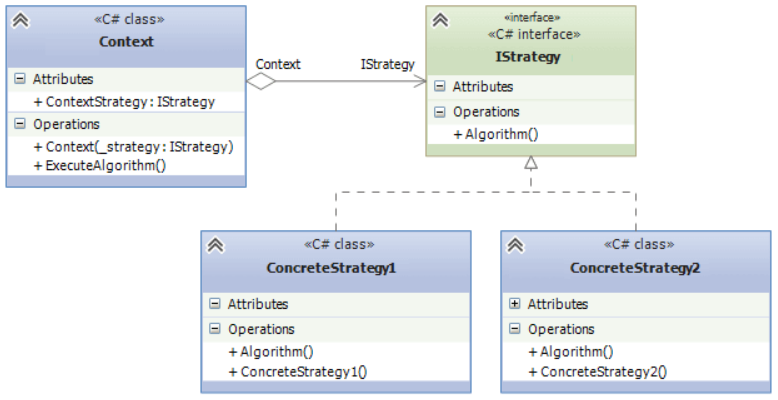


Рисунок 5 – Шаблон Strategy.

Пример Strategy на языке С#:

public interface IStrategy

{

    void Algorithm();

}

public class ConcreteStrategy1 : IStrategy

{

    public void Algorithm()

    {}

}

public class ConcreteStrategy2 : IStrategy

{

    public void Algorithm()

    {}

}

public class Context

{

    public IStrategy ContextStrategy { get; set; }

    public Context(IStrategy \_strategy)

    {

        ContextStrategy = \_strategy;

    }

    public void ExecuteAlgorithm()

    {

        ContextStrategy.Algorithm();

    }

}

Круговая диаграмма эффективности использования шаблонов.

После анализа известных программных продуктов в разных сферах применения можно выделить то, что паттерны используемые при разработке ПО зависят от сферы применения продукта. Таким образом в сфере GameDev большую нишу занимают паттерны для оптимизации, быстродействия и логики поведения игры такие как FlyWeight, Chain, Singleton, Prototype, Decorator, Iterator, State, Strategy, Builder.

Теперь рассмотрим это в процентном соотношении на круговой диаграмме.

Вывод к проделанной работе.

Шаблоны играют большую роль в проектировании программного продукта. Шаблоны могут как упростить и удешевить проект, так и сделать его невозможным для реализации. На данный момент шаблонов достаточно для решения большинства проблем и задач встречающихся при проектировании программного продукта.