第五章——抽象工厂模式

H1

> 本章概况：本章将在工厂模式的基础上，讲述如何利用抽象工厂模式生成"产品族"

一、产品等级和产品族

在正式学习抽象工厂模式前， 需要讲述一下产品族的概念。

H2 书上这样解释：

> 产品等级结构： 产品等级结构即产品的继承结构， 例如一个抽象类是电视机，其子
> 类包括海尔电视机、海信电视机、TCL 电视机， 则抽象电视机与具体品牌的电视机之间构成了一个产品等级结构， 抽象电视机是父类， 而具体品牌的电视机是其子类。
>
> 产品族： 在抽象工厂模式中， 产品族是指由同一个工厂生产的位于不同产品等级
> 结构中的一组产品， 例如海尔电器工厂生产的海尔电视机、海尔冰箱， 海尔电视机位于电视机产品等级结构中， 海尔冰箱位于冰箱产品等级结构中， 海尔电视机、海尔冰箱构成了一个产品族。

图示如下：

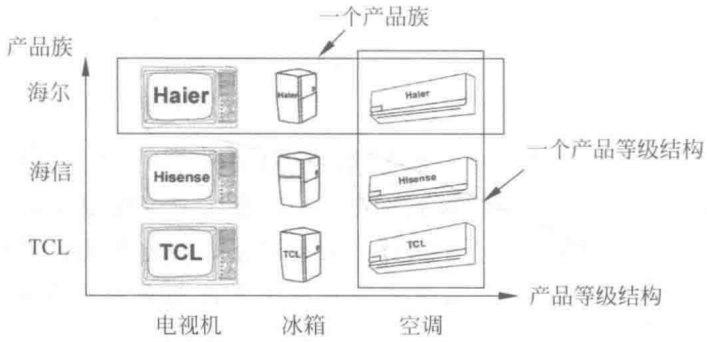


图 5-1　产品族与产品等级结构示意图

二、抽象模式

在前面介绍了产品族概念的基础上，我们正式介绍抽象模式。

H2 定义：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。

解释：

> 抽象模式中有 4 个角色：
>
> - 抽象工厂： 它声明了一组用于创建一族产品的方法， 每一个方法对应一种产品
> - 具体工厂 ： 它实现了在抽象工厂中声明的创建产品的方法，生成一组具体产品， 这些产品构成了一个产品族， 每一个产品都位于某个产品等级结构中。
> - 抽象产品：它为每种产品声明接口， 在抽象产品中声明了产品所具有的业务方法。
> - 具体产品：它定义具体工厂生产的具体产品对象， 实现抽象产品接口中声明的业务方法。

图示如下：

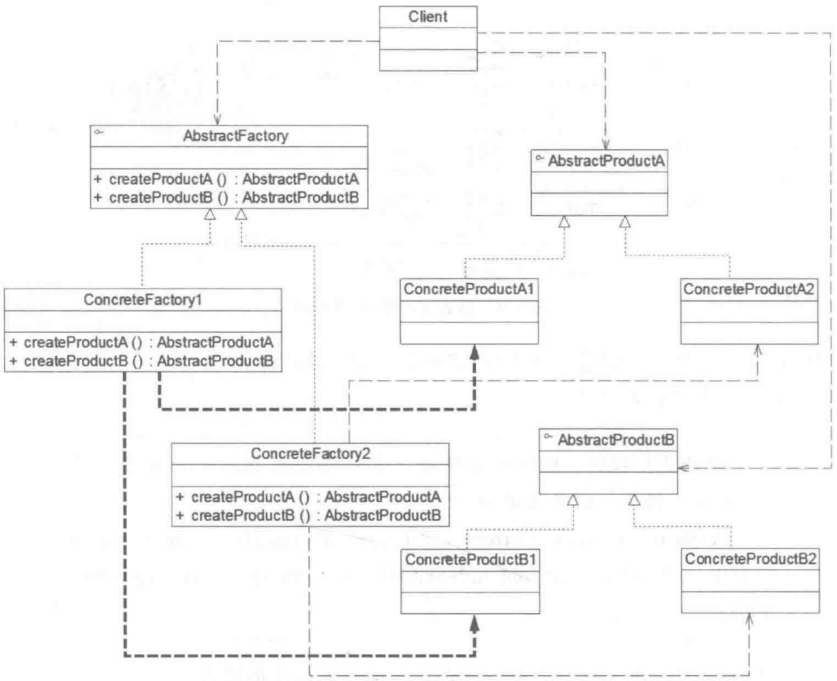


图 5-3　抽象工厂模式结构图

三、抽象工厂代码实现（示例）

1. 抽象皮肤工厂

H2

```
package com.IQIUM.Factory;

import com.IQIUM.Widgets.Button;
import com.IQIUM.Widgets.ComboBox;
import com.IQIUM.Widgets.TextField;

public interface SkinFactory {
    public Button createButton();

    public TextField createTextField();

    public ComboBox createComboBox();
}
```

2. 春天皮肤工厂

```
package com.IQIUM.Factory;

import com.IQIUM.Products.SpringButton;
import com.IQIUM.Products.SpringCombox;
import com.IQIUM.Products.SpringTextField;
```

```java
import com.IQIUM.Widgets.Button;
import com.IQIUM.Widgets.ComboBox;
import com.IQIUM.Widgets.TextField;

public class SpringSkinFactory implements SkinFactory {

    @Override
    public Button createButton() {
        return new SpringButton();
    }

    @Override
    public TextField createTextField() {
        return new SpringTextField();
    }

    @Override
    public ComboBox createComboBox() {
        return new SpringCombox();
    }
}

```

3. 夏天抽象工厂

```java
package com.IQIUM.Factory;

import com.IQIUM.Products.*;
import com.IQIUM.Widgets.Button;
import com.IQIUM.Widgets.ComboBox;
import com.IQIUM.Widgets.TextField;

public class SummerSkinFactory implements SkinFactory {

    @Override
    public Button createButton() {
        return new SummerButton();
    }

    @Override
    public TextField createTextField() {
        return new SummerTextField();
    }

```

```
    @Override
    public ComboBox createComboBox() {
        return new SummerCombox();
    }
}

```

4. 春天的按钮、文本框、组合框

```
package com.IQIUM.Products;

import com.IQIUM.Widgets.Button;

public class SpringButton implements Button {
    @Override
    public void display() {
        System.out.println("显示浅绿色按钮");
    }
}

```

```
package com.IQIUM.Products;

import com.IQIUM.Widgets.TextField;

public class SpringTextField implements TextField {
    @Override
    public void display() {
        System.out.println("显示浅绿色文本框");
    }
}

```

```java
package com.IQIUM.Products;

import com.IQIUM.Widgets.ComboBox;

public class SpringCombox implements ComboBox {
    @Override
    public void display() {
        System.out.println("显示浅绿色组合框");
    }
}

```

5. 夏天按钮、文本框、组合框

```java
package com.IQIUM.Products;

import com.IQIUM.Widgets.Button;

public class SummerButton implements Button {
    @Override
    public void display() {
        System.out.println("显示浅蓝色按钮");
    }
}

```

```java
package com.IQIUM.Products;

import com.IQIUM.Widgets.TextField;

public class SummerTextField implements TextField {
    @Override
    public void display() {
        System.out.println("显示浅蓝色文本框");
    }
}

```

```java
package com.IQIUM.Products;

import com.IQIUM.Widgets.ComboBox;

public class SummerCombox implements ComboBox {
    @Override
    public void display() {
        System.out.println("显示浅蓝色组合框");
    }
}

```

6. 按钮、组合框、文本框的接口

```java
package com.IQIUM.Widgets;

public interface Button {
    public void display();
}

```

```java
package com.IQIUM.Widgets;

public interface ComboBox {
    public void display();
}

```

```java
package com.IQIUM.Widgets;

public interface TextField {
    public void display();
}

```

四、抽象工厂的优缺点

（一）抽象工厂模式优点

抽象工厂模式的优点主要如下：
（1） 抽象工厂模式隔离了具体类的生成， 使得客户端并不需要知道什么被创建。由于
这种隔离， 更换一个具体工厂就变得相对容易， 所有的具体工厂都实现了抽象工厂中定义的那些公共接口， 因此只需改变具体工厂的实例就可以在某种程度上改变整个软件系统的
行为。
（2） 当一个产品族中的多个对象被设计成一起工作时， 它能够保证客户端始终只使用
同一个产品族中的对象。
（3） 增加新的产品族很方便， 无须修改已有系统， 符合开闭原则。

（二）抽象工厂模式缺点

抽象工厂模式的缺点主要如下：
　增加新的产品等级结构麻烦， 需要对原有系统进行较大的修改， 甚至需要修改抽象层代
码， 这显然会带来较大的不便， 违背了开闭原则。

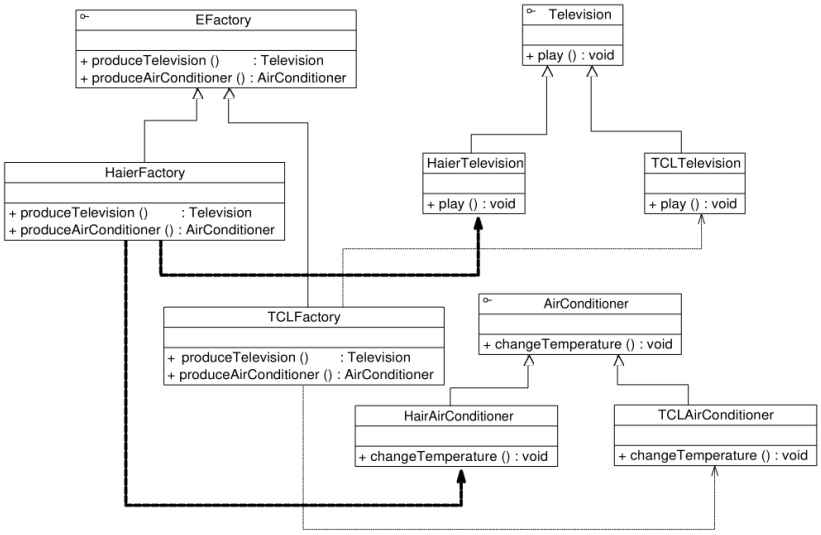
五、课后习题答案

1. D

2. D

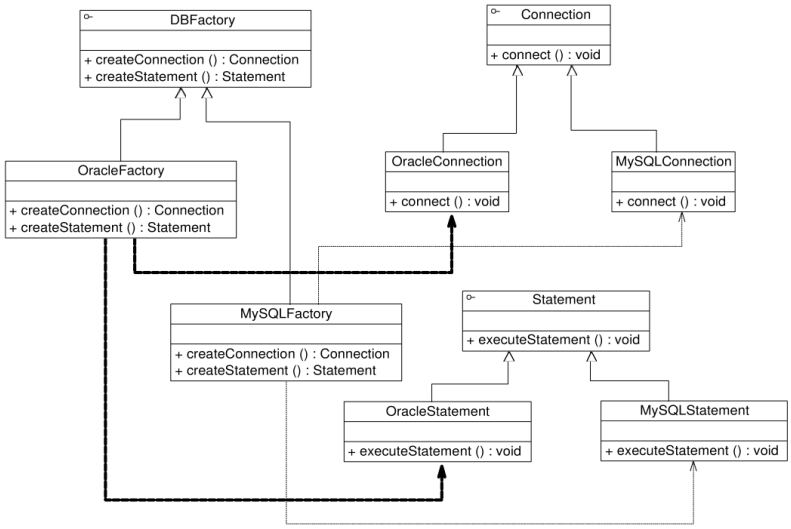
3. A

4. `EFactory` 充当抽象工厂， `HaierFactory` 和 `TCLFactory` 充当具体工厂， `Television` 和 `AirConditioner` 充当抽象产品，`HaierTelevision` 、 `TCLTelevision` 、 `HaierAirConditioner` 和 `TCLAirConditioner` 充当具体产品。



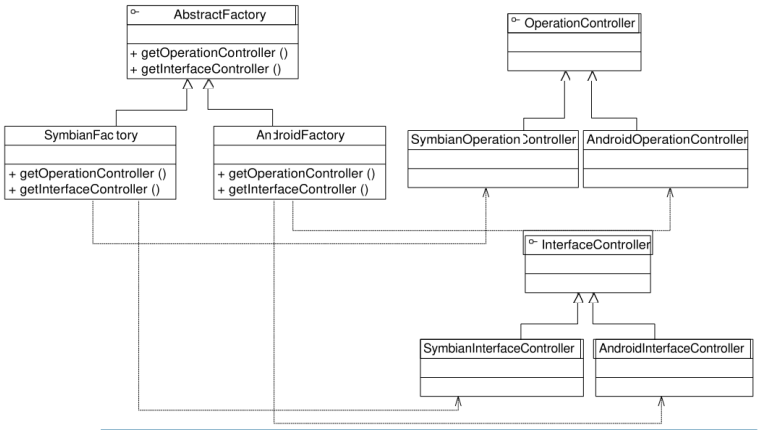
5.

接口 `DBFactory` 充当抽象工厂，其子类 `OracleFactory` 和 `MySQLFactory` 充当具体工厂，接口 `Connection` 和 Statement 充当抽象产品，其子类 `OracleConnection` 、`MySQLConnection` 和 `OracleStatement` 、`MySQLStatement` 充当具体产品。



6. 接口 `AbstractFactory` 充当抽象工厂，其子类 `SymbianFactory` 和 `AndroidFactory` 充当具体工厂；`OperationController` 和 `InterfaceController` 充当抽象产品，其子类 `SymbianOperationController` 、`AndroidOperationController` 、`SymbianInterfaceController` 和 `AbstractFactory` 、

`AndroidInterfaceController` 充当具体产品。



7.

接口 `AbstractFactory` 充当抽象工厂，其子类 `WindowsFactory` 、 `UnixFactory` 和 `LinuxFactory` 充当具体工厂； `Text` 和 `Button` 充当抽象产品，其子类 `WindowsText` 、 `UnixText` 、 `LinuxText` 和 `WindowsButton` 、 `UnixButton` 、 `LinuxButton` 充当具体产品。