Паттерны проектирования: Шаблонный метод (Template Method)

МКД 03.01 «Технология разработки программного обеспечения»

Определение

Шаблонный метод (**Template method**) — это поведенческий шаблон проектирования, который определяет алгоритм, некоторые методы которого делегируются подклассам, позволяя тем самым переопределить некоторые шаги алгоритма не меняя его структуры.

Назначение паттерна Шаблонный метод

- Паттерн Template Method определяет основу алгоритма и позволяет подклассам изменить некоторые шаги этого алгоритма без изменения его общей структуры.
- Базовый класс определяет шаги алгоритма с помощью абстрактных операций, а производные классы их реализуют.

Решаемая проблема

Имеются два разных, но в тоже время очень похожих компонента. Вы хотите внести изменения в оба компонента, избежав дублирования кода.

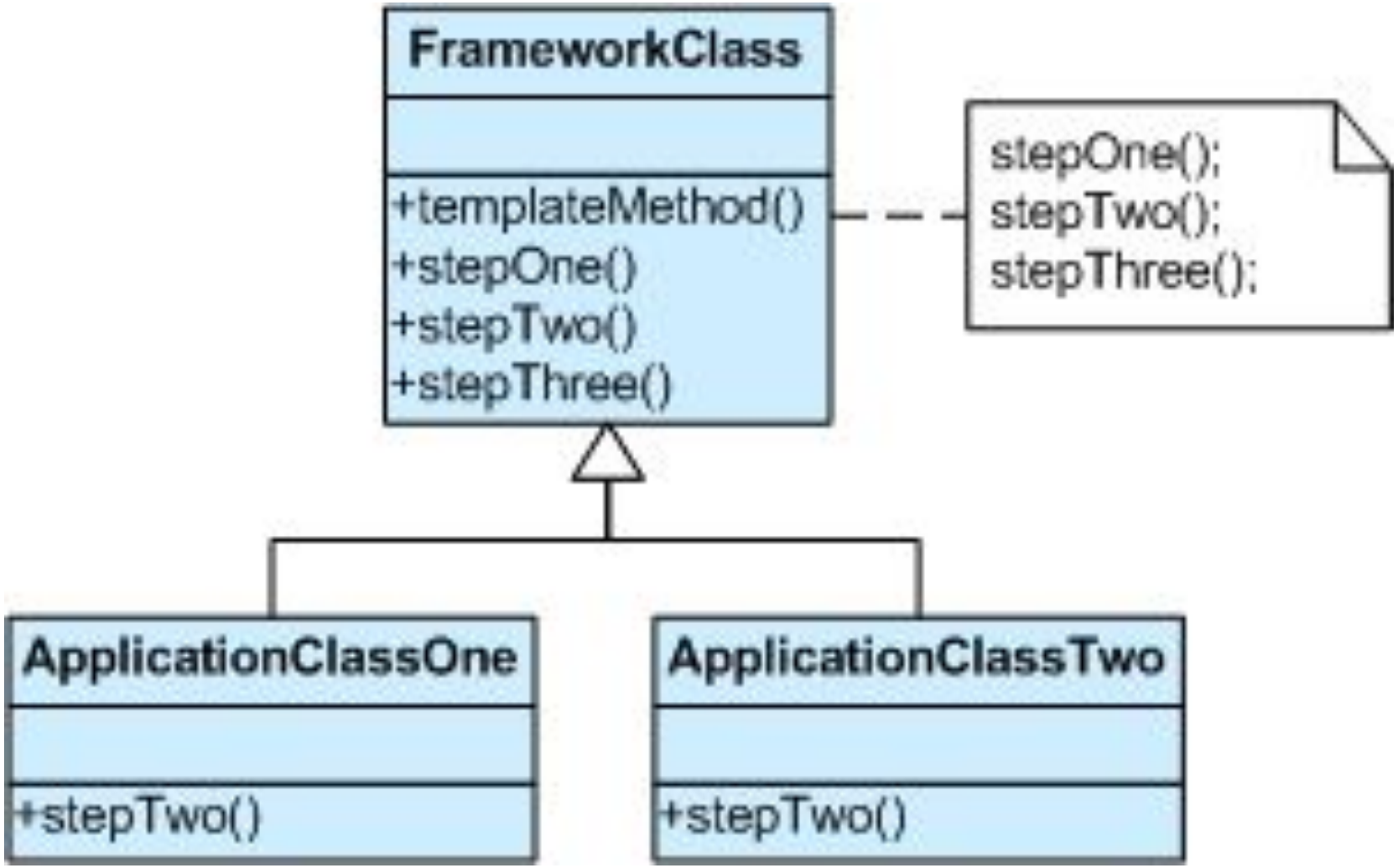
Обсуждение паттерна

Проектировщик компонента решает, какие шаги алгоритма являются неизменными (или стандартными), а какие изменяемыми (или настраиваемыми). Абстрактный базовый класс реализует стандартные шаги алгоритма и может предоставлять (или нет) реализацию по умолчанию для настраиваемых шагов. Изменяемые шаги могут (или должны) предоставляться клиентом компонента в конкретных производных классах.

Проектировщик компонента определяет необходимые шаги алгоритма, порядок их выполнения, но позволяет клиентам компонента расширять или замещать некоторые из этих шагов.

Паттерн Template Method широко применяется в каркасах приложений (frameworks). Каждый каркас реализует неизменные части архитектуры в предметной области, а также определяет те части, которые могут или должны настраиваться клиентом. Таким образом, каркас приложения становится "центром вселенной", а настройки клиента являются просто "третьей планетой от Солнца". Эту инвертированную структуру кода ласково называют принципом Голливуда - "Не звоните нам, мы сами вам позвоним".

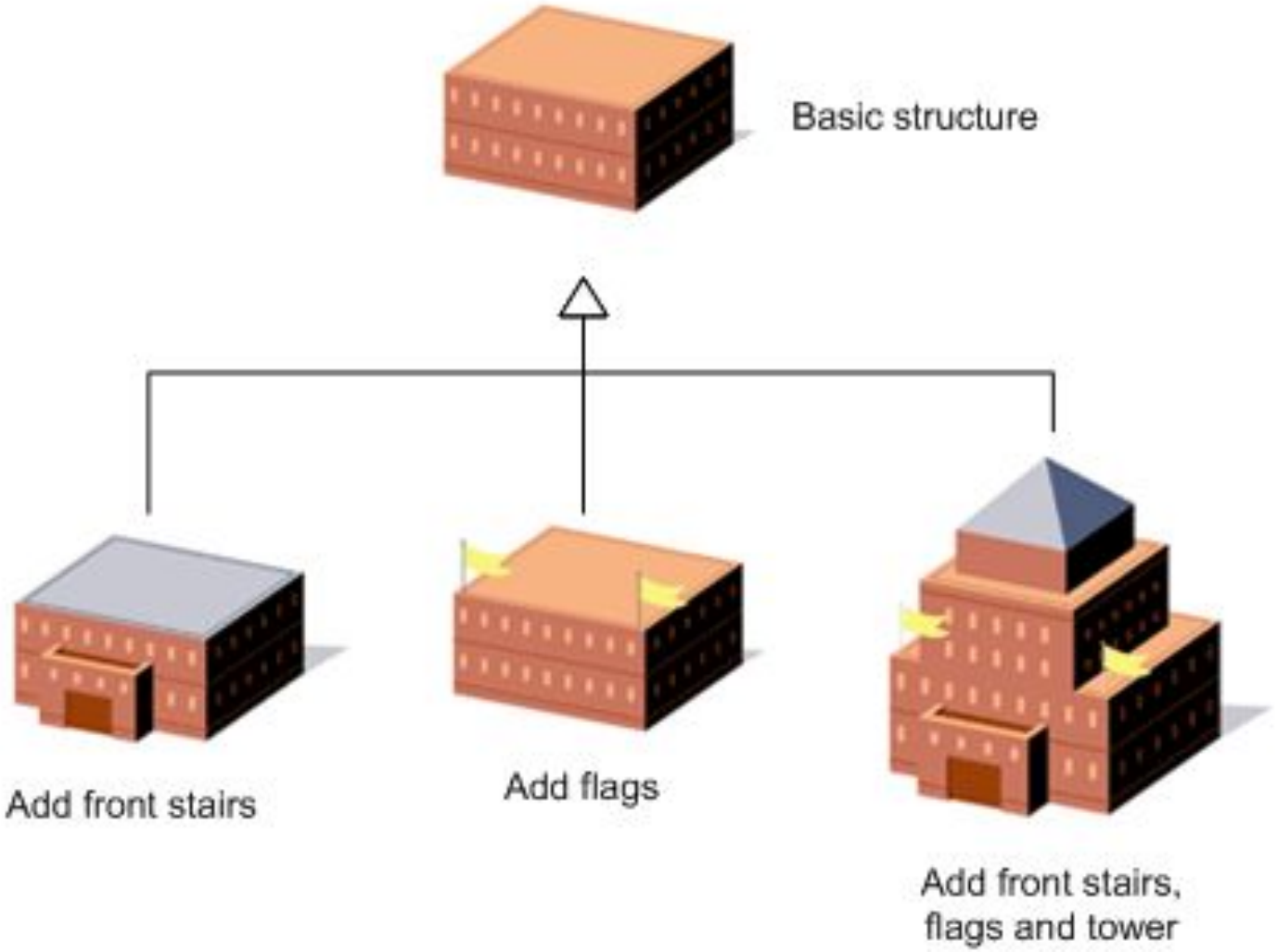
UML-диаграмма паттерна



Пример паттерна

Паттерн Template Method определяет основу алгоритма и позволяет подклассам изменить некоторые шаги этого алгоритма без изменения его общей структуры. Строители зданий используют шаблонный метод при проектировании новых домов. Здесь могут использоваться уже существующие типовые планы, в которых модифицируются только отдельные части.

Пример паттрена Template Method



Использование паттерна

- Исследуйте алгоритм и решите, какие шаги являются стандартными, а какие должны определяться подклассами.
- Создайте новый абстрактный базовый класс, в котором будет реализован принцип "не звоните нам, мы сами вам позвоним".
- Поместите в новый класс основу алгоритма (шаблонный метод) и определения стандартных шагов.
- Для каждого шага, требующего различные реализации, определите "замещающий" виртуальный метод. Этот метод может иметь реализацию по умолчанию или быть чисто виртуальным.
- Вызовите "замещающий" метод из шаблонного метода.
- Создайте подклассы от нового абстрактного базового класса и реализуйте в них "замещающие" методы.

Приготовление двух блюд

```
namespace TemplateMethod
{
    ссылок 0
    class Program
    {
        ссылок 0
        static void Main(string[] args)
        {
            HotDog hotDog = new HotDog();
            Hamburger hamburger = new Hamburger();
            Console.WriteLine("\nHotDog:");
            hotDog.Prepare();
            Console.WriteLine("\nHamburger:");
            hamburger.Prepare();
            Console.ReadKey();
        }
    }
}
```

Класс Hamburger

```
namespace TemplateMethod
{
    ссылок 2
    class Hamburger
    {
        ссылка 1
        public void Prepare()
        {
            RoastBread(); // Поджарка хлеба
            FryMeat(); // Поджарка мяса
            PutVegatables(); // Положить овощи
            AddKetchup(); // Добавить кетчуп
        }
        ссылка 1
        private void AddKetchup()  {
            Console.WriteLine("Ketchup");
        }
        ссылка 1
        private void PutVegatables()   {
            Console.WriteLine("Vegetables");
        }
        ссылка 1
        private void FryMeat()  {
            Console.WriteLine("Meat");
        }
        ссылка 1
        private void RoastBread()  {
            Console.WriteLine("Bread");
        }
    }
}
```

Класс HotDog

```
namespace TemplateMethod
{
    ссылок 2
    class HotDog
    {
        ссылка 1
        public void Prepare()
        {
            RoastBread(); // Поджарка хлеба
            BoilSausage(); // Сварить сосиску
            PutVegatables(); // Положить овощи
            AddMustard(); // Добавить горчицу
        }
        ссылка 1
        private void AddMustard()   {
            Console.WriteLine("Mustard");
        }

        ссылка 1
        private void PutVegatables()  {
            Console.WriteLine("Vegetables");
        }
        ссылка 1
        private void BoilSausage()  {
            Console.WriteLine("Sausage");
        }
        ссылка 1
        private void RoastBread()  {
            Console.WriteLine("Bread");
        }
    }
}
```

Класс FastFood

```
ссылок 0
class FastFood
{
    ссылок 0
    public void Prepare()
    {

    }
    ссылок 0
    private void PutVegatables()
    {
        Console.WriteLine("Vegetables");
    }
    ссылок 0
    private void RoastBread()
    {
        Console.WriteLine("Bread");
    }
}
```

Абстрактный класс FastFood

```
ссылок 0
abstract class FastFood
{
    ссылок 0
    public abstract void Prepare();

    ссылок 0
    public void PutVegatables()
    {
        Console.WriteLine("Vegetables");
    }
    ссылок 0
    public void RoastBread()
    {
        Console.WriteLine("Bread");
    }
}
```

Изменения в класса-наследниках

```
class HotDog : FastFood
{
    ссылка 1
    public override void Prepare()
    {
        RoastBread(); // Поджарка хлеба
        BoilSausage(); // Сварить сосиску
        PutVegatables(); // Положить овощи
        AddMustard(); // Добавить горчицу
    }

    ссылка 1
    private void AddMustard()
    {
        Console.WriteLine("Mustard");
    }

    ссылка 1
    private void BoilSausage()
    {
        Console.WriteLine("Sausage");
    }
}
```

Изменения в абстрактном классе FastFood

```
abstract class FastFood
{
    ссылок 0
    public void Prepare()
    {
        RoastBread(); // Поджарка хлеба
        PrepareMainIngredient(); // Поджарка мяса
        PutVegatables(); // Положить овощи
        AddTopings(); // Добавить кетчуп
    }

    ссылок 2
    public void PutVegatables()
    {
        Console.WriteLine("Vegetables");
    }
    ссылок 2
    public void RoastBread()
    {
        Console.WriteLine("Bread");
    }
    ссылка 1
    public abstract void AddTopings();

    ссылка 1
    public abstract void PrepareMainIngredient();
}
```

Итоговый класс-наследник Hamburger

```
class Hamburger : FastFood
{
    ссылок 2
    public override void AddTopings()
    {
        Console.WriteLine("Ketchup");
    }

    ссылок 2
    public override void PrepareMainIngredient()
    {
        Console.WriteLine("Meat");
    }
}
```

Итоговый класс-наследник HotDog

```
class HotDog : FastFood
{
    ссылок 3
    public override void AddTopings()
    {
        Console.WriteLine("Mustard");
    }
    ссылок 3
    public override void PrepareMainIngredient()
    {
        Console.WriteLine("Sausage");
    }
}
```

Класс FastFood: новый метод для реализации выбора

```
ссылок 3
abstract class FastFood
{
    ссылок 2
    public void Prepare()
    {
        RoastBread(); // Поджарка хлеба
        PrepareMainIngredient(); // Поджарка мяса
        PutVegatables(); // Положить овощи
        if(CustomerWantsTopings())
            AddTopings(); // Добавить кетчуп
    }
    ссылка 1
    public virtual bool CustomerWantsTopings()
    {
        return true;
    }
    ссылок 2
```

Изменение в классе-наследнике: возможность выбора

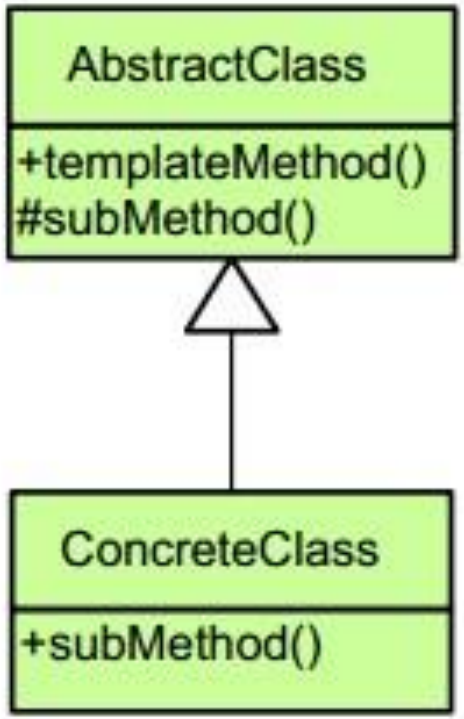
```
class HotDog : FastFood
{
    ссылок 2
    public override bool CustomerWantsTopings()
    {
        Console.Write("Do you want mustard?: (y/n)");
        var userInput = Console.ReadLine();
        return userInput.ToLower() == "y" || userInput.ToLower() == "yes";
    }
    ссылок 3
    public override void AddTopings()
    {
        Console.WriteLine("Mustard");
    }
    ссылок 3
    public override void PrepareMainIngredient()
    {
        Console.WriteLine("Sausage");
    }
}
```

Определение

Шаблонный Метод определяет основу алгоритма и позволяет подклассам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.

Шаблонный метод – это каркас, в который наследники могут подставить свои реализации

Диаграмма классов «Шаблонный метод»



Спасибо за внимание!

Паттерны проектирования: Шаблонный Метод (Template Method)