我将带大家一起去看一看在面向对象编程中出现的23大设计模式和9大设计原则。设计原则是我们开发研究设计模式的目的，而设计模式则是达到设计原则的手段和方法。在记录笔记过程中，会在不同的设计模式中穿插讲解设计原则。
本笔记知识点主要来自于《Head First 设计模式》

本次笔记将讲解三种设计模式，实际上再抽象一点是两种，如果把简单工厂模式看成工厂方法模式的一种特例的话。这两种设计模式紧密相连却又彼此不同。

创建型模式一、二、三：简单工厂模式，工厂方法模式，抽象工厂模式

定义：
简单工厂模式(Simple Factory Pattern)：又被称作静态工厂方法；在这种模式中，专门定义一个类根据其传进来的参数来创建其他类的实例，而这些其他类往往继承于同一个超类；
工厂方法模式(Factory Method Pattern): 在工厂方法模式中，定义了一个用来创建产品对象的工厂父类接口，而各个不同的具体子类工厂则负责生产具体的对象。其目的在于将产品的实例化推迟到工厂子类中完成，解除类客户和产品的高耦合；
抽象工厂模式(Abstract Factory Pattern)：提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。这个看起来比较抽象，在后面的笔记中会有更加深入的探讨；

1: 简单工厂模式

在简单模式产生之前，客户和产品是紧密相连的。也就是说，一个客户如果需要某件产品，他必须知道怎么去制作这件产品。看下面的代码：

```
Dick很喜欢开车，因此为了有更多的好车，他必须自己亲自动手去造车。
public class Dick {
    public static void main(String[] args) {
            Car1 car1 = new Car1();
            Car2 car2 = new Car2();
    }
}
public class Car1 {
    public Car1() {
        System.out.println("Dick is producing car1...");
    }
}
public class Car2 {
    public Car2() {
        System.out.println("Dick is producing car2...");
    }
}
但是这样很累，Dick不仅要去自己冶炼金属，还要学会大量的机械知识，Dick觉得很烦。终于有一天，Dick发现了一个好东西，他发现他周边有一家工厂是专门用来制作车的，你只要告诉你需要什么车，工厂就直接会把车生产好给你。于是......

public class Dick {
    public static void main(String[] args) {
            Factory factory = new Factory();
            Car car1 = factory.createCar(1);
            Car car2 = factoty.createCar(2);
    }
}
public class Car {
    public Car() {}
}
public class Car1 extends Car {
    public Car1() {
        System.out.println("The factory is producing car1...");
    }
}
public class Car2 extends Car {
    public Car2() {
        System.out.println("The factory is producing car2...");
    }
}
public class Factory() {
    public Car createCar(int type) {
        switch (type) {
            case 1:
                return new Car1();
                break;
            case 2:
                return new Car2();
                break;
            default:
                break;
        }
        return null;
    }
}
这下Dick省心了，他只需要把自己想要的车型号告诉工厂，他就会得到自己想要的车，而不需要去学习那些什么乱七八糟的知识了。
```

从上面的代码来看，从客户和产品的高度耦合到利用工厂这个中间媒介来创建对象，改变还是比较彻底的，我们不仅仅需要一个工厂类，而且还需要一个产品的超类，其他所有的具体产品必须扩展(extends)这个超类。所以从客户产品高度耦合到第一次工业革命(简单工厂模式)的三个要素：

- 工厂类：这是这次工业革命的绝对核心，它用来创建客户所需要的对象，降低客户和对象的高度耦合关系；
- 抽象产品类： 这个是工厂类用来看家的绝密法宝，工厂所能生产的产品必须实现这个类；
- 具体产品类： 这个就是实实在在工厂生产出来交给客户的产品了，它必须实现抽象产品类；

> 注意：我们这里说的“实现”，并不是说那些超类一定会是类，而不是接口。事实上这里的“实现”只是代表一种泛泛的说法。

变身过程：
来分析一下简单工厂模式，当工厂通过市场调查发现客户可能不再满足于这几种车型。对于客户而言，他们只需要将自己最新想需要的车型告诉工厂就行，他们并不在乎工厂是怎么样制造的；但是对于工厂来说就惨了，客户每需要一种新车型，他们都必须去重新**修改代码**，或重写或添加很多的case语句，这很明显违背了我们 `对扩展开放，对修改关闭的设计原则。` 工厂急需一种新的方式来解放他们的程序员。

2: 工厂方法模式

在上面我们知道，简单工厂模式只有一个工厂，它负责生产所有的汽车类型。如果客户需要一种新的汽车类型，就必须在工厂内部加一条生产流水线，当然这对于实际上的工厂来说，这很合理。但如果从程序员角度看待的话，意味着你要修改之前的代码以适合这种新的需求。
因此出现了一种新的方法－－工厂方法模式，在这种方法下，抽象产品类和实际产品类并不需要改变，如果你需要新的产品，继续扩展抽象产品类就好了；但是对于工厂类而言，我们也做和产品同样的处理，抽象出来一个抽象工厂类，所有其他的工厂必须实现这个抽象工厂类。这是第二次工业革命。。。
因此，工厂方法模式的组成：

- 抽象产品类： 这个是工厂类用来看家的绝密法宝，工厂所能生产的产品必须实现这个类；
- 具体产品类： 这个就是实实在在工厂生产出来交给客户的产品了，它必须实现抽象产品类；
- 抽象工厂类：这个是工厂方法模式的核心，它是一个接口或者超类，所有具体过程必须实现这个抽象工厂类；
- 具体工厂类：这个才是真正生产产品的工厂，它根据客户的需要生产不同的产品；

看上面的例子的工厂方法模式版本：

```
John是工厂的经理，由于客户的需求日新月异，频繁修改代码显然不符合OO原则，因此他决定将工厂抽象出来，然后建立不同的小工厂，然后由这些小工厂来创建对象。
//首先是产品类，如果有新产品加进来，继续实现Car类就好，无需修改之前代码
public class Car {
    public Car() {}
}
public class Car1 extends Car {
    public Car1() {
        System.out.println("The factory is producing car1...");
    }
}
public class Car2 extends Car {
    public Car2() {
        System.out.println("The factory is producing car2...");
    }
}

//接下来是工厂超类
public class Factory {
    public abstract Car createCar();
}
//对于不同的工厂，生产不同的车
public FactoryCar1 extends Factory {
    public Car createCar() {
        return new Car1();
    }
}
public FactoryCar2 extends Factory {
    public Car createCar() {
        return new Car2();
    }
}

//接下来是客户的需求
public class Dick {
    public static void main(String[] args) {
            Factory factory1 = new FactoryCar1();
            Car car1 = factoryCar1.createCar();

            Factory factory2 = new FactoryCar2();
            Car car2 = factoty.createCar();
    }
}
这个和简单工厂模式不同了，在简单工厂模式中，只存在一个工厂，我们为这种工厂传入参数，然后由这个工厂创建对象。在这里，我们对于不同的需求，利用不同的工厂来创建。
```

变身过程：
来分析一下工厂方法模式，的确工厂方法模式已经很好的将客户和产品的耦合程度降到很低。但是注意到在上面的例子中，我们只出现了一种产品超类 Car。如果当客户的需求进一步加大，他不仅仅是需要一辆车，他还需要这辆车由空调、GPS导航仪以及一系列其他设施。而且对于不同的车型，这些设施不尽相同。那么工厂方法模式就捉襟见肘了。这里的车的例子很形象，因为这个不断精细化的过程可以看作对车的一次又一次的扩展，不断的去构建这辆车。因而进入在第三次工业革命－－“分工合作时代”。

3: 抽象工厂模式

在讲抽象工厂模式之前，先讲两个概念：

- 产品等级结构： 产品等级结构其实描述的就是继承关系。比如一个抽象类是汽车，那么有奔驰，宝马，大众一系列具体的品牌，这些品牌都从汽车这个概念扩展开来。因此叫做产品等级结构；
- 产品族：在抽象工厂模式中，由同一家工厂生产的产品就叫做产品族。这些产品从属于不同的产品等级结构。比如小米公司，它既生产手机，从属于手机产品等级；又生产路由器，从属于路由器产品等级结构。等等一系列的产品，这构成一个产品族。

这两个概念揭示了抽象工厂模式和工厂方法模式的重大区别：

- 工厂方法模式注重一个产品等级结构，而抽象工厂模式组合多个产品等级结构；
- 抽象工厂模式高度抽象且具有一般性；

抽象工厂模式组成：

- 抽象产品类(多个)： 这个是工厂类用来看家的绝密法宝，工厂所能生产的产品必须实现这个类；
- 具体产品类： 这个就是实实在在工厂生产出来交给客户的产品了，它必须实现抽象产品类；
- 抽象工厂类：这个是工厂方法模式的核心，它是一个接口或者超类，所有具体过程必须实现这个抽象工厂类；
- 具体工厂类：这个才是真正生产产品的工厂，它根据客户的需要生产不同的产品；

继续看上面代码的深入：

```
//汽车，引擎和空调
public interface Engine {
}
public Engine1 implements Engine {
    public Engine1() {
        System.out.println("Producing engine 1...");
    }
}
public Engine2 implements Engine {
    public Engine2() {
        System.out.println("Producing engine 2...");
    }
}
public interface Air {
}
public Air1 implements Air {
    public Air1() {
        System.out.println("Producing Air 1...");
    }
}
public Air2 implements Air {
    public Air2() {
        System.out.println("Producing Air 2...");
    }
}
public class Car {
    public Car() {}
}
public class Car1 extends Car {
    public Car1() {
        System.out.println("The factory is producing car1...");
    }
}
public class Car2 extends Car {
    public Car2() {
        System.out.println("The factory is producing car2...");
    }
}
//接下来是工厂超类
public class Factory {
    public abstract Car createCar();
    public abstract Engine createEngine();
    public abstract Air createAir();
}
//对于不同的工厂，生产不同的车
public FactoryCar1 extends Factory {
    public Car createCar() {
        createEngine();
        createAir();
        return new Car1();
    }
    public Engine createEngine() {
        return new Engine1();
    }
    public Air createAir() {
        return new Air1();
    }
}
public FactoryCar2 extends Factory {
    public Car createCar() {
        createEngine();
        createAir();
        return new Car2();
    }
    public Engine createEngine() {
        return new Engine2();
    }
    public Air createAir() {
        return new Air2();
    }
}

//接下来是客户的需求
public class Dick {
    public static void main(String[] args) {
            Factory factory1 = new FactoryCar1();
            Car car1 = factoryCar1.createCar();

            Factory factory2 = new FactoryCar2();
            Car car2 = factoty.createCar();
    }
}
在这个例子中，有两个具体工厂，即工厂1和工厂2.对于这两个工厂，他们不仅生产汽车，而且还顺便生产某种汽车带有的配件，比如引擎和空调。也就是说在这里面存在两个产品族－－（工厂1产品族和工厂2产品族）和三个产品等级结构－－（车，引擎，空调）。
```