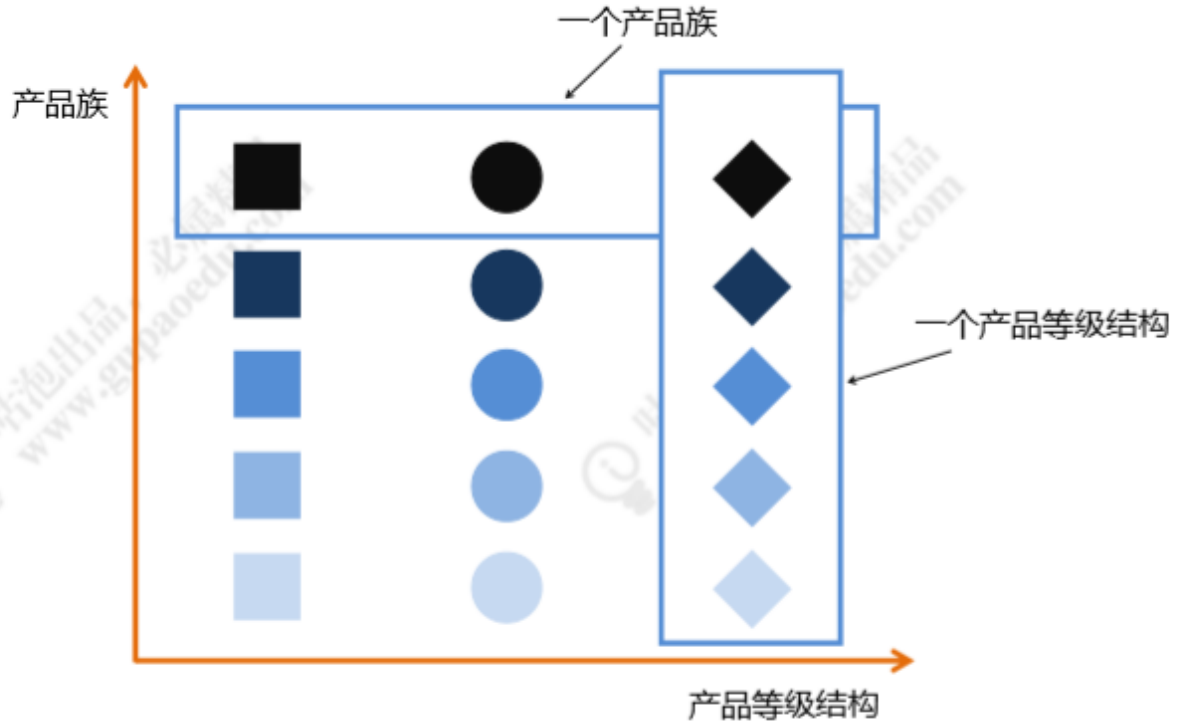
讲解抽象工厂之前，我们要了解两个概念产品等级结构和产品族，看下面的图



从上图中看出有正方形，圆形和菱形三种图形，相同颜色深浅的就代表同一个产品族，

相同形状的代表同一个产品等级结构。同样可以从生活中来举例，比如，美的电器生产

多种家用电器。那么上图中，颜色最深的正方形就代表美的洗衣机、颜色最深的圆形代

表美的空调、颜色最深的菱形代表美的热水器，颜色最深的一排都属于美的品牌，都是

美的电器这个产品族。再看最右侧的菱形，颜色最深的我们

指定了代表美的热水器，那

么第二排颜色稍微浅一点的菱形，代表海信的热水器。同理，同一产品结构下还有格力

热水器，格力空调，格力洗衣机。

1.定义接口

```
public interface Fruit {
    void getFruit();
}
```

2.定义抽象类

```
public abstract class Apple implements Fruit{
    public abstract void getFruit();
}

```

```
public abstract class Banana implements Fruit{
    public abstract void getFruit();
}
```

3.水果分为北方水果和南方水果。就相当于产品等级结构

```
public class NorthApple extends Apple {
    @Override
    public void get() {
        System.out.println("采集北方苹果");
```

```
    }
}
```

```
public class SouthApple extends Apple {
    @Override
    public void get() {
        System.out.println("采集南方苹果");
    }
}
```

```
public class NouthBanana extends Banana {
    @Override
    public void getFruit() {
        System.out.println("采集北方香蕉");
    }
}
```

```
public class SouthBanana extends Banana {
    @Override
    public void get() {
        System.out.println("采集南方香蕉");
    }
}
```

4.定义工厂接口

```
public interface FruitFactory {
    Fruit getApple();
    Fruit getBanana();
   // Fruit getLemon();
```

```
}

```

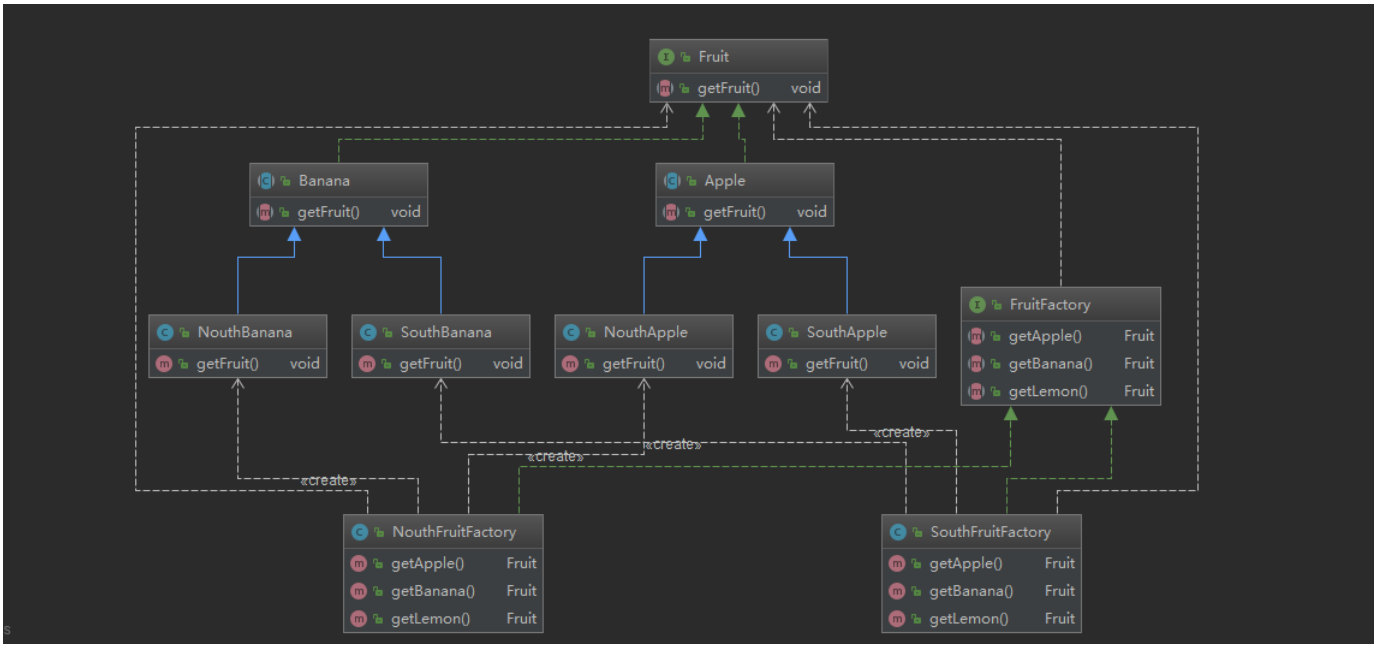
```
public class NouthFruitFactory implements FruitFactory{
    @Override
    public Fruit getApple() {
        return new NouthApple();
    }

    @Override
    public Fruit getBanana() {
        return new NouthBanana();
    }

    @Override
    public Fruit getLemon() {
        return new NouthLemon();
    }
}
```

```
public class SouthFruitFactory implements FruitFactory {
    @Override
    public Fruit getApple() {
        return new SouthApple();
    }

    @Override
    public Fruit getBanana() {
        return new SouthBanana();
    }

    @Override
    public Fruit getLemon() {
        return new SouthLemon();
    }
}
```

```
public class Test {
    public static void main(String[] args) throws Exception{
        NorthFruitFactory northFruitFactory = new
NorthFruitFactory();
        Fruit apple = northFruitFactory.getApple();
        Fruit banana = northFruitFactory.getBanana();
        apple.get();
        banana.get();

        SouthFruitFactory southFruitFactory = new
SouthFruitFactory();
        Fruit apple1 = southFruitFactory.getApple();
        Fruit banana1 = southFruitFactory.getBanana();
        apple1.get();
        banana1.get();


    }
}

```

如果想要增加一个柠檬。需要创建一个柠檬类，继承fruit类，然后在创建南方柠檬和北方柠檬，然后在FruitFactory里添加采集柠檬的接口。然后所有实现FruitFactory的类都需要新加一个采集柠檬的功能。违背了开闭原则。而且增加了系统的抽象性和理解难度。这也是抽象工厂的缺点。

类图：


Fruit
getFruit() void
Banana
getFruit() void
Apple
getFruit() void
NouthBanana
getFruit() void
SouthBanana
getFruit() void
NouthApple
getFruit() void
SouthApple
getFruit() void
FruitFactory
getApple() Fruit
getBanana() Fruit
getLemon() Fruit
«create»
«create»
«create»
«create»
NouthFruitFactory
getApple() Fruit
getBanana() Fruit
getLemon() Fruit
SouthFruitFactory
getApple() Fruit
getBanana() Fruit
getLemon() Fruit