

## 简单工厂模式

### 1.1 经典模式

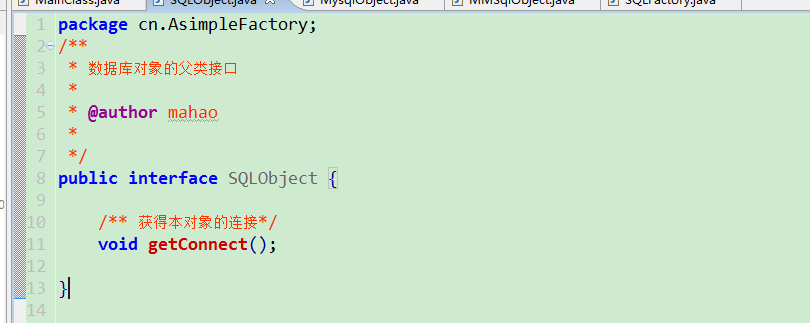
1.说明：将类的创建集中起来，建造一个工厂，用来集中创建实例，被创建的类，通常具有相同的父类。

2.组成： 工厂，抽象父类，具体实现。

3.理解：工厂类将各个子类的创建交由自己处理，用户使用工厂的方法，获取提供的自己需要的实例，用户无需要了解这些类如何创建的和组织的（类似与类的封装）。

4.应用：对于不同数据库的连接，用户通过数据库工厂，获取不同数据库的对象，不必关心这个对象是如何创建的。

1.父类接口



2.实现实例

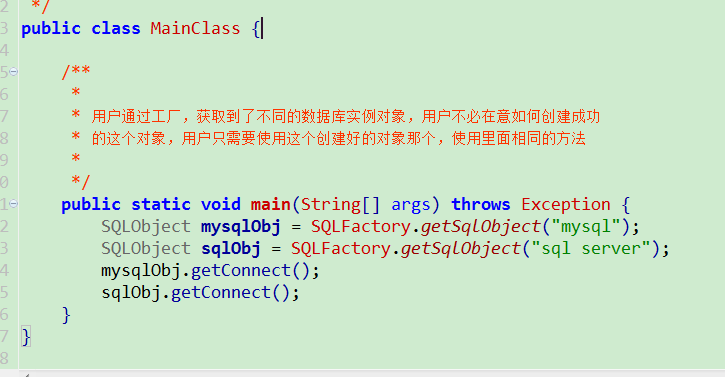




1. 工厂类

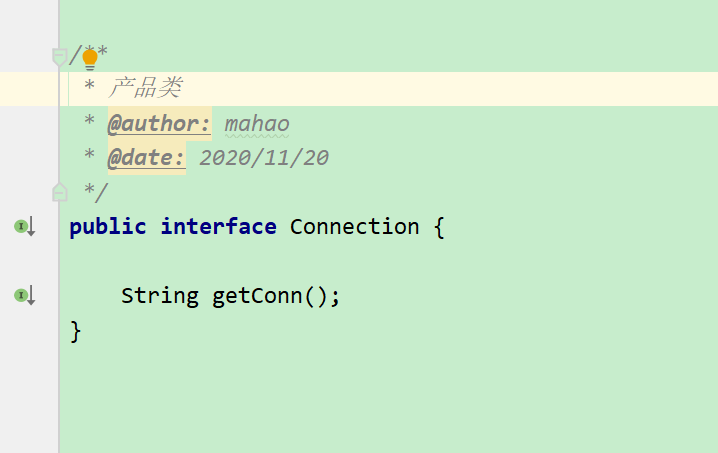


1. 测试说明



### 变形

使用枚举类，继承产品的接口，枚举中的枚举值，则都是具体产品。





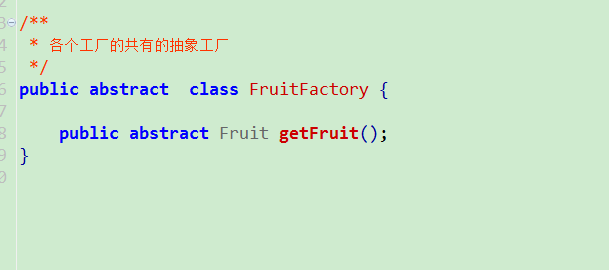
## 2. 工厂方法模式

1.说明：将工厂设定为一个抽象工厂，各个实例实现这个接口，称为具体工厂，多态特性去取的各自的实例对象，所以这个模式又被称为多态工厂模式。工厂方法模式之所以有一个别名叫多态性工厂模式是因为具体工厂类都有共同的接口，或者有共同的抽象父类

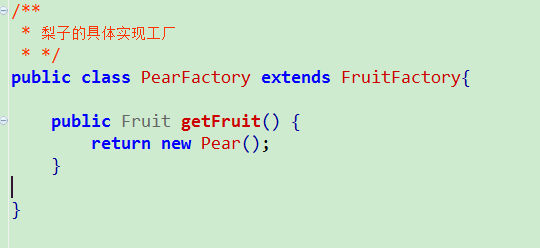
2.组成：抽象工厂，具体工厂，

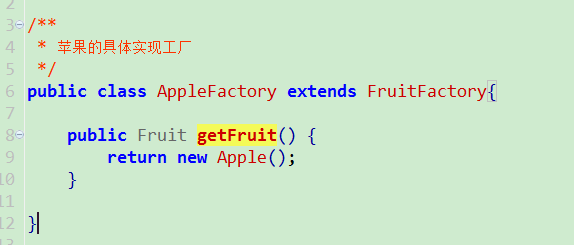
3.理解：这个模式，理解多态。实例的创建分配到子类当中，一个子类负责一个实例的创建，工厂提供的是共有的方法名称。好处就是在修改产品时，不需要改变工厂，只需要增加子类工厂就行，利于扩展。

抽象工厂

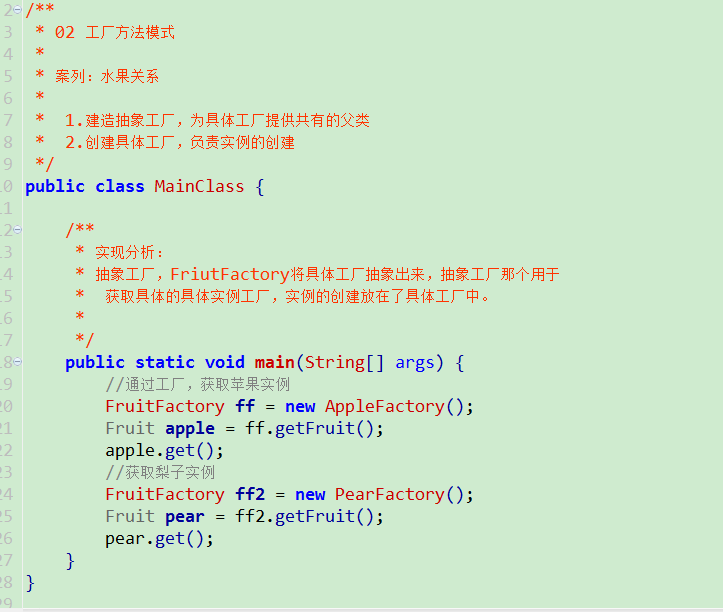


2.实现了抽象工厂的，pear和apple的具体工厂





1. 方法测试



## 3. 抽象工厂模式

说明：工厂方法模式对象对应的是产品的结构，显示是垂直的；抽象工厂方法模式，对象那个的是产品的种类，是纵向的。

Fruit

SouthPear

NorthPear

SouthApple

NorthApple

Apple

Pear

获取的是具有相同特征的对象，是纵向的

Northfactory

SouthFactory

1. 抽象工厂方法模式

2.工厂方法模式

垂直型的获取所需要的对象

用于获取梨子对象

用于获取apple对象

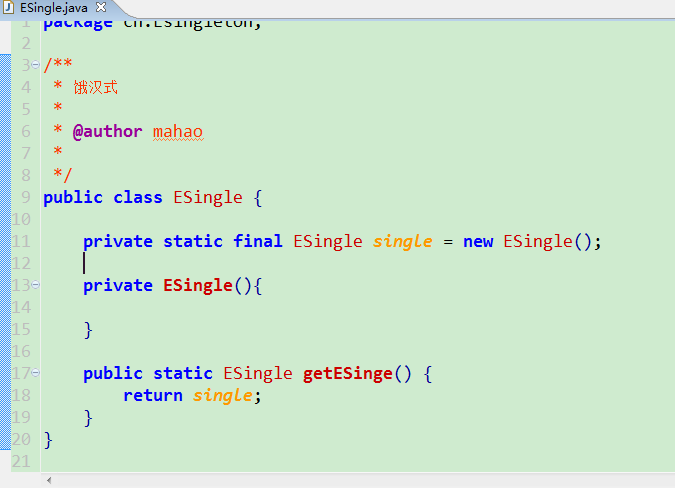
PearFactory

AppleFactory

FruitFactory

## 单例模式

1. 饿汉式



1. 懒汉式



1. 懒汉式的双重检查

## 5. 原型模式 (克隆模式)

1.说明： 这个模式是对原型实例的复制，在堆内存中，生成一个和原型一样的内存空间，不仅仅与原型对象具有相同的结构，还与原型对象具有相同的

值。

2.使用：当创建对象时，我们不仅是希望被创建的对象继承基类的结构，还希望得到基类的数据。

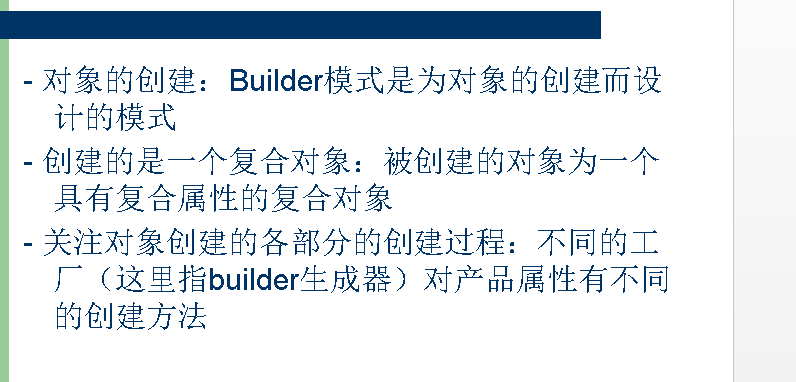




## 6. 建造者模式

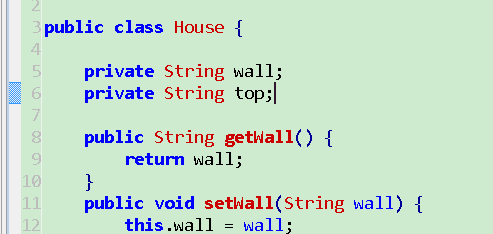
1.说明:构建者模式是一种对象创建者模式，用来隐藏复合对象模式的创建过程，将对象的创建过程抽象，通过子类继承和重载的方式是实现

2.

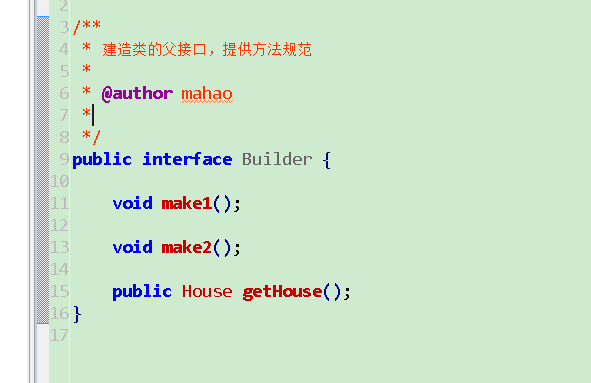


3.理解：建造者模式是用来创建对象使用的，把对象的创建隐藏起来，用构建器builder，特意去创建某种类型对象。这些构建器有一个抽象的父类，这些父类提供通用的对象那个创建方法，让子类去重构这些方法，提供构建器。Director作用是执行不同构建器，使用它们的方法。如果构建器是工人团体，那director则是设计者，负责调用工人团体，实现创建对象。

1. 要被动态创建的对象

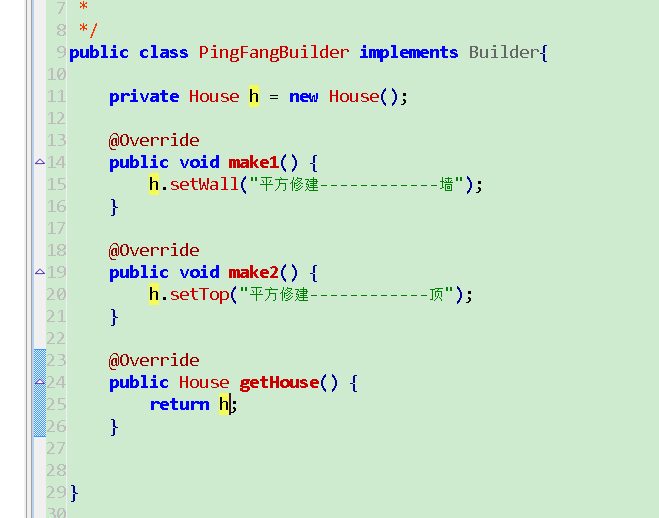


1. 构建器的父类接口



1. 子类具体的构建器

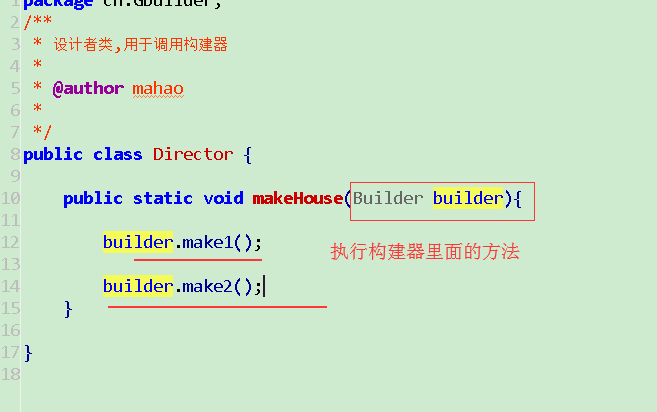
//1.平方房屋构建器



// 2.公寓构建器

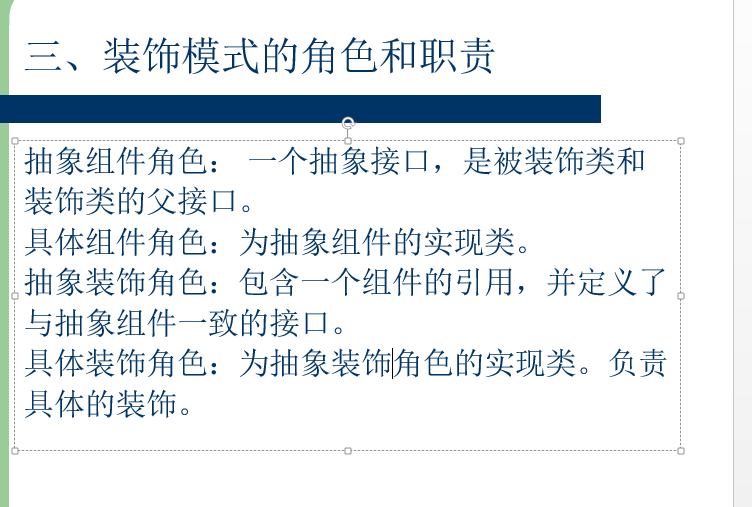


1. 执行器



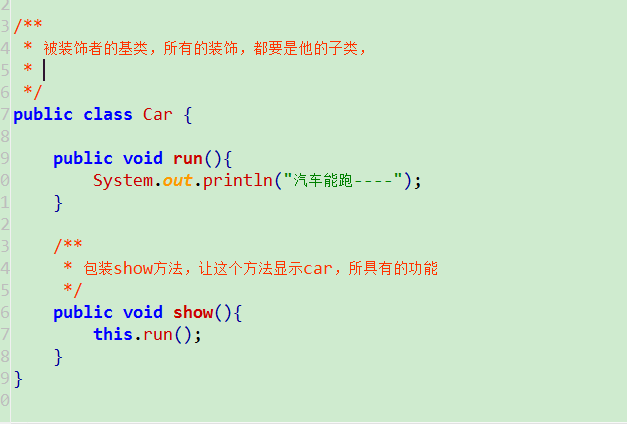
1. 运行
2. 

## 装饰模式（包装模式）

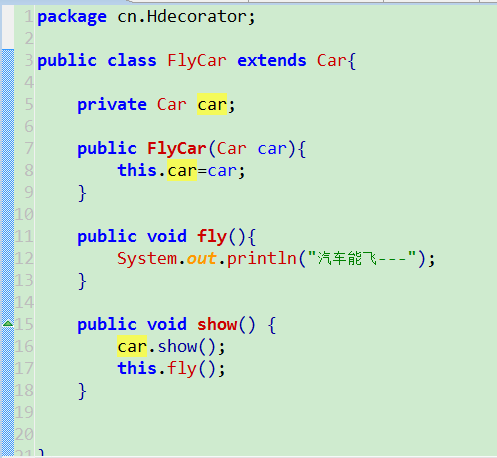
1. 说明：包装模式是一种方法的增强，对类的某一个方法在原有基础上进行增强。是继承方案的另外一种解决方案。
2. 角色：在装饰模式中，一个抽象接口，是装饰类和被装饰类的共同父接口；
3. 

具体实现：

1.共同的父类



1. 装饰类

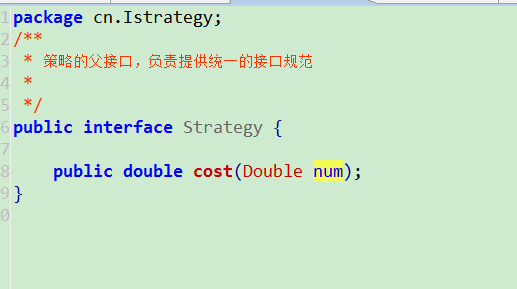


1. 测试

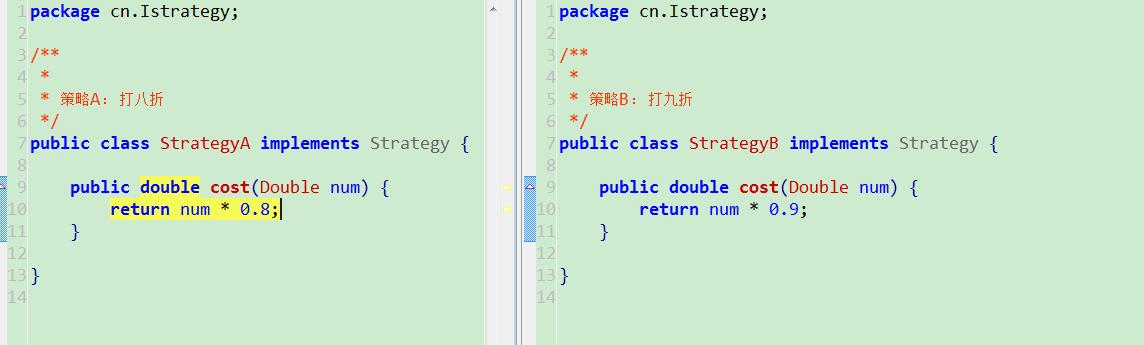


## 策略模式

1. 说明： 策略模式类似于包装模式，包装模式是将父类传入子类，子类调用父类方法，同时进行增强。策略模式是调用策略类的方法，传入子类，调用子类的方法。
2. 策略父接口

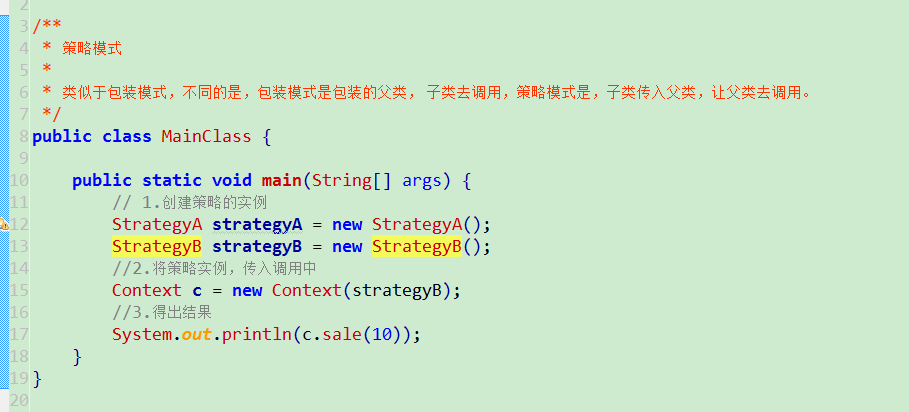


1. 两个子类实现





1. 运行



## 享元模式

1. 说明：是构造模型之一，与其他类似对象，共享内存数据，减小内存消耗。
2. 线程池，连接池技术的开发模式，将连接放入容器，使用时从容器中获取。

实现：

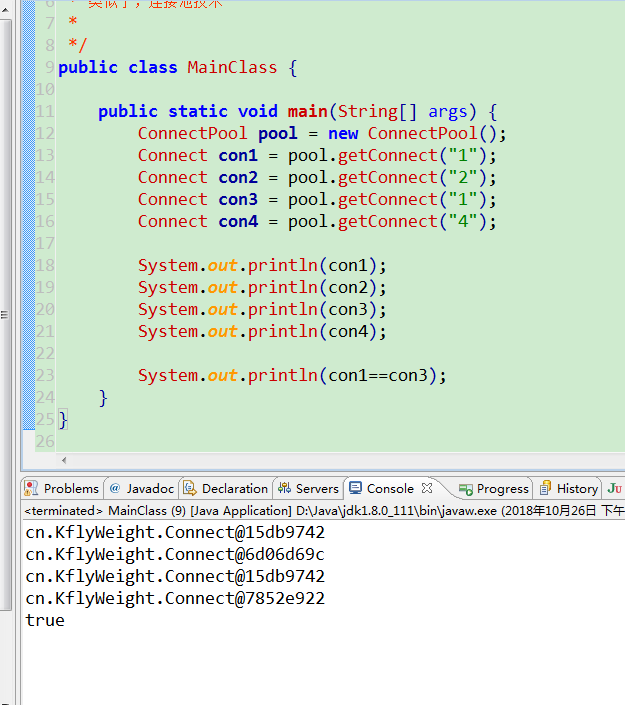
1. 单个连接对象



1. 数据库的连接池



1. 测试



## 代理模式

\* JDK动态代理二次许学习理解；

\*

\* 动态代理实现MySubject的增强：

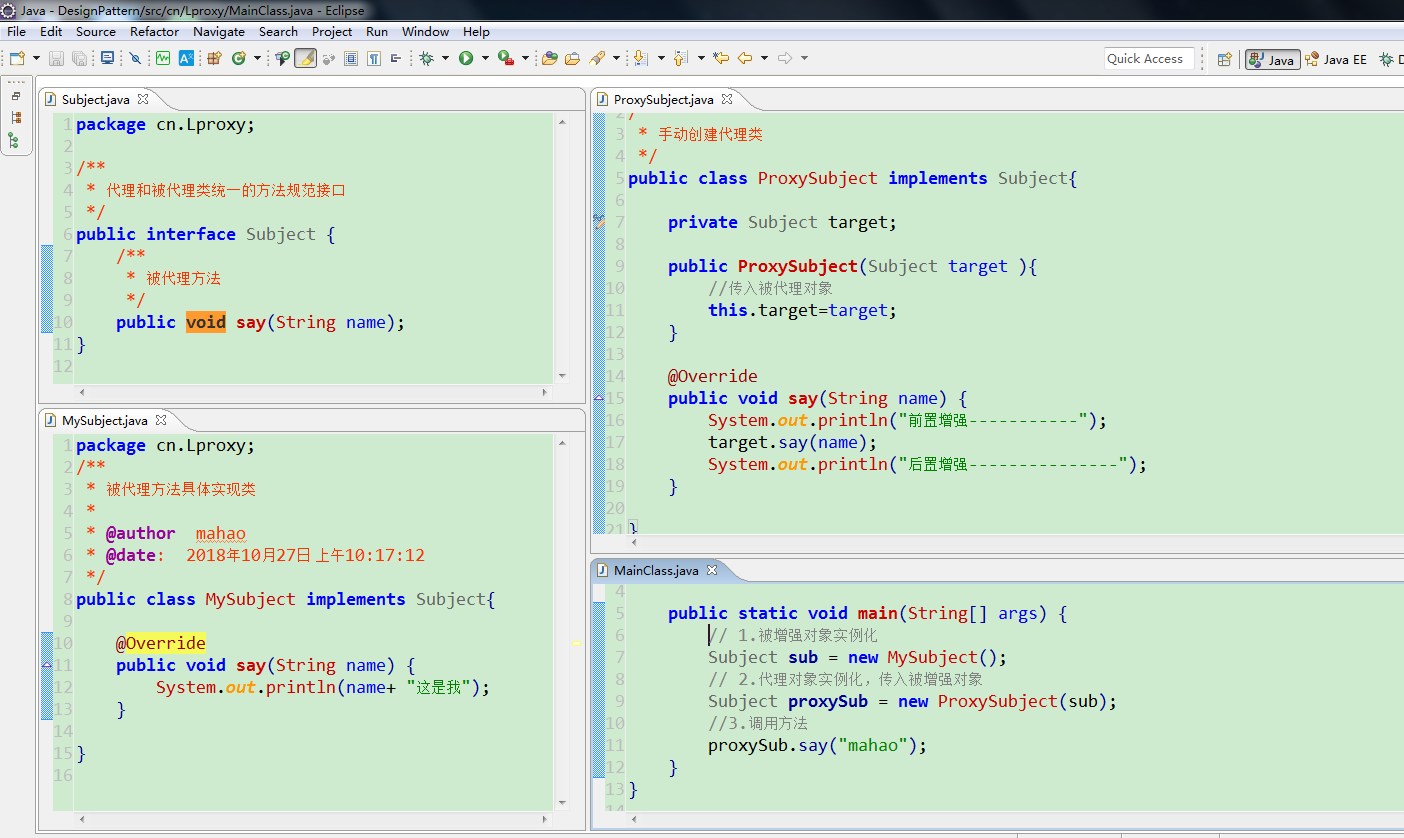
\* jdk的动态代理是以前需要手动用代理模式书写一个代理类，现在是jdk自动

\* 完成代理类的创建，jdk创建代理好的字节码文件，并且将实例对象通过Proxy.newProxyInstance（）

\* 创建出来，对每个方法，都会在 InvocationHandler()的未实现方法中触发，所以对需要增强的方法

\* 可以进行增强。

1. 手动代理



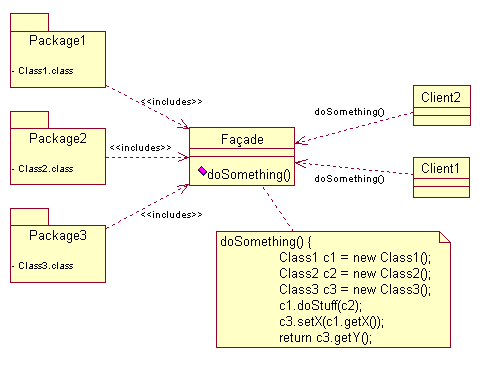
1. Jdk自动代理





## 外观模式

1. 说明：在一组类似的系统中，提供一个一致的简单的界面，这个简单的页面就被称为facaed；



1. 组成：

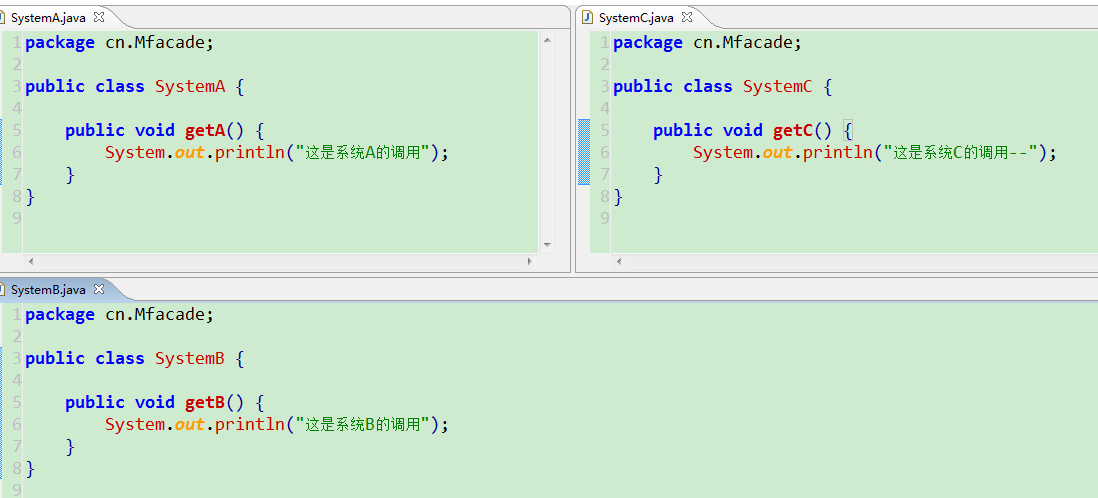
façade：为调用方定义简单的调用接口

clients：调用者，通过Facade接口调用提供某功能的内部类群。

Packages：功能提供者。指提供功能的类群（模块或者子系统）

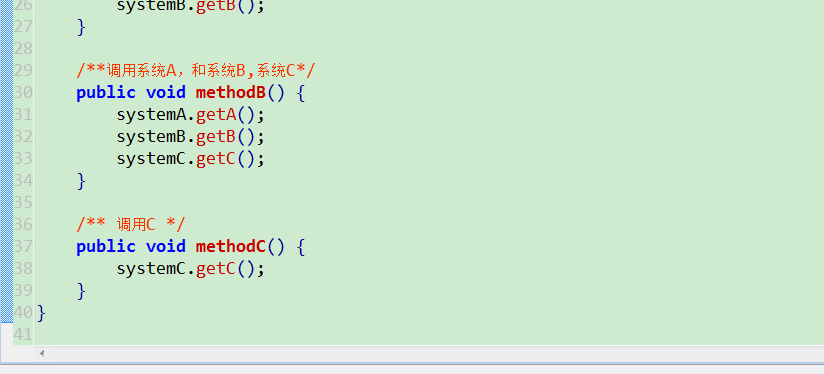
1. 代码样例

系统类群



Façade接口：



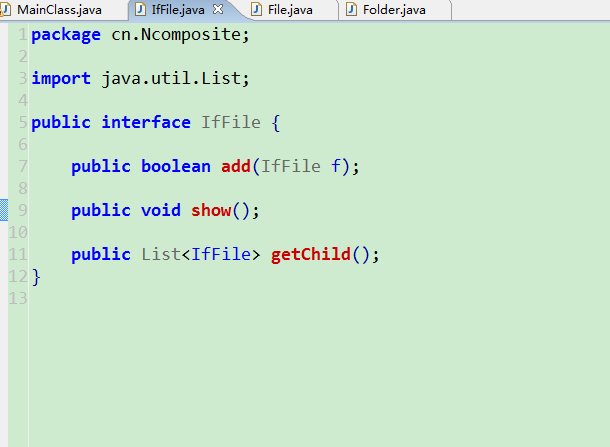


MainClass



## 组合模式

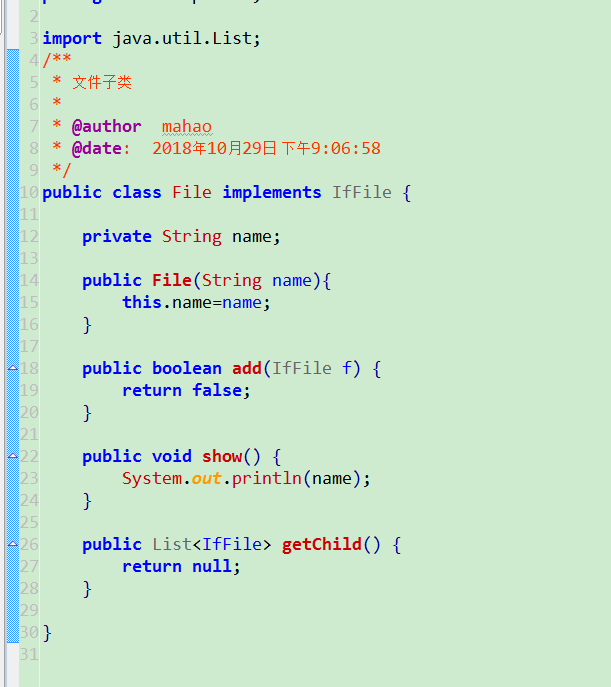
1. 说明，类似于文件的结构系统，树形目录显示，从一个对象，可以找到所有对象
2. 父类接口



1. 目录子类



1. 文件子类



**import** java.util.List;

/\*\*

\* 文件系统目录测试

\*

\* **@author** mahao

\* **@date**: 2018年10月29日 下午8:21:19

\*/

**public** **class** MainClass {

**public** **static** **void** **main**(String[] args) {

/\*

\* C

\* |--a

\* |--d.txt

\* |--e

\* |--f.txt

\* |--g.txt

\* |--b

\* |--h

\* |--i.txt

\* |--j

\*/

Folder **C** = **new** Folder("C");

Folder **a** = **new** Folder("a");

Folder **b** = **new** Folder("b");

Folder **e** = **new** Folder("e");

Folder **h** = **new** Folder("h");

Folder **j** = **new** Folder("j");

File **d** = **new** File("d");

File **f** = **new** File("f");

File **g** = **new** File("g");

File **i** = **new** File("i");

e.add(f);

e.add(g);

a.add(d);

a.add(e);

C.add(a);

h.add(i);

b.add(h);

b.add(j);

C.add(b);

List<IfFile> **child** = C.getChild();

//for (IfFile ifFile : child) {

*printIfFile*(C,"|--");

//}

}

**private** **static** **void** **printIfFile**(IfFile ifFile,String str) {

str += "--";

System.***out***.print(str);ifFile.show();

**if**(ifFile **instanceof** Folder){

**for** (IfFile **f** : ifFile.getChild()) {

*printIfFile*(f,str);

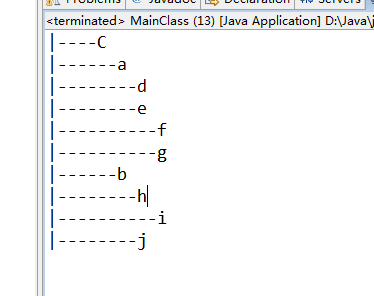
}

}

}

}

结果显示



## 桥接模式

说明：

\* 桥接模式

\* 桥接模式，类似于策略模式，

\* 1.策略模式是，策略的引用有多个子类，调用的话，

\* 只有一个类去调用，是调用类的方法去调用策略引用，调用类不变。

\* 2.桥接模式是，桥接的引用也有多个子类，调用的话，

\* 引用的话，也是有多个子类的，在引用处，可以有多个子类

\*

\* 策略模式注重是方法的调用，包装了不同算法的实现

\* 桥接模式注重的是结构的构造，扩展了两个类之间的实现。

\*

\* 案例是将汽车有安装引擎的操作，由于引擎不同，将引擎和汽车做成

\* 桥接模式，汽车中含有引擎的引用。引擎做成接口，让子类实现不同的引擎

\* 规格。汽车哪儿，则去调用子类引擎。由于汽车规格也不近相同，所以汽车也做成了

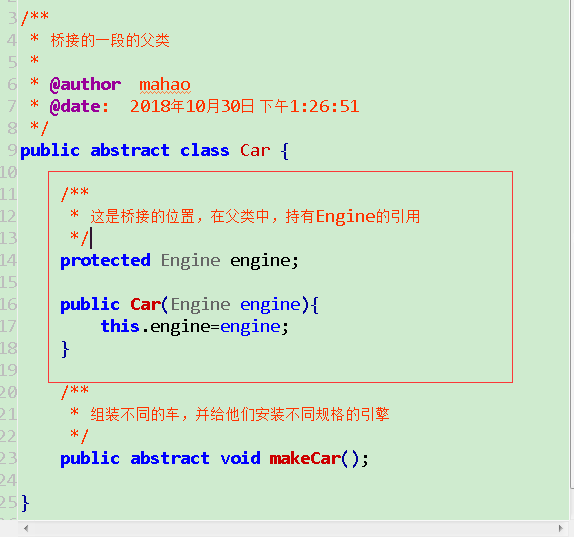
\* 抽象模式，不同汽车也有不同的规格。比如有吉普车，去安装200的引擎。也有公交车

\* 安装300c的引擎。而策略模式，只是利用实现算法的封装，只能是策略算法能扩展，引用哪儿调用

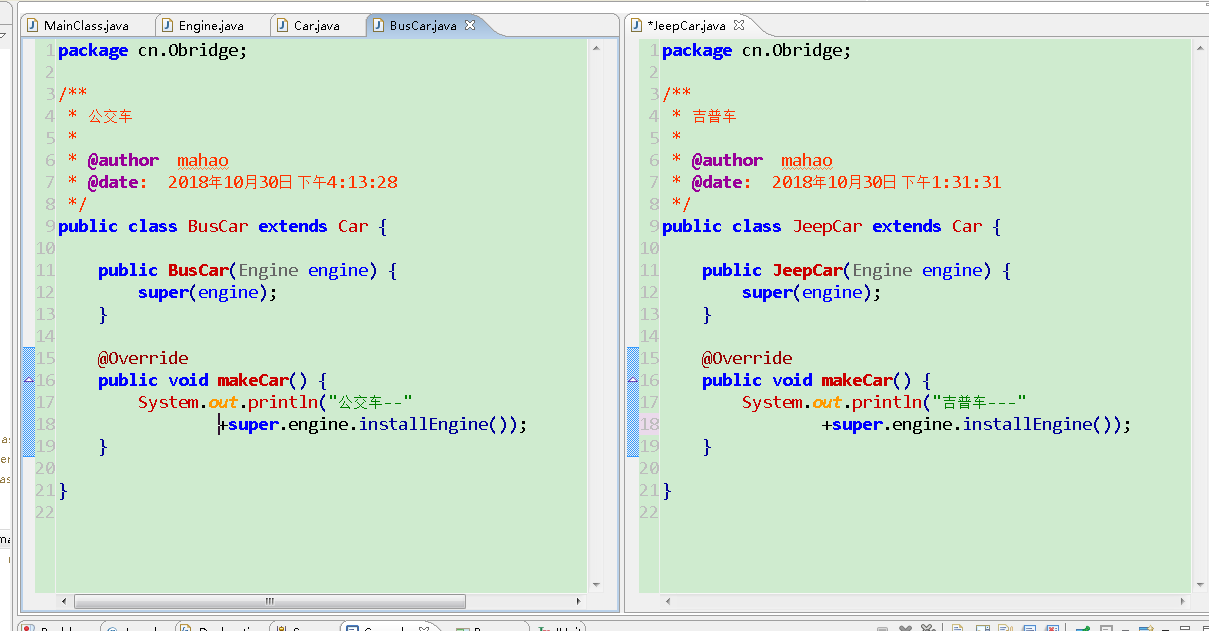
\* 桥接这个是两面都有扩展，但有个桥，做着关联。（比较两种模式的图结构）。

1. 案列：

汽车的抽象父类，会有不同规格的汽车，所有做成抽象了



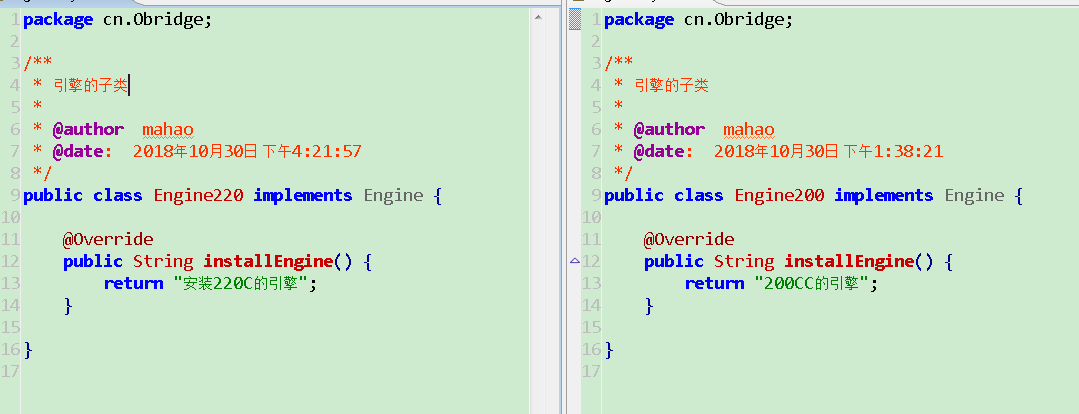
两个汽车实现类：



另一个类，引擎类



两个引擎类的实现

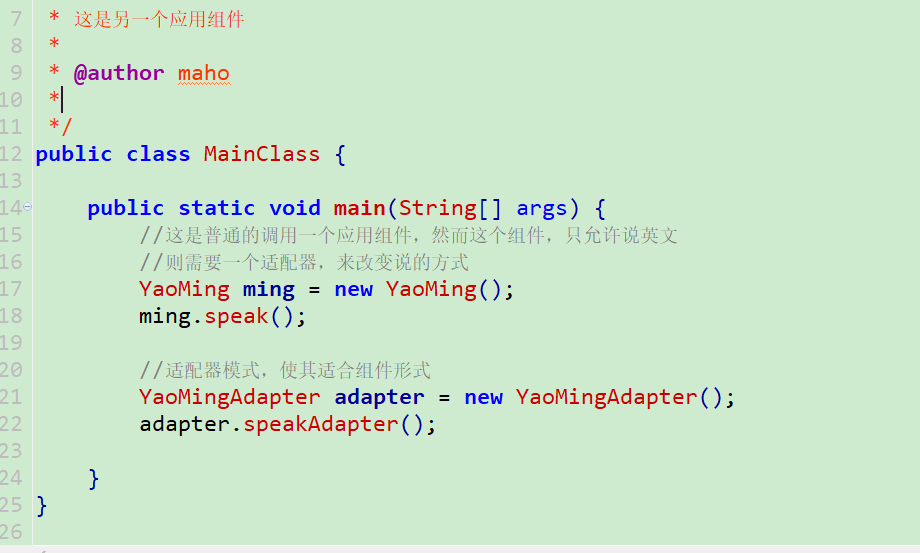


测试样例

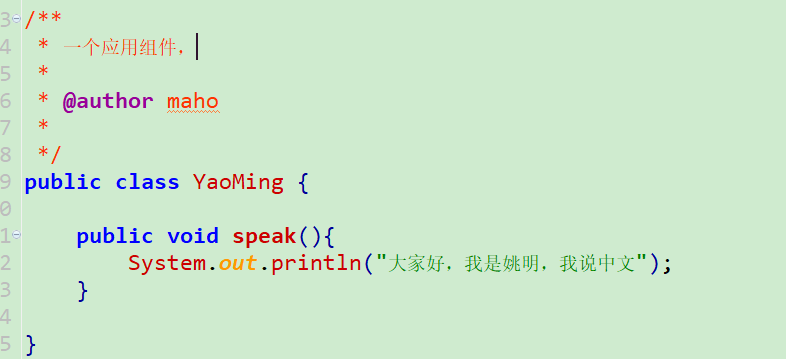


## 适配器模式

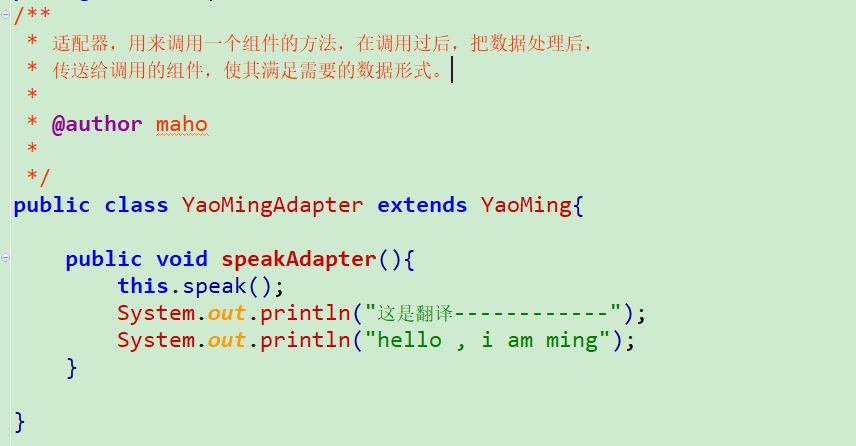
1. 说明：适配器是为了之前两个不同的组件，或者有差异的组件，提供一个中间件，让两个类适合调用。
2. 组成：组件一，被调用组件，组件适配器
3. 案列：翻译
4. 组件一：

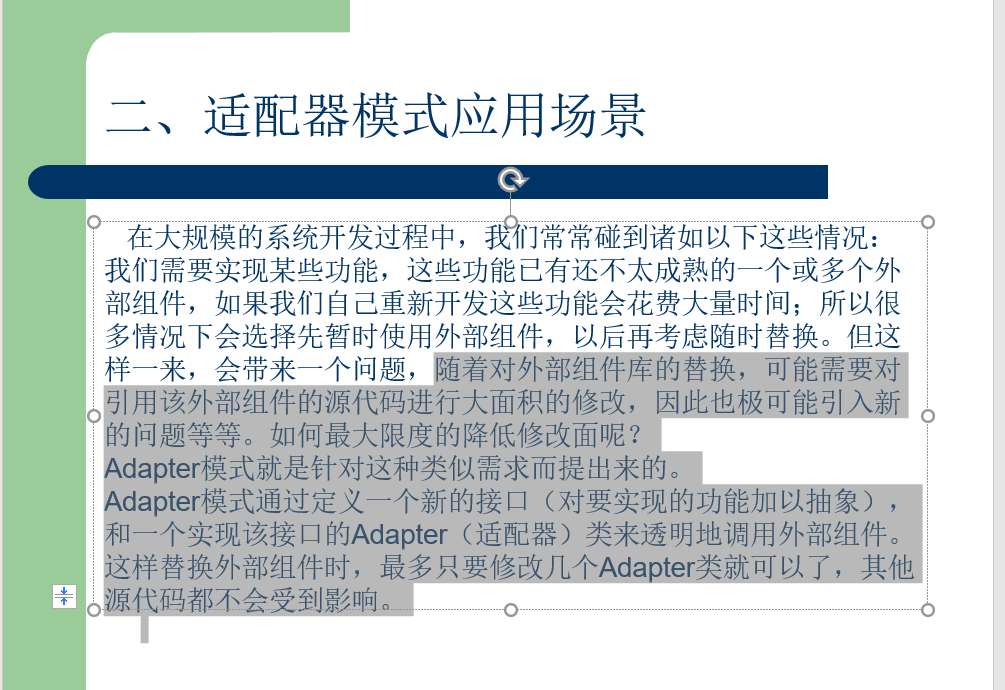


1. 被调用组件：



1. 适配器：





## 解释器模式

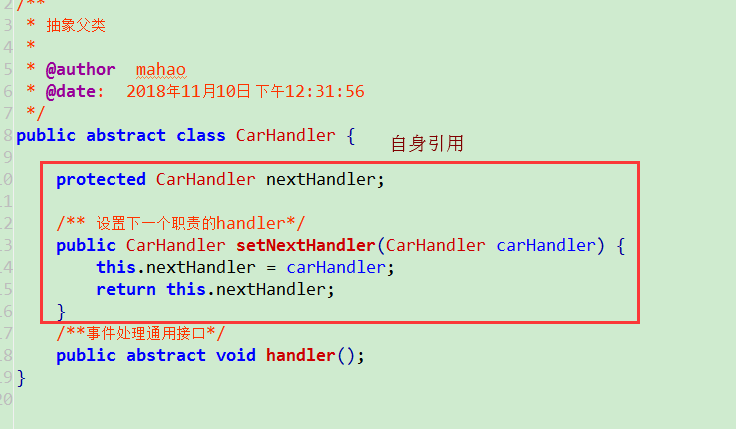
## 中介者模式

## 责任链模式

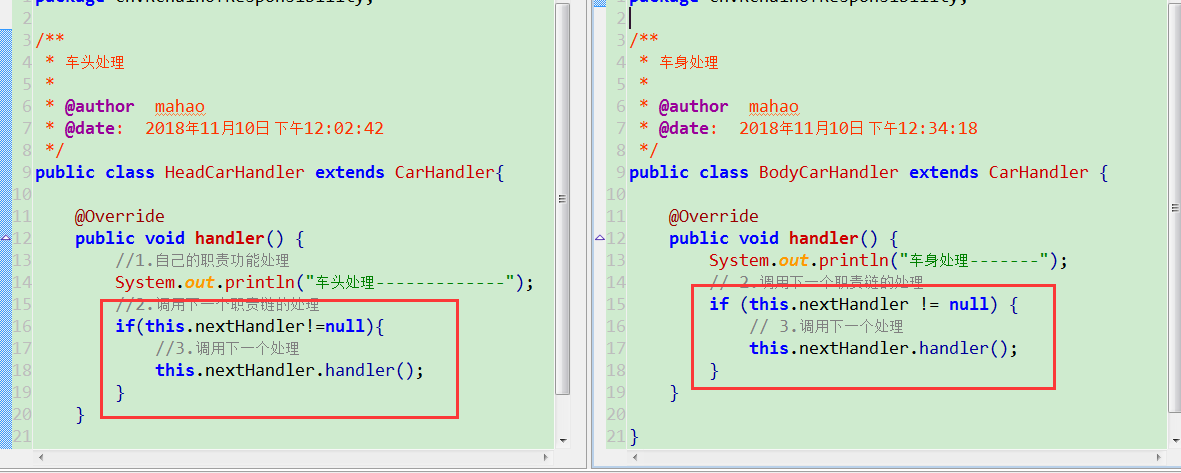
1. 说明：该模式是行为模式，将一个工作分为几个handler去完成，每一个handler含有下一个handdler对象的引用，并且当前handler调用下一个handler
2. 案列：汽车装配
3. 测试案例：



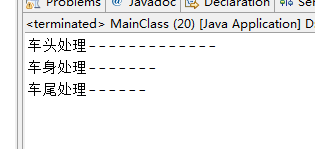
1. handler的抽象父类



1. 子类实现

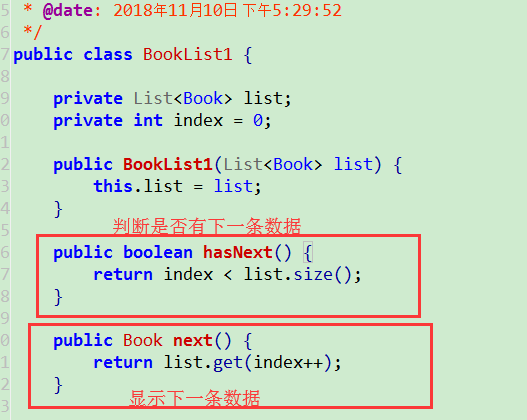


1. 结果

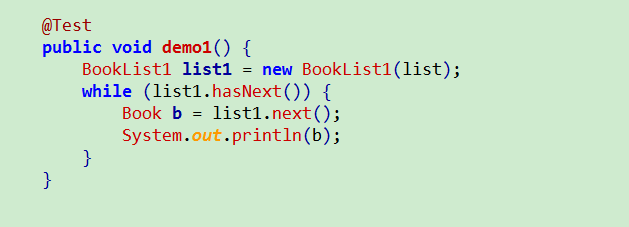


## 迭代模式

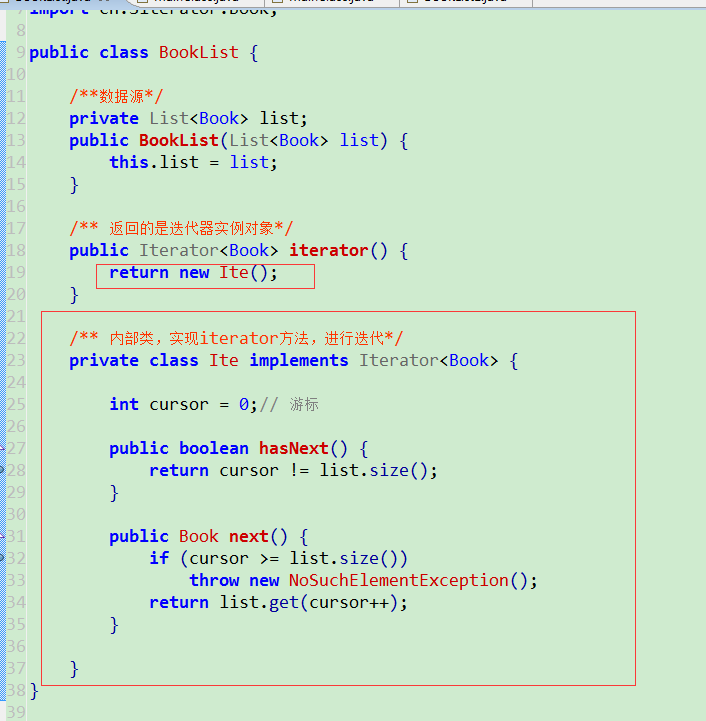
1. 说明： 实现集合或者容器内部数据的遍历。迭代模式是将一个类的功能交由另一个去实现，简化了类的内部结构，迭代模式本身还是调用的这个类的实例方法。
2. 两种方式：
3. 内部自身实现实现迭代逻辑



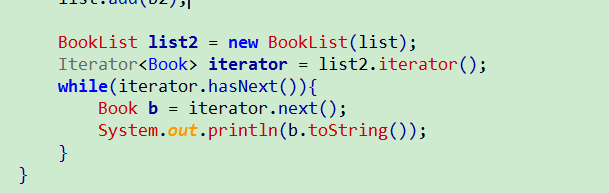
调用



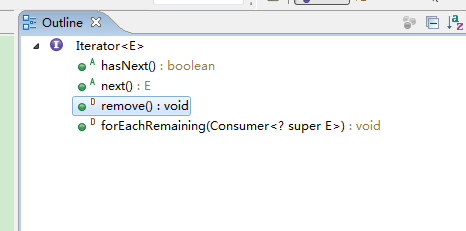
1. 使用迭代模式



调用



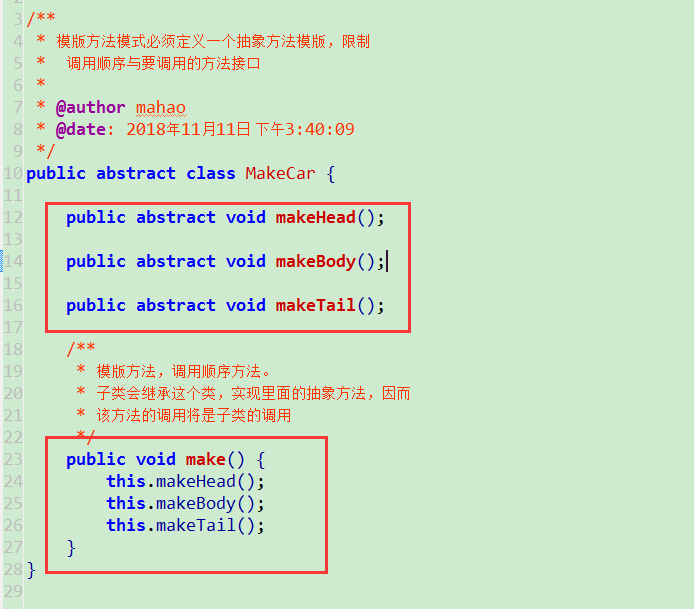
Java.util包的Iterator接口



## 模版方法模式

1. 说明，用于方法的顺序调换，一个功能中有多个方法，方法的具体实现是不确定的，但是必须按照顺序调用这些方法，则把这些方法抽成一个方法调用。
2. 格式： 一个抽象的模版父类，定义了子类的需要实现的抽象方法，和方法调用的顺序
3. 案例：

抽象模版父类



子类的实现：



模式调用：

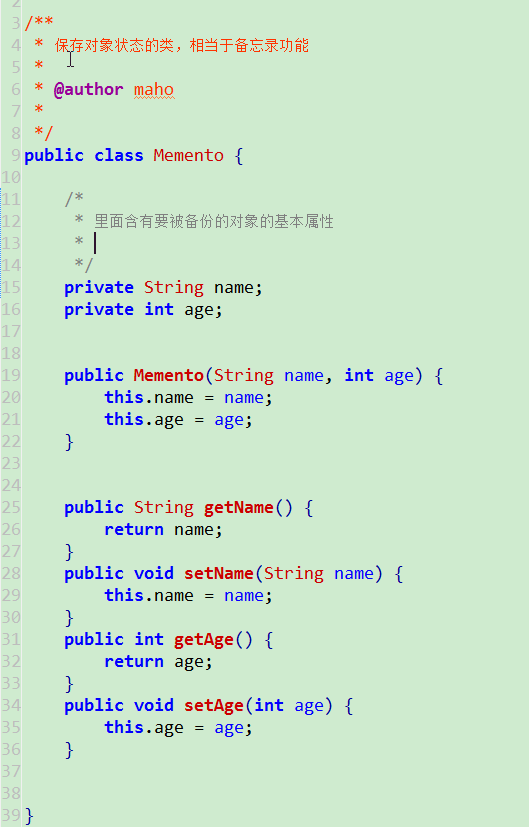


## 备忘录模式

1. 说明：用来保存一个对象的数据，必要时恢复这些数据
2. 组成：要备份的对象（Originator），备份对象（Memento），管理对象（Caretaker）
3. 案列：
4. 被备份对象



1. 备份对象



1. 管理备份对象的类



1. 测试

