软件架构与设计模式

软件工程系 刘慰

liuwei@nbu.edu.cn

QQ 秀程序

设计一个类似于QQ秀的程序，可以将不同的服饰进行组合。服饰的类别有很多种，组合方式也是各种各样。

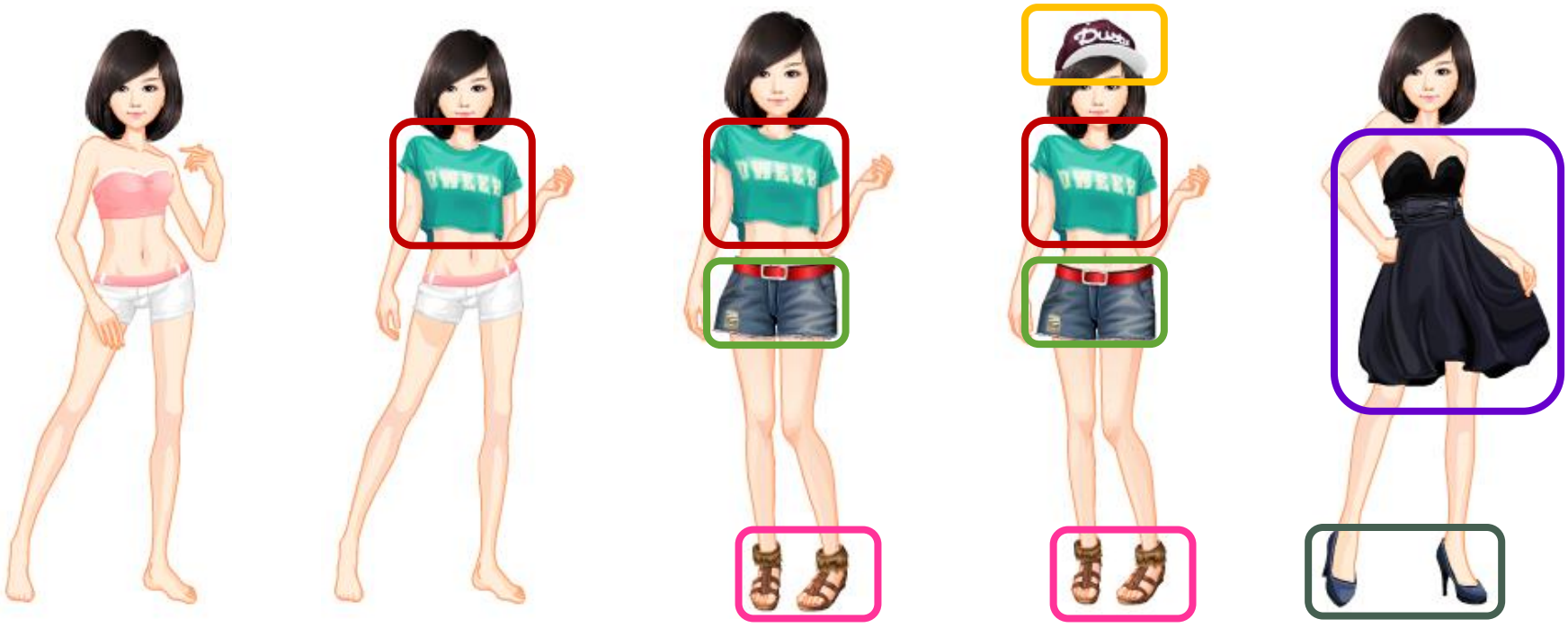
问题

此程序能否使用上节课介绍的建造者模式实现？

```
class PersonDirector
{
    private PersonBuilder _builder;

    public PersonDirector(PersonBuilder builder)
    {
        _builder = builder;
    }

    public void CreatePerson()
    {
        _builder.BuildHead();
        _builder.BuildBody();
        _builder.BuildLeftArm();
        _builder.BuildRightArm();
        _builder.BuildLeftLeg();
        _builder.BuildRightLeg();
    }
}
```

QQ秀程序与上节课中画小人程序的区别

- QQ秀程序中的组件个数不确定



结论

- QQ秀程序并不适合使用建造者模式来实现
  - 因为建造者模式只适用于组件数固定的情况。
- 生活中还有没有类似于 QQ 秀这样，在一个基本对象上任意添加各种特性或者功能的例子？

浓缩

美式

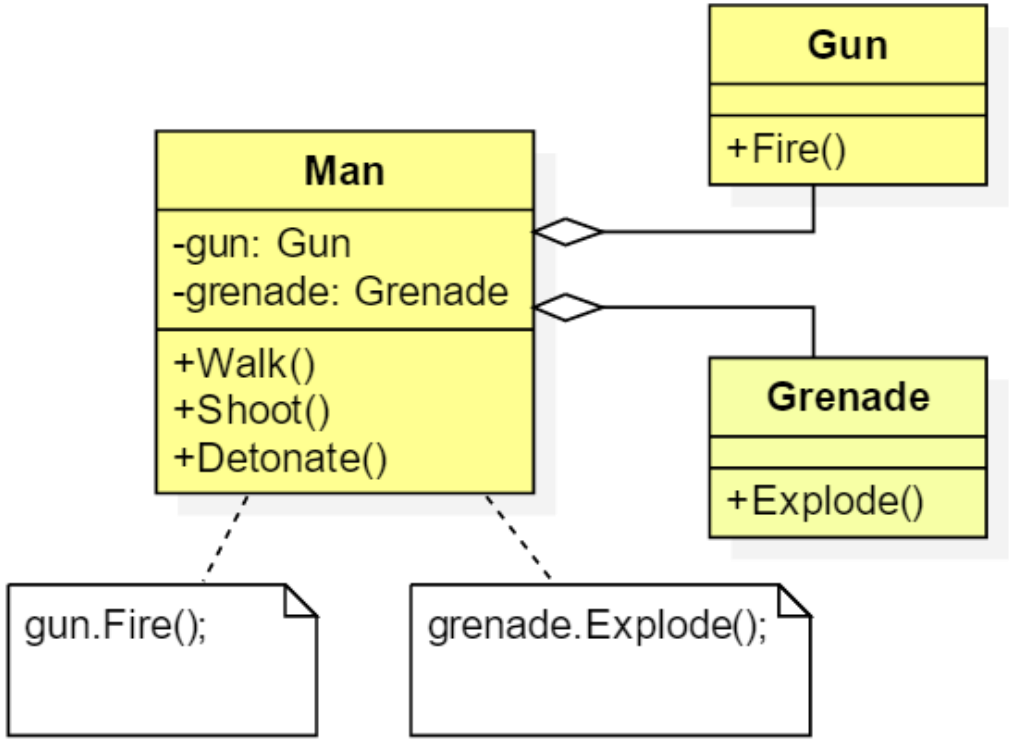
拿铁

摩卡

不同种类咖啡之间的区别

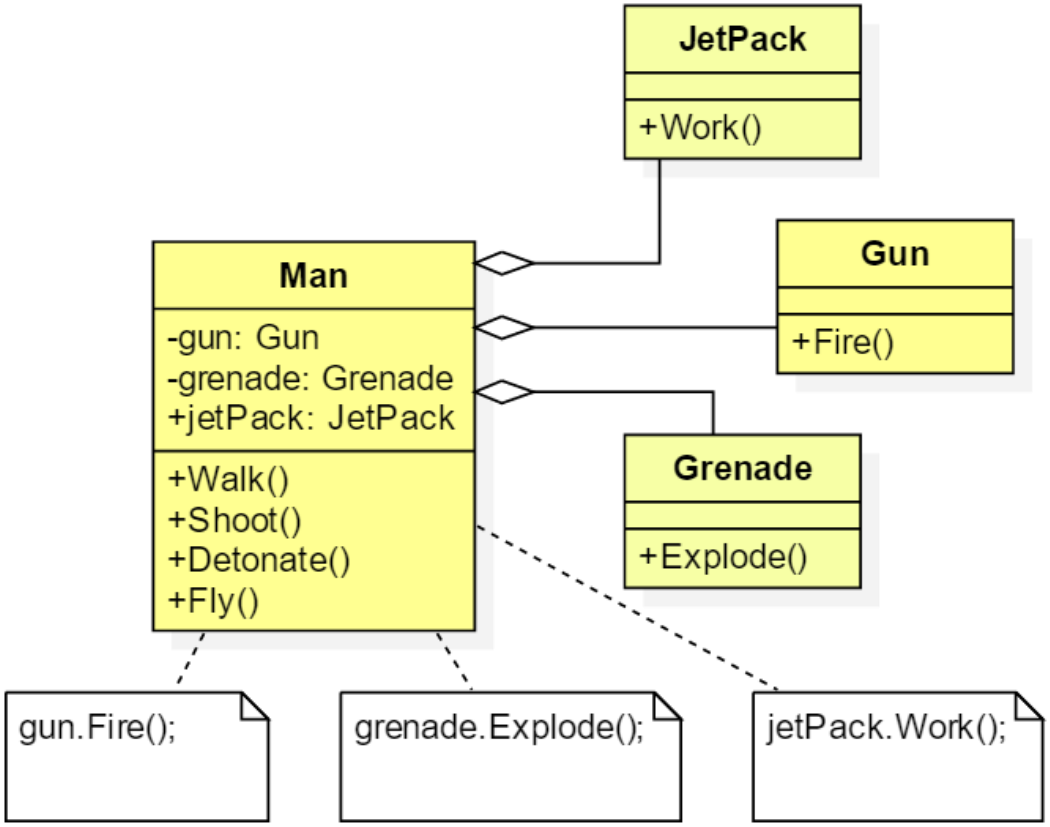
给类增加特性与功能的方法

- 组合



组合的缺点

1. 每增加一种新的功能，都需要修改已有的类，违反了“开放——封闭原则”。



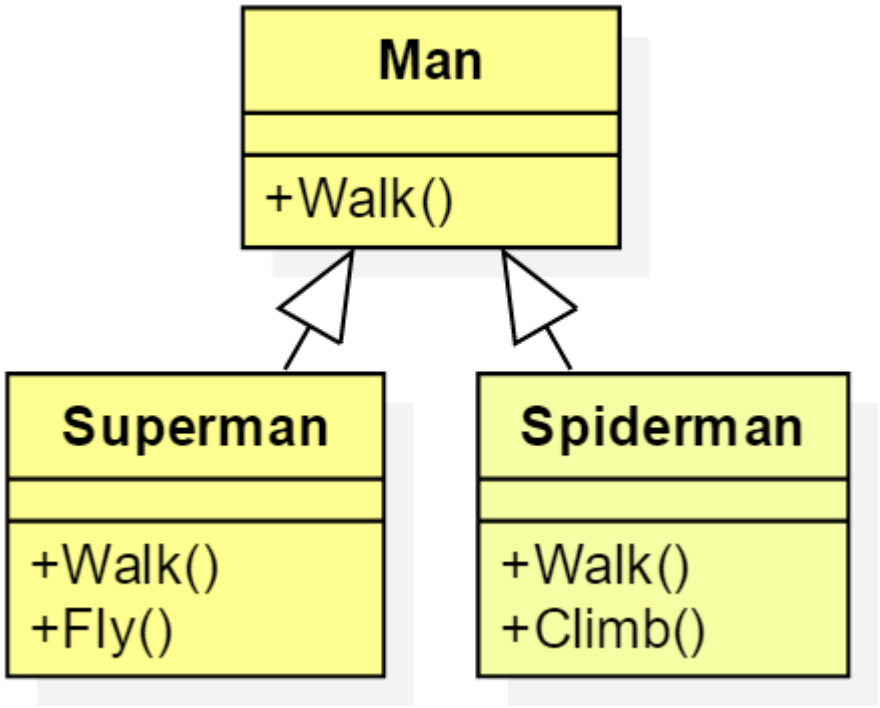
组合的缺点

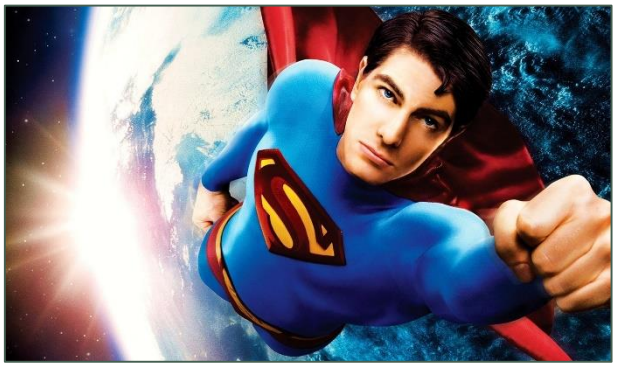
2. 组合只能给基本对象增加新的功能，而不能修改基本对象已有的功能。

   - 假如需要根据不同的配料计算每种咖啡的费用，那么就无法在各种配料中重写主料（Espresso）的 Cost() 方法。

还有别的方法吗？

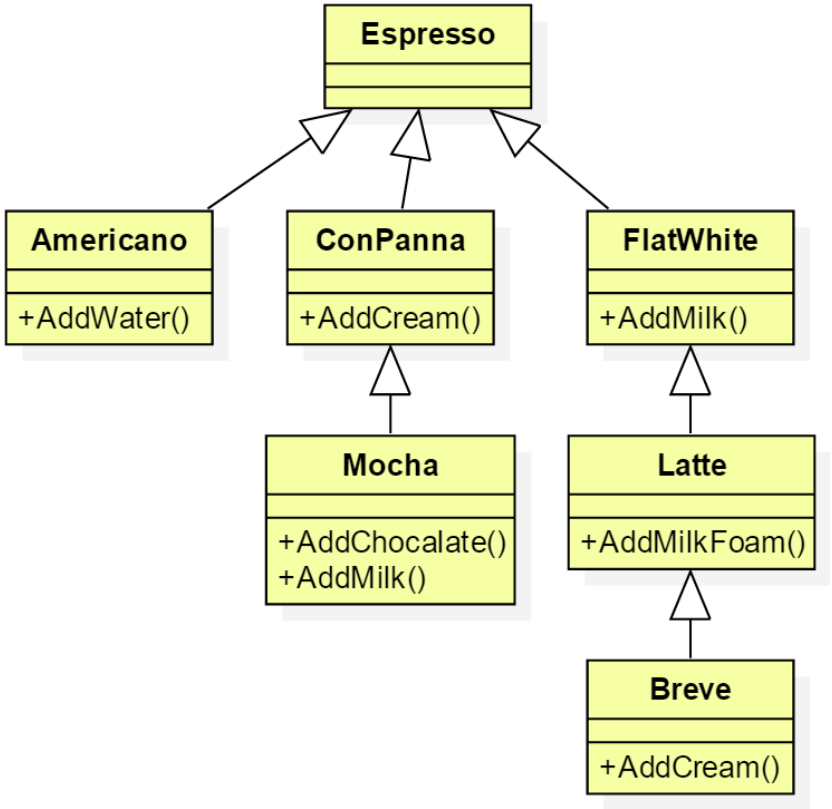
- 继承





对于咖啡的例子，可以用继承吗？

- 答案：可以



- 5 种配料，任意组合，一共有多少种可能？
  - $C_5^0 + C_5^1 + C_5^2 + C_5^3 + C_5^4 + C_5^5$
    $= 1 + 5 + 10 + 10 + 5 + 1$
    $= 32$
- 使用继承的缺点
  - 随着功能的增加，子类的个数会成倍增长。

既然组合与继承有各自的缺点

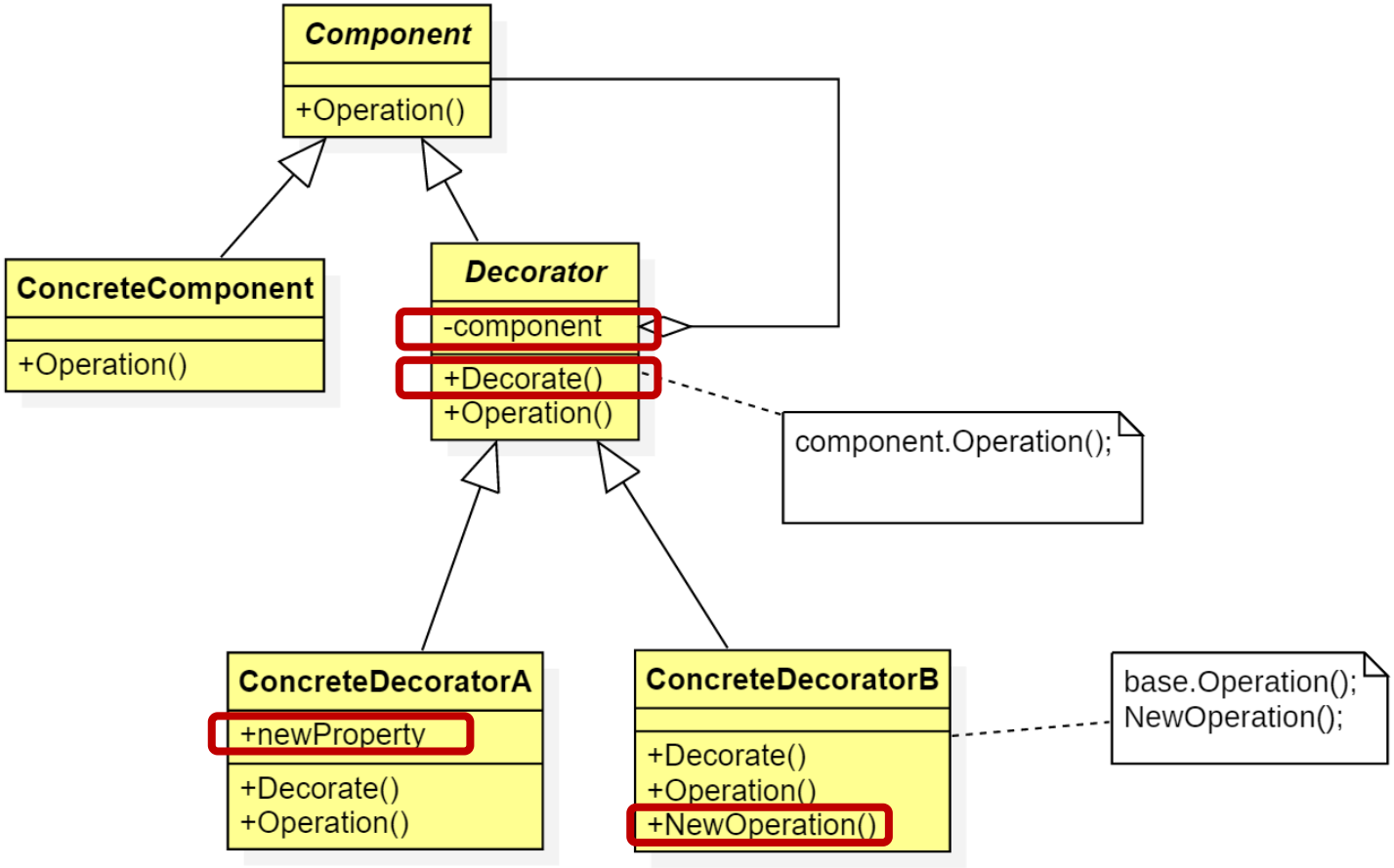
- 将组合与继承结合起来
  - 将需要添加的特性/行为封装到一个基本类的新子类中，然后将这个“包装”类作为“被包装”的类的成员变量。

装饰模式（Decorator Pattern）

- 定义
  - 又叫包装（Wrapper）模式。该模式动态地给一个对象添加一些额外的功能，就扩展功能来说，提供了比继承更具弹性的替代方案。
  - 是结构型模式的一种。

装饰模式（Decorator Pattern）

- 模式结构 UML 类图

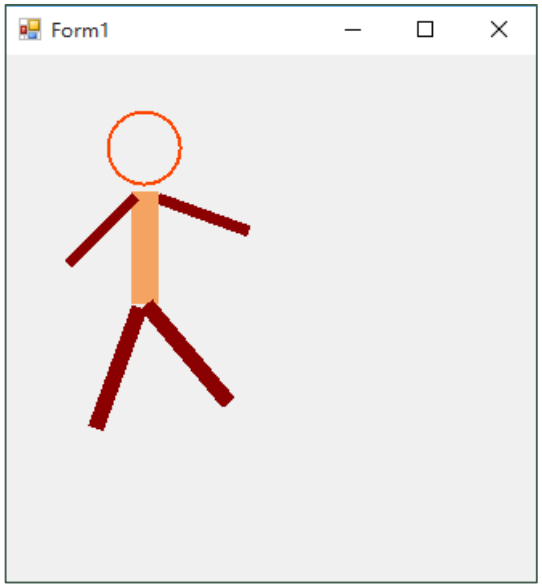


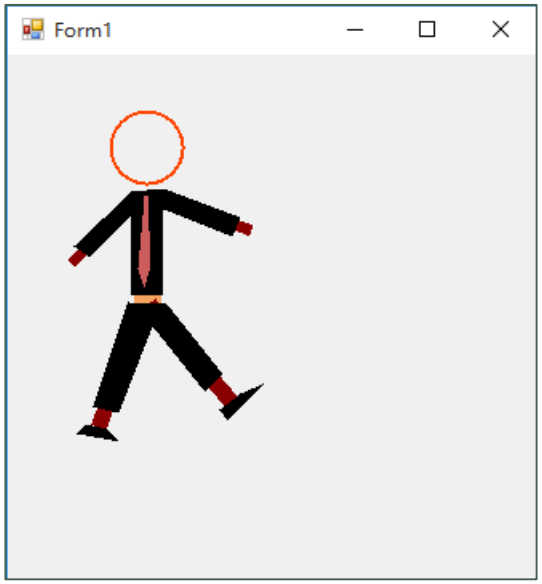
装饰模式各角色及其职能

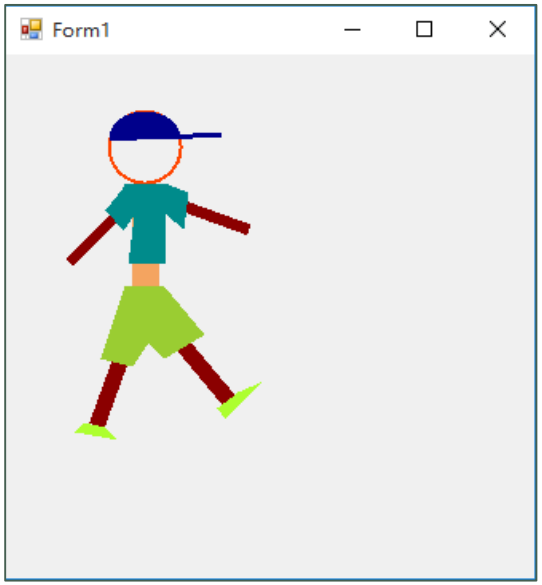
- Component（抽象组件）
  - 定义了可能需要增加特性与行为的基本对象的抽象接口。
- ConcreteComponent（具体组件）
  - 实现 Componenet 接口，定义了具体的 Component 对象。
- Decorator（抽象装饰类）
  - 是 Component 类的子类，包含一个 Componet 类型的私有变量。
- ConcreteDecorator（具体装饰类）
  - 是 Decorator 类的子类，封装需要添加的新特性或新行为。

QQ秀简易版

- 面向过程的实现



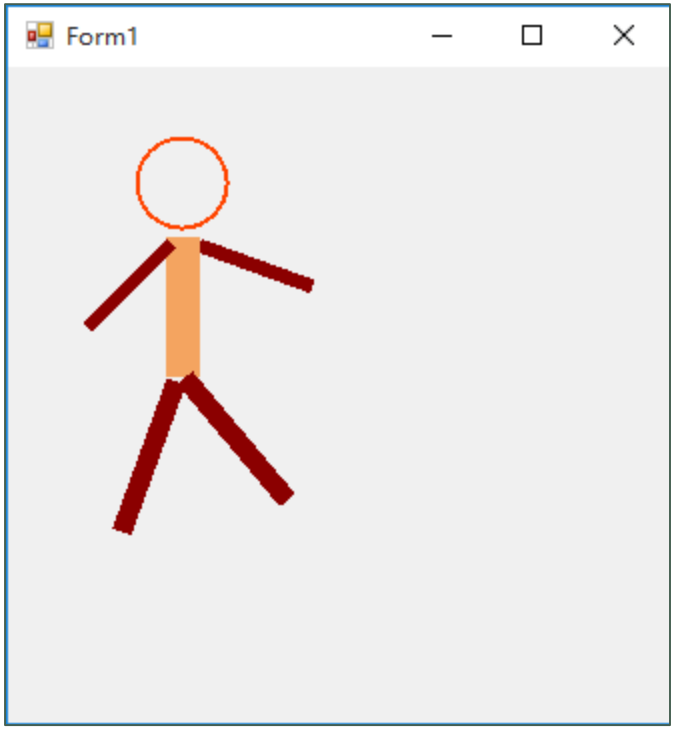




QQ秀简易版

- 画小人的部分

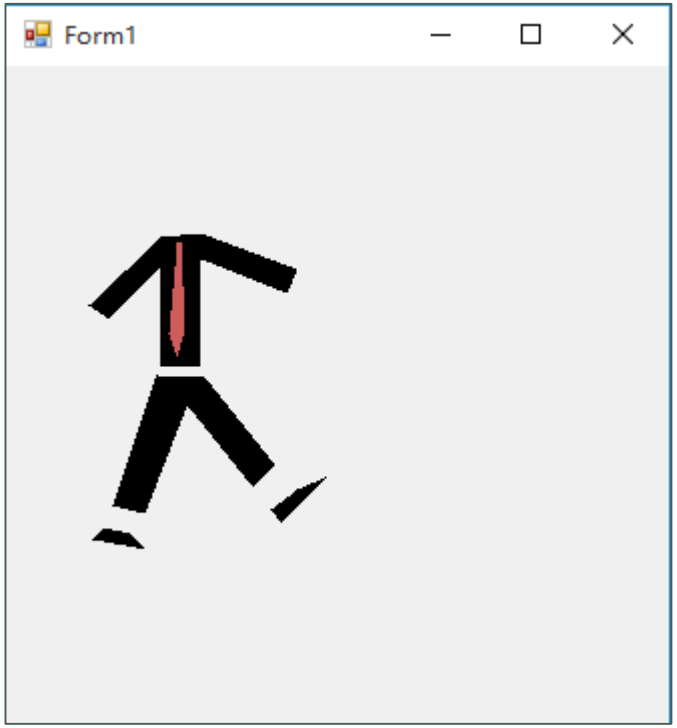
```
#region 画瘦小人代码

Graphics gThin = pictureBox1.CreateGraphics();

//head
gThin.DrawEllipse(new Pen(Color.OrangeRed, 2), 50, 20, 45, 45);

//right arm
gThin.TranslateTransform(80, 77);
gThin.RotateTransform(290);
gThin.FillRectangle(new SolidBrush(Color.DarkRed), 0, 0, 7, 60);
gThin.ResetTransform();
//right arm end

//body
gThin.FillRectangle(new SolidBrush(Color.SandyBrown), 65, 70, 17, 70);

//left arm
gThin.TranslateTransform(65, 70);
gThin.RotateTransform(45);
gThin.FillRectangle(new SolidBrush(Color.DarkRed), 0, 0, 7, 60);
gThin.ResetTransform();
//left arm end


//left leg
gThin.TranslateTransform(65, 140);
gThin.RotateTransform(20);
gThin.FillRectangle(new SolidBrush(Color.DarkRed), 0, 0, 10, 80);
gThin.ResetTransform();
//left leg end

//right leg
gThin.TranslateTransform(70, 143);
gThin.RotateTransform(320);
gThin.FillRectangle(new SolidBrush(Color.DarkRed), 0, 0, 10, 80);
gThin.ResetTransform();
//right leg end

#endregion
```



QQ秀简易版

- 画服饰的部分（正式装扮）

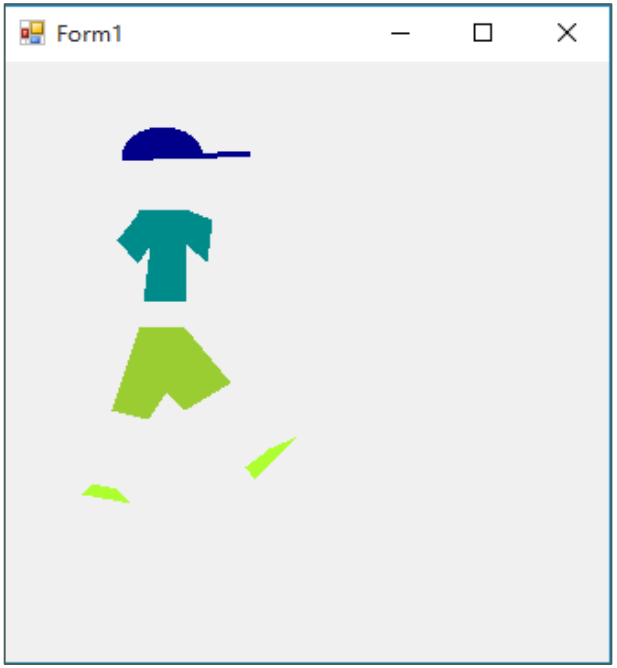
```
#region 正装装扮

//shoes
gThin.FillPolygon(
    new SolidBrush(Color.Black),
    new Point[]
    {
        new Point(34, 215), new Point(29, 221), new Point(56, 226), new Point(48, 218)
    });

gThin.FillPolygon(
    new SolidBrush(Color.Black),
    new Point[]
    {
        new Point(118, 206), new Point(123, 213), new Point(147, 189), new Point(131, 196)
    });
//shoes end

//pants
gThin.FillPolygon(
    new SolidBrush(Color.Black),
    new Point[]
    {
        new Point(61, 138), new Point(39, 204), new Point(55, 208), new Point(76, 154), new Point(109, 195),
        new Point(120, 184), new Point(85, 140), new Point(62, 139)
    });


//pants end

//jacket
gThin.FillPolygon(
    new SolidBrush(Color.Black),
    new Point[]
    {
        new Point(63, 70), new Point(27, 104), new Point(37, 111), new Point(63, 84), new Point(63, 135),
        new Point(83, 135), new Point(83, 81), new Point(126, 98), new Point(131, 86), new Point(83, 68)
    });
//jacket end

//tie
gThin.FillPolygon(
    new SolidBrush(Color.IndianRed),
    new Point[]
    {
        new Point(71, 73), new Point(67, 118), new Point(71, 130), new Point(75, 116), new Point(73, 72)
    });

//tie end

#endregion
```



QQ秀简易版

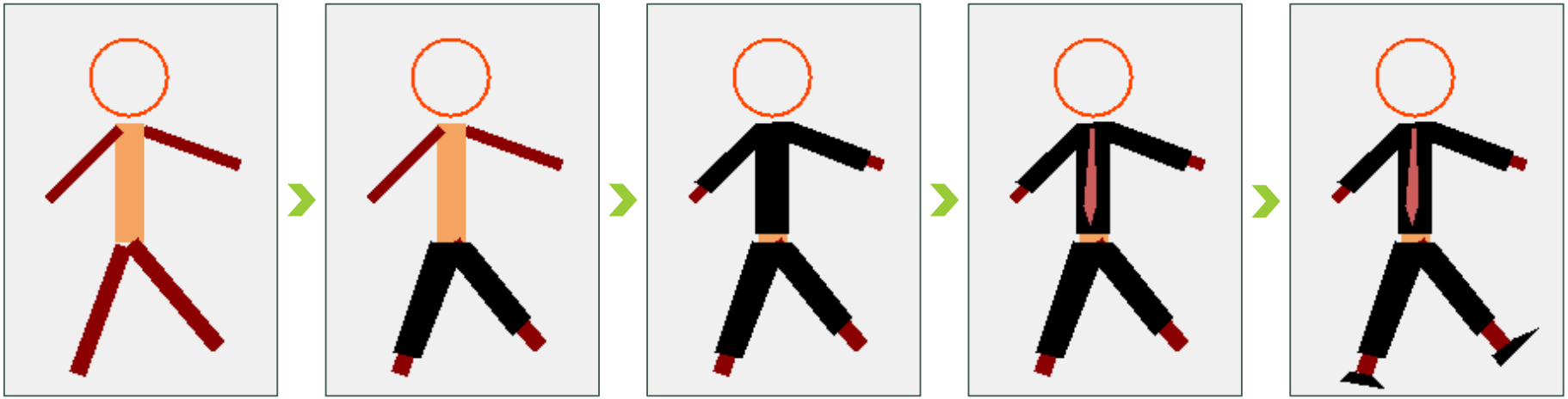
- 画服饰的部分（休闲装扮）

```
#region 休闲装扮

//hat
gThin.FillPie(new SolidBrush(Color.DarkBlue), 50, 20, 45, 35, 180, 180);
gThin.DrawLine(new Pen(Color.DarkBlue, 3), 50, 37, 120, 35);
//hat end

//shorts
gThin.FillPolygon(
    new SolidBrush(Color.YellowGreen),
    new Point[]
    {
        new Point(60, 130), new Point(85, 130), new Point(110, 160), new Point(85, 175), new Point(75, 165),
        new Point(65, 180), new Point(45, 175)
    });
//shorts end

//shirt
gThin.FillPolygon(
    new SolidBrush(Color.DarkCyan),
    new Point[]
    {
        new Point(59, 68), new Point(48, 82), new Point(59, 95), new Point(66, 85), new Point(62, 116), new Point(86, 116),
        new Point(85, 84), new Point(97, 94), new Point(100, 71), new Point(85, 65), new Point(60, 66)
    });


//shirt end

//shoes
gThin.FillPolygon(
    new SolidBrush(Color.GreenYellow),
    new Point[]
    {
        new Point(34, 215), new Point(29, 221), new Point(56, 226), new Point(48, 218)
    });

gThin.FillPolygon(
    new SolidBrush(Color.GreenYellow),
    new Point[]
    {
        new Point(118, 206), new Point(123, 213), new Point(147, 189), new Point(131, 196)
    });
//          //shoes end

#endregion
```



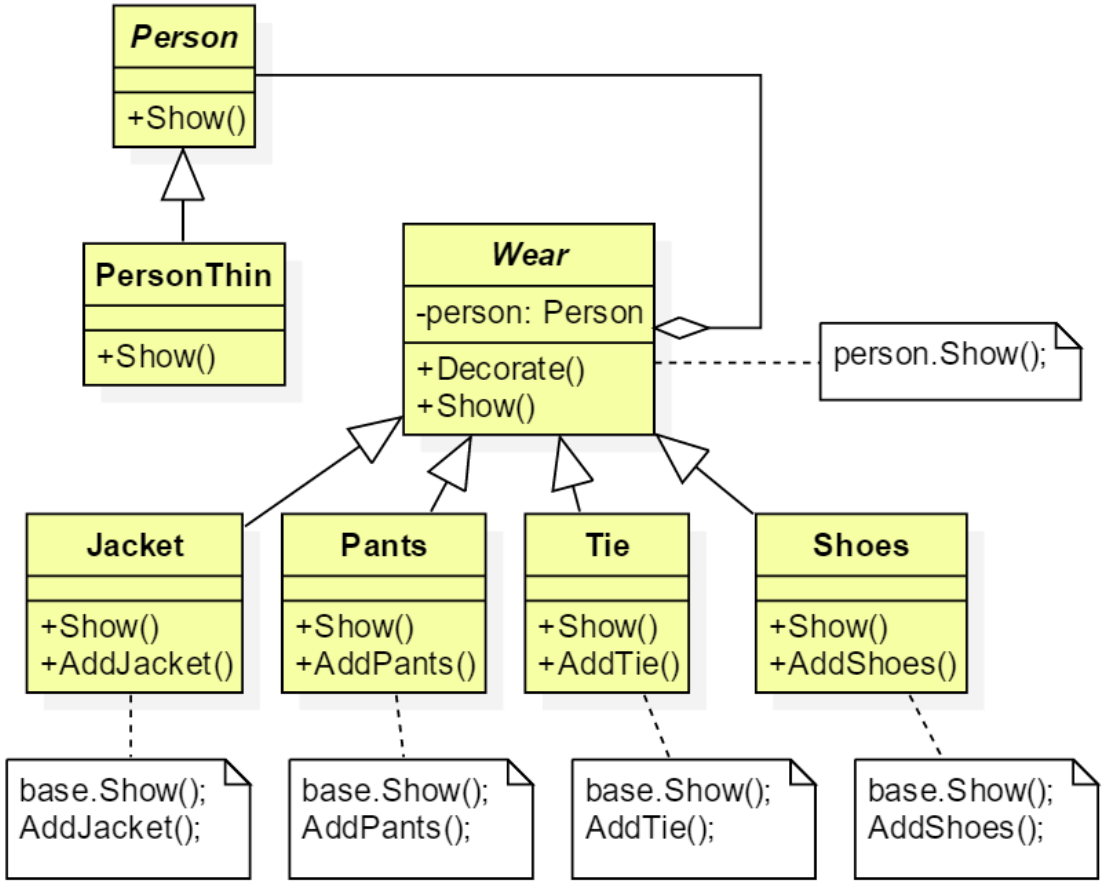
QQ秀简易版

- 改进思路（应用装饰模式后）
  - 将服饰作为“包装类”，每添加一种服饰，就相当于在原有基本对象（小人）的基础上“套”上一种新的服饰。



使用了装饰模式的QQ秀简易版

- UML 类图



使用了装饰模式的QQ秀简易版

- Person 类（抽象组件类）

```
abstract class Person
{
    protected Graphics _g;

    public Person()
    {
    }

    public Person(Graphics g)
    {
        _g = g;
    }

    public abstract void Show();
}
```

使用了装饰模式的QQ秀简易版

- PersonThin 类（具体组件类）

```
class PersonThin : Person
{
    public PersonThin(Graphics g) : base(g)
    {
    }


    public override void Show()
    {
        #region 画瘦小人代码

        #endregion
    }
}
```

使用了装饰模式的QQ秀简易版

- Wear 类（抽象装饰类）

```
abstract class Wear : Person
{
    private Person _person;

    public Wear(Graphics g) : base(g)
    {
    }

    public void Decorate(Person person)
    {
        _person = person;
    }

    public override void Show()
    {
        _person.Show();
    }
}
```

使用了装饰模式的QQ秀简易版

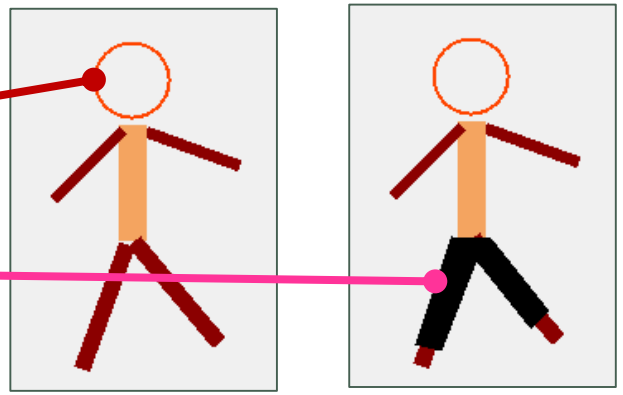
- Jacket 类（具体装饰类）

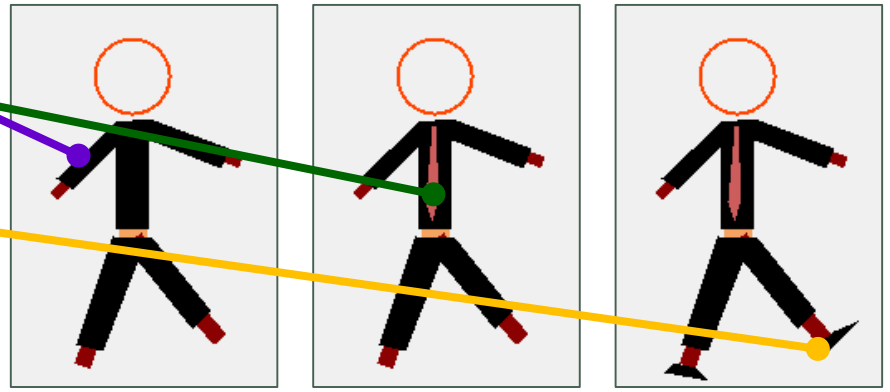
```
class Jacket : Wear
{
    public Jacket(Graphics g) : base(g)
    {
    }

    public override void Show()
    {
        base.Show();
        AddJacket();
    }

    public void AddJacket()
    {
        //jacket
        base._g.FillPolygon(
            new SolidBrush(Color.Black),
            new Point[]
            {
                new Point(63, 70), new Point(27, 104), new Point(37, 111), new Point(63, 84), new Point(63, 135),
                new Point(83, 135), new Point(83, 81), new Point(126, 98), new Point(131, 86), new Point(83, 68)
            });
        //jacket end
    }
}
```

使用了装饰模式的QQ秀简易版

- 客户端

```
private void pictureBox1_Click(object sender, EventArgs e)
{
    Graphics g = pictureBox1.CreateGraphics();

    Person person = new PersonThin(g);

    Pants pants = new Pants(g);
    pants.Decorate(person);

    Jacket jacket = new Jacket(g);
    jacket.Decorate(pants);

    Tie tie = new Tie(g);
    tie.Decorate(jacket);

    Shoes shoes = new Shoes(g);
    shoes.Decorate(tie);
    shoes.Show();
}
```

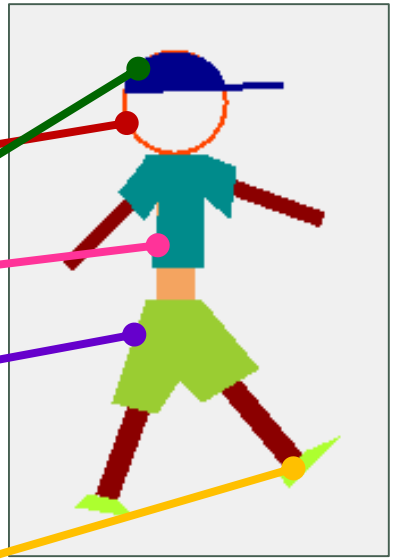




使用了装饰模式的QQ秀简易版

- 换一套装扮

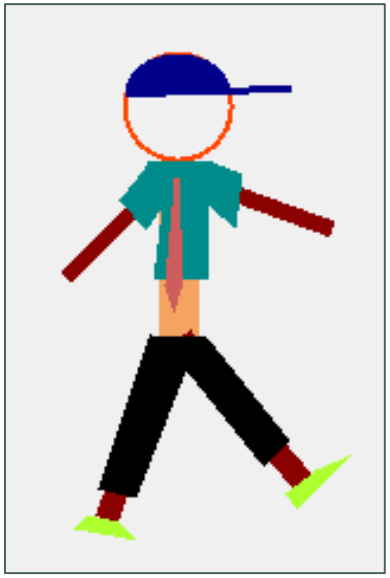
```
private void pictureBox1_Click(object sender, EventArgs e)
{
    Graphics g = pictureBox1.CreateGraphics();

    Person person = new PersonThin(g);

    Shirt shirt = new Shirt(g);
    shirt.Decorate(person);

    Shorts shorts = new Shorts(g);
    shorts.Decorate(shirt);

    Hat hat = new Hat(g);
    hat.Decorate(shorts);

    Sneakers sneakers = new Sneakers(g);
    sneakers.Decorate(hat);
    sneakers.Show();
}
```



使用了装饰模式的QQ秀简易版

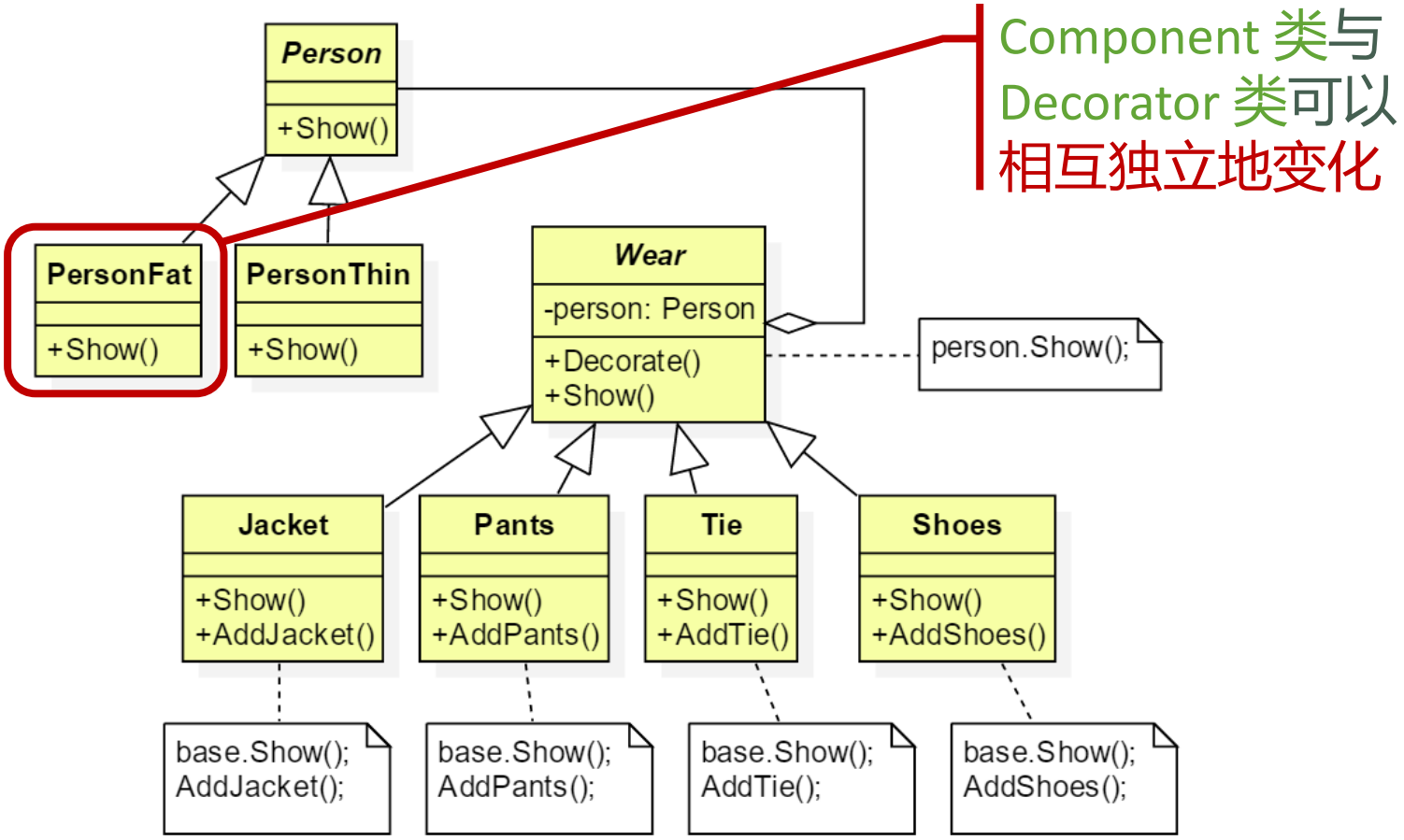
- 甚至可以混搭

```
private void pictureBox1_Click(object sender, EventArgs e)
{
    Graphics g = pictureBox1.CreateGraphics();

    Person person = new PersonThin(g);

    Pants pants = new Pants(g);
    pants.Decorate(person);

    Shirt shirt = new Shirt(g);
    shirt.Decorate(pants);

    Hat hat = new Hat(g);
    hat.Decorate(shirt);

    Tie tie = new Tie(g);
    tie.Decorate(hat);

    Sneakers sneakers = new Sneakers(g);
    sneakers.Decorate(tie);
    sneakers.Show();
}
```



增加了新具体组件类的QQ秀简易版

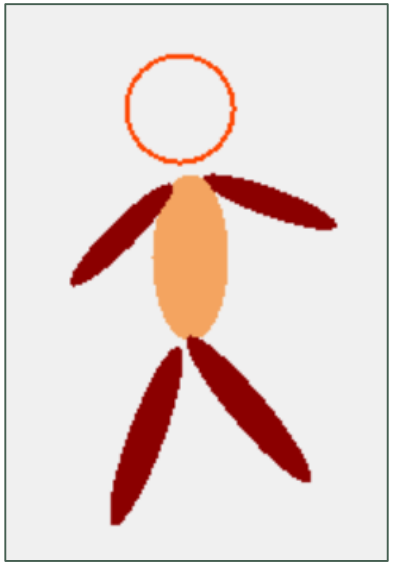
- 新的 UML 类图



增加了具体组件类的QQ秀简易版

- PersonFat 类

```
class PersonFat : Person
{
    public PersonFat(Graphics g) : base(g)
    {
    }


    public override void Show()
    {
        #region 画胖小人代码

        ...

    }
}
```



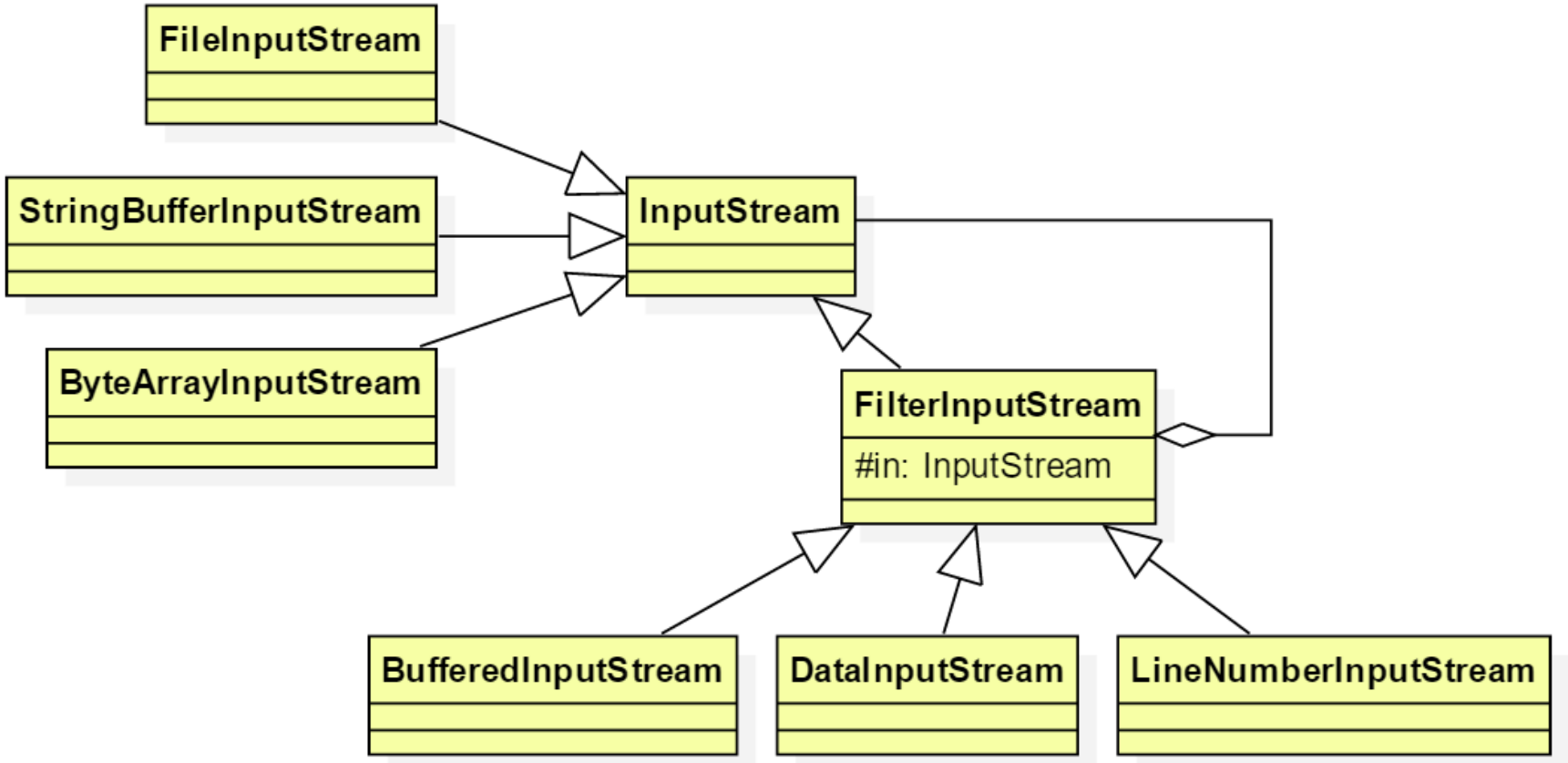
增加了具体组件类的QQ秀简易版

- 同样的服饰可以穿到胖小人的身上

```
private void pictureBox1_Click(object sender, EventArgs e)
{
    Graphics g = pictureBox1.CreateGraphics();

    Person person = new PersonFat(g);

    Pants pants = new Pants(g);
    pants.Decorate(person);

    Jacket jacket = new Jacket(g);
    jacket.Decorate(pants);

    Tie tie = new Tie(g);
    tie.Decorate(jacket);

    Shoes shoes = new Shoes(g);
    shoes.Decorate(tie);
    shoes.Show();
}
```

装饰模式在实际工程中的应用

- JDK 中的 IO 包



装饰模式在实际工程中的应用

- JDK 中的 IO 包
  - 在 InputStream 提供的基本方法的基础上，FilterInputStream 的子类提供了更多附加功能：
    1. BufferedInputStream 类会提供一个内部的字节数组作为输入缓存。
    2. DataInputStream 类可以用与机器无关的方式从底层数据流中读取基本 Java 数据类型。
    3. LineNumberInputStream 类可以跟踪当前行号。

装饰模式在实际工程中的应用

- FilterInputStream 类

```
class FilterInputStream extends InputStream {
    /**
     * The input stream to be filtered.
     */
    protected volatile InputStream in;

    /**
     * Creates a <code>FilterInputStream</code>
     * by assigning the  argument <code>in</code>
     * to the field <code>this.in</code> so as
     * to remember it for later use.
     *
     * @param   in   the underlying input stream, or <code>null</code> if
     *               this instance is to be created without an underlying stream.
     */
    protected FilterInputStream(InputStream in) {
        this.in = in;
    }
```

装饰模式在实际工程中的应用

- 客户端

```
try {
    InputStream inputStream = new FileInputStream(new File("c:\\test\\test.txt"));
    BufferedInputStream bufferedInputStream = new BufferedInputStream(inputStream);
    LineNumberInputStream lineNumberInputStream = new LineNumberInputStream(bufferedInputStream);

    int result = lineNumberInputStream.read();
    while (result != '\n') {
        result = lineNumberInputStream.read();
    }
    System.out.println(lineNumberInputStream.getLineNumber());//2
    lineNumberInputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

另一个应用

- 经典游戏 FC 版《坦克大战》

坦克大战简易版

- 设计要求

1. 坦克有初始值为 100 点的生命值（HealthPower），每受到一次攻击扣 20 点生命值。生命值为 <= 0 时，坦克被击毁。
2. 如果“吃”到“护盾”（Shield），则增加 100 点防御值（ShieldPower）。受到攻击时扣 20 点防御值，不会扣生命值。防御值 <= 0 时，护盾消失。
3. 如果“吃”到“船”（Boat），则可以在水面上移动。
4. 如果“吃”到“钟”（Clock），则所有敌方坦克冻结（Freeze）30 秒。

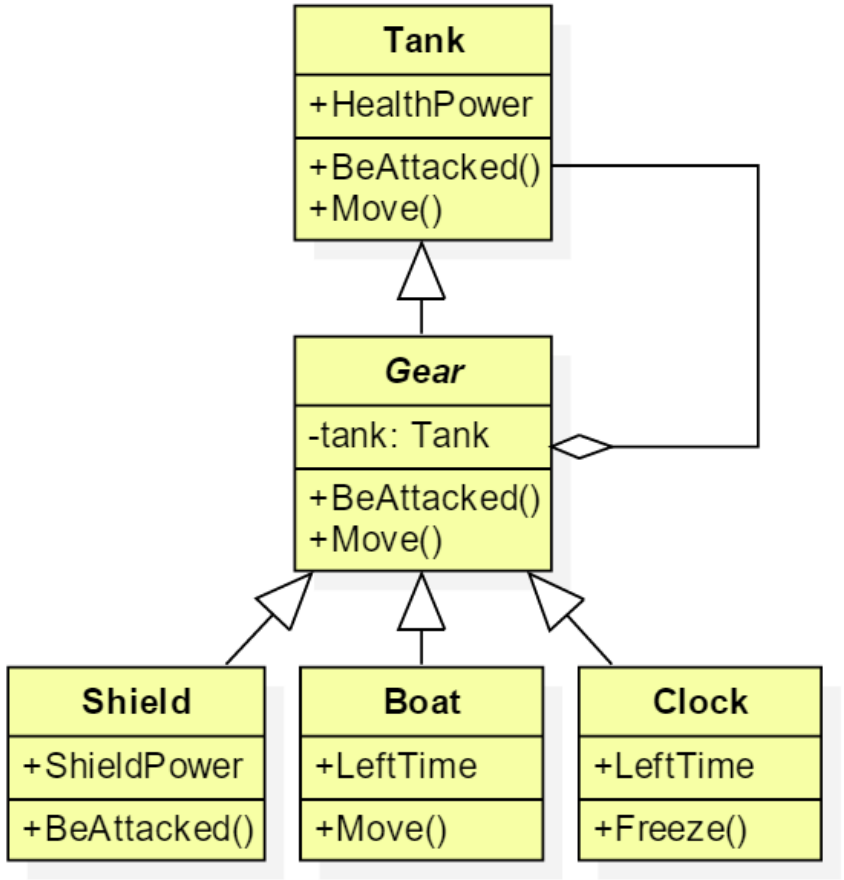
坦克大战简易版

- 实现方法

1. 将坦克（Tank）作为“组件类（Component）”，提供 HealthPower 属性与 BeAttacked() 及 Move() 两个方法。
2. 将“装备（Gear）类”作为“抽象装饰类（Decorator）”，并重写 Tank 类的 BeAttacked() 与 Move() 方法。
3. “护盾（Shield）类”作为“具体装饰类（ConcreteDecorator）”，继承自“装备类”，并增加一个 ShieldPower 属性。“船（Boat）类”类似。
4. “钟（Clock）类”也继承自“装备类”，并增加一个 Freeze() 方法。

坦克大战简易版

- UML 类图



此处省略了
抽象组件类
（Component）

坦克大战简易版

- Tank 类

```
class Tank
{
    public int HealthPower { get; set; } = 100;

    public virtual void BeAttacked()
    {
        const int damagePerAttack = 20;
        HealthPower = HealthPower - damagePerAttack;

        if (HealthPower > 0)
        {
            Console.WriteLine("坦克受到攻击，HP减{0}，仍有{1}", damagePerAttack, HealthPower);
        }
        else
        {
            Console.WriteLine("坦克受到攻击，被击毁");
        }
    }

    public virtual void Move()
    {
        Console.WriteLine("移动中");
    }
}
```

坦克大战简易版

- Gear 类

```
class Gear : Tank
{
    private Tank _tank;

    public Gear(Tank tank)
    {
        _tank = tank;
    }

    public override void BeAttacked()
    {
        _tank.BeAttacked();
    }

    public override void Move()
    {
        _tank.Move();
    }
}
```

坦克大战简易版

- Shield 类

```
class Shield : Gear
{
    public int ShieldPower { get; set; } = 100;

    public Shield(Tank tank) : base(tank)
    {

    }

    public override void BeAttacked()
    {
        const int damagePerAttack = 20;
        ShieldPower = ShieldPower - damagePerAttack;
        if (ShieldPower > 0)
        {
            Console.WriteLine("坦克受到攻击，护盾减{0}，仍有{1}，坦克未受损害", damagePerAttack, ShieldPower);
        }
        else if (ShieldPower == 0)
        {
            Console.WriteLine("坦克受到攻击，护盾减{0}，护盾消失，坦克未受损害", damagePerAttack);
        }
        else
        {
            base.BeAttacked();
        }
    }
}
```

坦克大战简易版

- Boat 类

```
class Boat : Gear
{
    public int LeftTime { get; set; } = 60;

    public Boat(Tank tank) : base(tank)
    {

    }

    public override void Move()
    {
        if (LeftTime > 0)
        {
            Console.Write("现在可以过河");
        }
        base.Move();
    }
}
```

坦克大战简易版

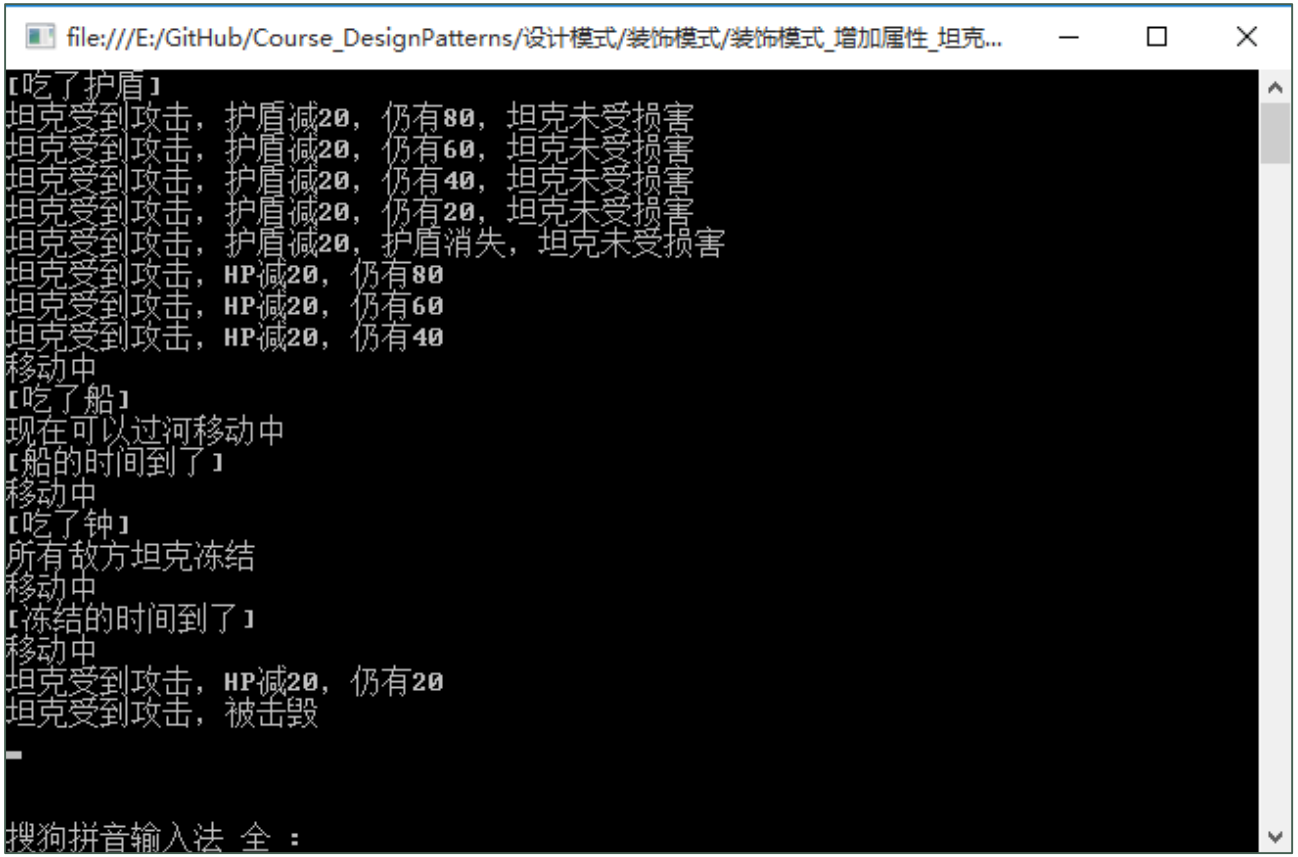
- Clock 类

```
class Clock : Gear
{
    public int LeftTime { get; set; } = 30;

    public Clock(Tank tank) : base(tank)
    {

    }

    public void Move()
    {
        Freeze();
        base.Move();
    }

    public void Freeze()
    {
        if (LeftTime > 0)
        {
            Console.WriteLine("所有敌方坦克冻结");
        }
    }
}
```

坦克大战简易版

- 最后的效果

```
Tank tank = new Tank();

Console.WriteLine("[吃了护盾]");
Shield shield = new Shield(tank);

shield.BeAttacked();
shield.BeAttacked();
shield.BeAttacked();
shield.BeAttacked();
shield.BeAttacked();
shield.BeAttacked();
shield.BeAttacked();
shield.BeAttacked();


tank.Move(); //没吃船之前的移动

Console.WriteLine("[吃了船]");
Boat boat = new Boat(shield);
boat.Move(); //吃了船之后的移动

Console.WriteLine("[船的时间到了]");
boat.LeftTime = -1;
boat.Move(); //船消失之后的移动

Console.WriteLine("[吃了钟]");
Clock clock = new Clock(boat);
clock.Move();

Console.WriteLine("[冻结的时间到了]");
clock.LeftTime = -1;
clock.Move();

clock.BeAttacked();
clock.BeAttacked();

Console.ReadLine();
```



装饰模式总结

- 优点
  - 可以提供比继承更多的灵活性，以一种动态的方式扩展一个对象的功能，并通过使用不同的具体装饰类以及这些装饰类的排列组合，创造出很多不同行为的组合。
  - 具体构建类与具体装饰类可以独立变化。
- 缺点
  - 使用装饰模式时将产生很多小对象，且比继承更容易出错。
- 适用场景
  - 在不影响其他对象的情况下，以动态、透明的方式给单个对象添加职责。
  - 需要动态地给一个对象增加功能，这些功能也可以动态地被撤销。

课后作业

苹果（Apple）与小米（Mi）的手机（Mobilephone）都有短信（SendMessage）和通话（Call）的功能。

现在有三种功能模块：蓝牙（Bluetooth）、GPS定位（GPS）与摄像头（Camera），可以任意地安装在手机上。蓝牙模块会给手机增加连接（Connect()）功能；GPS 给手机增加方位（Location）属性；摄像头将手机原有的通话功能升级到带视频的通话功能（在通话时显示画面）。

试用装饰模式编写一个 Console 程序进行模拟，同时需要画出 UML 图。要求与之前作业相同。