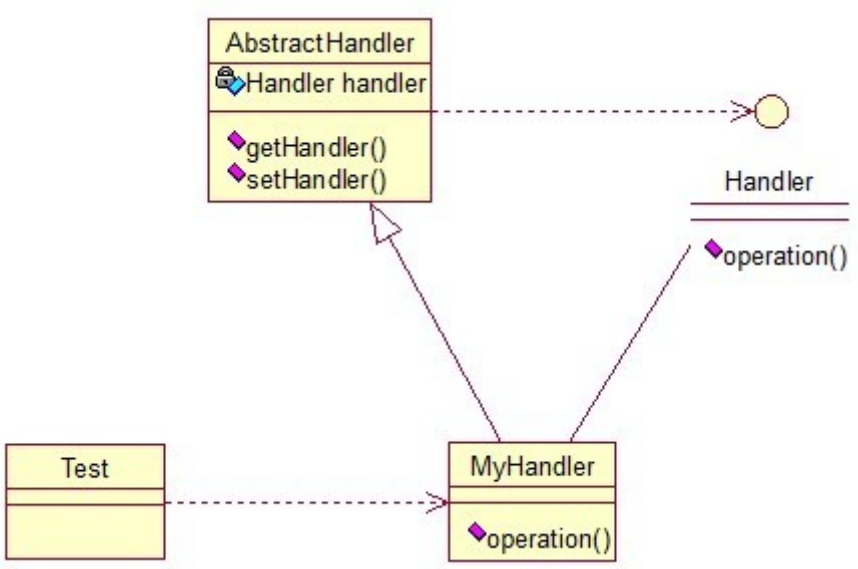
责任链模式（Chain of Responsibility）

接下来我们将要谈谈责任链模式，有多个对象，每个对象持有对下一个对象的引用，这样就会形成一条链，请求在这条链上传递，直到某一对象决定处理该请求。但是发出者并不清楚到底最终那个对象会处理该请求，所以，责任链模式可以实现，在隐瞒客户端的情况下，对系统进行动态的调整。先看看关系图：



Abstracthandler类提供了get和set方法，方便MyHandle类设置和修改引用对象，MyHandle类是核心，实例化后生成一系列相互持有的对象，构成一条链。

[java] view plaincopy

```
1. public interface Handler {
2.     public void operator();
3. }
```

[java] view plaincopy

```
1. public abstract class AbstractHandler {
2.
3.     private Handler handler;
4.
5.     public Handler getHandler() {
6.         return handler;
7.     }
8.
9.     public void setHandler(Handler handler) {
10.         this.handler = handler;
11.     }
12.
```

```
13. }
```

[java] view plaincopy

```
1. public class MyHandler extends AbstractHandler implements Handler {
2.
3.     private String name;
4.
5.     public MyHandler(String name) {
6.         this.name = name;
7.     }
8.
9.     @Override
10.     public void operator() {
11.         System.out.println(name+"deal!");
12.         if(getHandler()!=null){
13.             getHandler().operator();
14.         }
15.     }
16. }
```

[java] view plaincopy

```
1. public class Test {
2.
3.     public static void main(String[] args) {
4.         MyHandler h1 = new MyHandler("h1");
5.         MyHandler h2 = new MyHandler("h2");
6.         MyHandler h3 = new MyHandler("h3");
7.
8.         h1.setHandler(h2);
9.         h2.setHandler(h3);
10.
11.         h1.operator();
12.     }
13. }
```

输出：

h1deal!

h2deal!

h3deal!

此处强调一点就是，链接上的请求可以是一条链，可以是一个树，还可以是一个环，模式本身不约束这个，需要我们自己去实现，同时，在一个时刻，命令只允许由一个对象传给另一个对象，而不允许传给多个对象。