**ПАТТЕРН DECORATOR (ДЕКОРАТОР)**

Декоратор — паттерн, структурирующий объекты.

**Назначение**

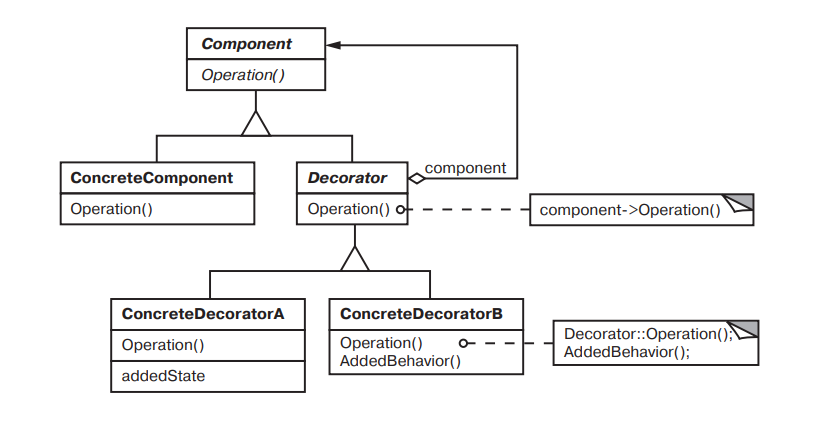
Динамически добавляет объекту новые обязанности. Является гибкой альтернативой порождению подклассов с целью расширения функциональности

Позволяет добавлять дополнительную функциональность, которая будет выполняться до, после или вместо основной функции объекта.

**Применимость**

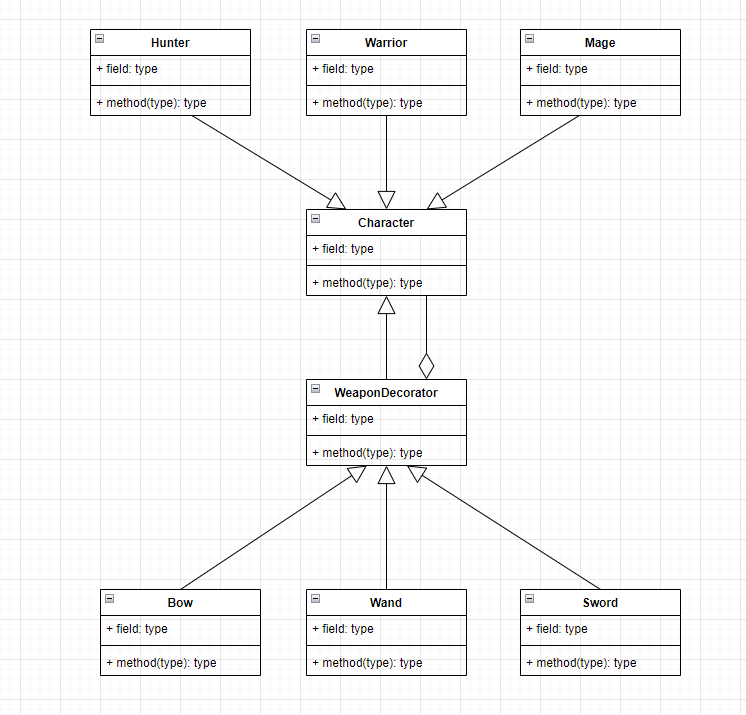
Основные условия для применения паттерна декоратор:

* динамическое, прозрачное для клиентов добавление обязанностей объектам (не затрагивающее другие объекты);
* реализация обязанностей, которые могут быть сняты с объекта;
* расширение путем порождения подклассов по каким-то причинам неудобно или невозможно. Иногда приходится реализовывать много независимых расширений, так что порождение подклассов для поддержки всех возможных комбинаций приведет к стремительному росту их числа. В других случаях определение класса может быть скрыто или почему-либо еще недоступно, так что породить от него подкласс нельзя



**Участники**

* Component (VisualComponent) — компонент:
  + определяет интерфейс для объектов, на которые могут быть динамически возложены дополнительные обязанности;
* ConcreteComponent (TextView) — конкретный компонент:
  + определяет объект, на который возлагаются дополнительные обязанности;
* Decorator — декоратор:
  + хранит ссылку на объект Component и определяет интерфейс, соответствующий интерфейсу Component;
* ConcreteDecorator (BorderDecorator, ScrollDecorator) — конкретный декоратор:
  + возлагает дополнительные обязанности на компонент.



**Участники**

* Component (Character) — компонент:
* определяет интерфейс для объектов, на которые могут быть динамически возложены дополнительные обязанности;
* ConcreteComponent (Hunter,Warrior, Mage) — конкретный компонент:
* определяет объект, на который возлагаются дополнительные обязанности;
* WeaponDecorator — декоратор:
* хранит ссылку на объект Component и определяет интерфейс, соответствующий интерфейсу Component;
* ConcreteDecorator (Bow, Wand, Sword) — конкретный декоратор:
* возлагает дополнительные обязанности на компонент

abstract class Character

{

public Character (string n)

{

this.Name = n;

}

public string Name { get; protected set; }

public abstract int Helth();

public abstract int Attack();

}

class Warrior : Character

{

public Warrior() : base ("Воин") { }

public override int Helth()

{

return 100;

}

public override int Attack()

{

return 10;

}

}

class Mage : Character

{

public Mage() : base("Маг") { }

public override int Helth()

{

return 60;

}

public override int Attack()

{

return 20;

}

}

class Hunter : Character

{

public Hunter() : base("Охотник") { }

public override int Helth()

{

return 80;

}

public override int Attack()

{

return 15;

}

}

abstract class WeaponDecorator : Character

{

protected Character character;

public WeaponDecorator(string n, Character character) : base(n)

{

this.character = character;

}

}

class Sword: WeaponDecorator

{

public Sword(Character ch) : base (ch.Name + " с мечом", ch) { }

public override int Attack()

{

return character.Attack() + 10;

}

public override int Helth()

{

return character.Helth() + 5;

}

}

class Bow : WeaponDecorator

{

public Bow(Character ch) : base(ch.Name + " с луком", ch) { }

public override int Attack()

{

return character.Attack() + 15;

}

public override int Helth()

{

return character.Helth() + 10;

}

}

class Wand : WeaponDecorator

{

public Wand(Character ch) : base(ch.Name + " с жезлом", ch) { }

public override int Attack()

{

return character.Attack() + 5;

}

public override int Helth()

{

return character.Helth() + 20;

}

}

class Program

{

static void Main()

{

Character player1 = new Warrior();

player1 = new Sword(player1);

Console.WriteLine($"Класс: {player1.Name}");

Console.WriteLine($"Урон:{player1.Attack()}");

Console.WriteLine($"хп:{player1.Helth()}");

Character player2 = new Mage();

player2 = new Wand(player2);

Console.WriteLine($"Класс: {player2.Name}");

Console.WriteLine($"Урон:{player2.Attack()}");

Console.WriteLine($"хп:{player2.Helth()}");

Character player3 = new Hunter();

player3 = new Bow(player3);

Console.WriteLine($"Класс: {player3.Name}");

Console.WriteLine($"Урон:{player3.Attack()}");

Console.WriteLine($"хп:{player3.Helth()}");

Character player4 = new Warrior();

//player4 = new Sword(player4);

//player4 = new Bow(player4);

player4 = new Bow(new Wand(new Sword(player4)));

Console.WriteLine($"Класс: {player4.Name}");

Console.WriteLine($"Урон:{player4.Attack()}");

Console.WriteLine($"хп:{player4.Helth()}");

}

}