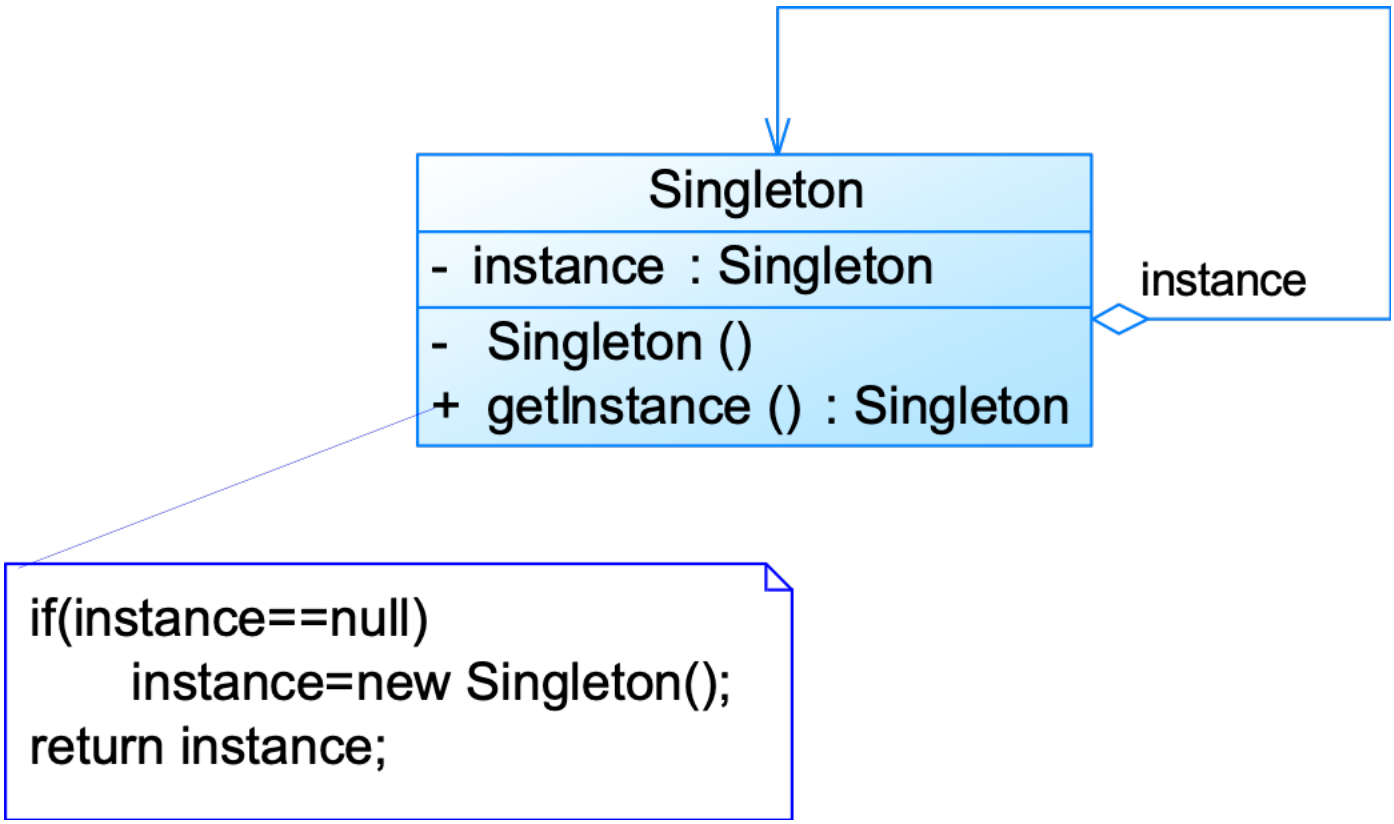
1 Singleton单例模式

1.1 类图



1.2 代码

饿汉就是声明时实例化

懒汉就是用到的时候再实例化

双向加锁孤子模式

```
package h6.q1;

public class Singleton {
    private static volatile Singleton singleton;
    private Singleton(){}
    public static Singleton getInstance()
    {
        if(singleton==null)
            synchronized (Singleton.class)
            {
                if(singleton==null)
                    singleton=new Singleton();
            }
        return singleton;
    }
```

```
}
```

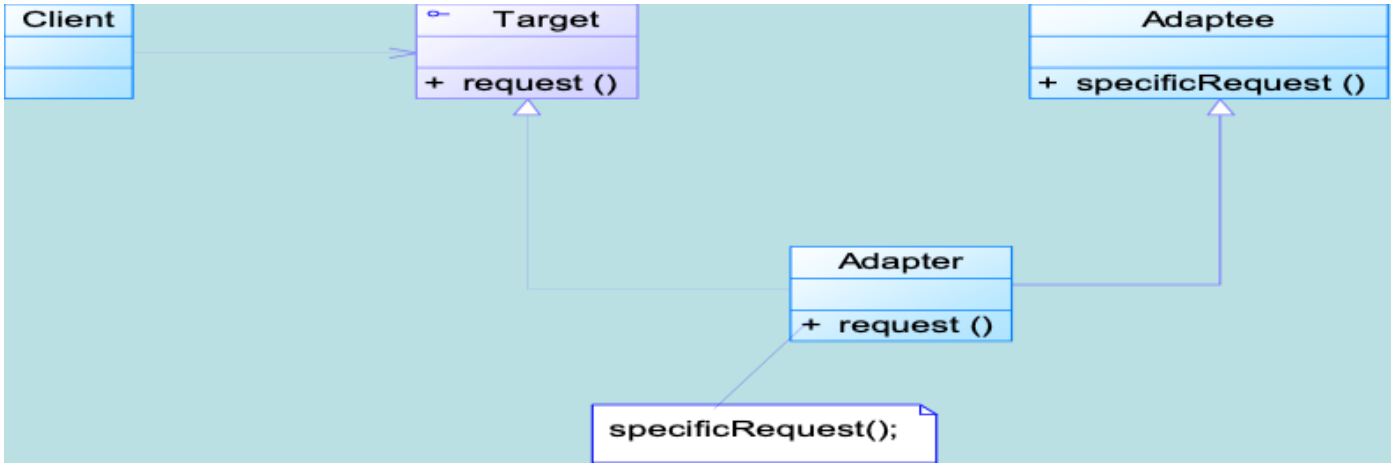
可变用例数目孤子模式

```
package h6.q2;

import java.util.ArrayList;
import java.util.List;

public class MultiSingleton {
    private static volatile int maxnum = 0;
    private static volatile List<MultiSingleton> singletons = new ArrayList<>();

    private MultiSingleton() {
    }

    public static void setMaxnum(int maxnum) {
        MultiSingleton.maxnum = maxnum;
    }

    public static MultiSingleton getInstance() {
        if (singletons.size() < maxnum) {
            synchronized (MultiSingleton.class) {
                System.out.println("synchronized now.");
                if (singletons.size() < maxnum) {
                    singletons.add(new MultiSingleton());
                    System.out.println("create an instance:" + singletons.size());
                }
            }
            System.out.println("unsynchronized now.");
        }
        return singletons.get(singletons.size()-1);
    }

}
```

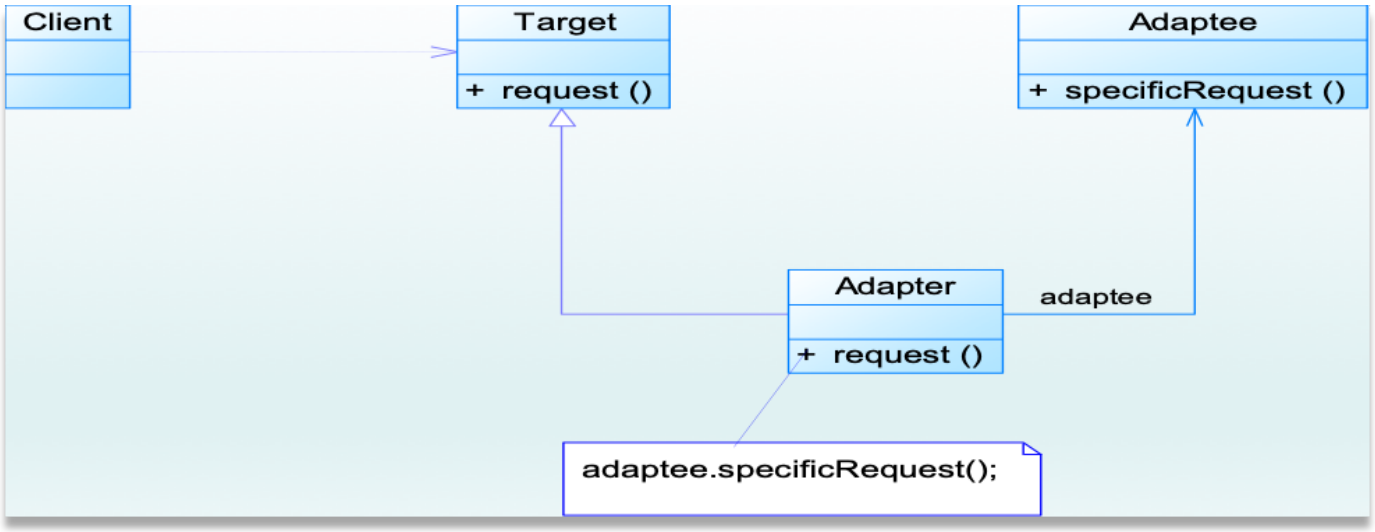
2 Adapter适配器

2.1 类图

类适配



对象适配



2.2 代码

双向适配器

```
package h7;

public interface AC220 {
    public void showAC220();
}
```

```
package h7;

public interface DC12 {
    public void showDC12();
}
```

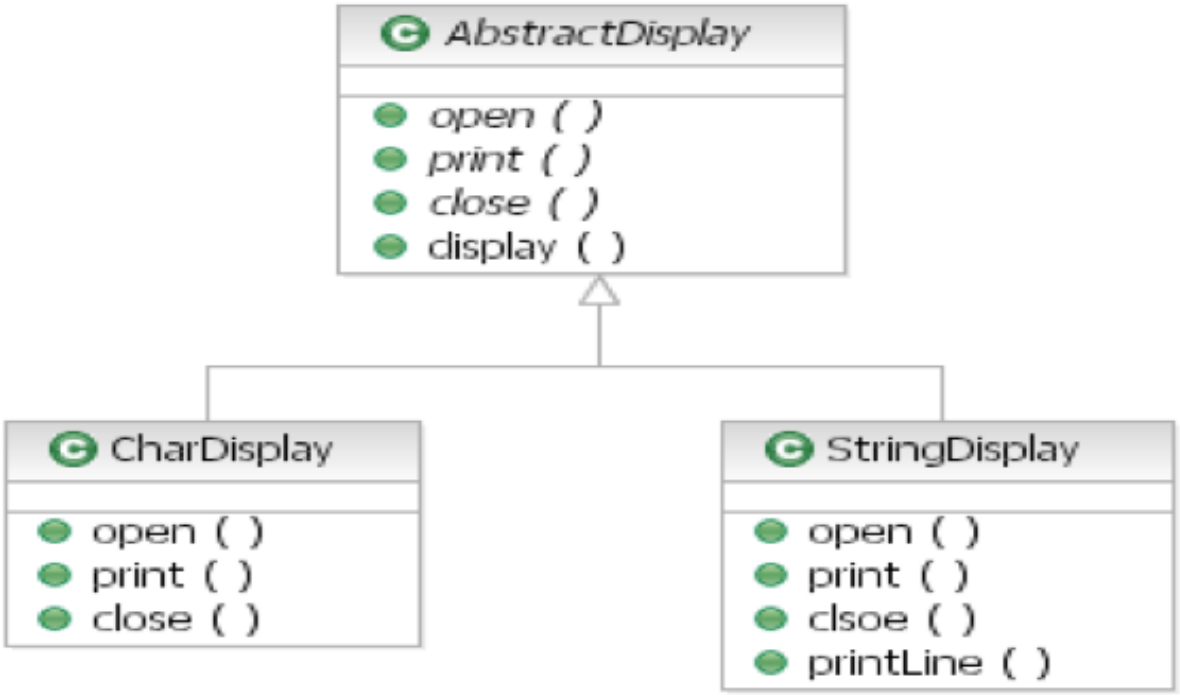
```java
package h7;

public class AC220Impl implements AC220{
    @Override
    public void showAC220() {
        System.out.println("h17.AC220");
    }
}
```

```java
package h7;

public class DC12Impl implements DC12{
    @Override
    public void showDC12() {
        System.out.println("h17.DC12");
    }
}
```

```java
package h7;

public class Adaptor {
    AC220 ac220;
    DC12 dc12;

    public Adaptor(AC220 ac220, DC12 dc12) {
        this.ac220 = ac220;
        this.dc12 = dc12;
    }

    public void transformAC220()
    {
        System.out.print("transform to");
        ac220.showAC220();
    }

    public void transformDC12()
    {
        System.out.print("transform to");
        dc12.showDC12();
    }
}
```

3 Template模板方法

3.1 类图



3.2 代码

这里代码是IOC

```java
package h8;

public interface MessageService {
    public void sendMessage(String message);
}
```

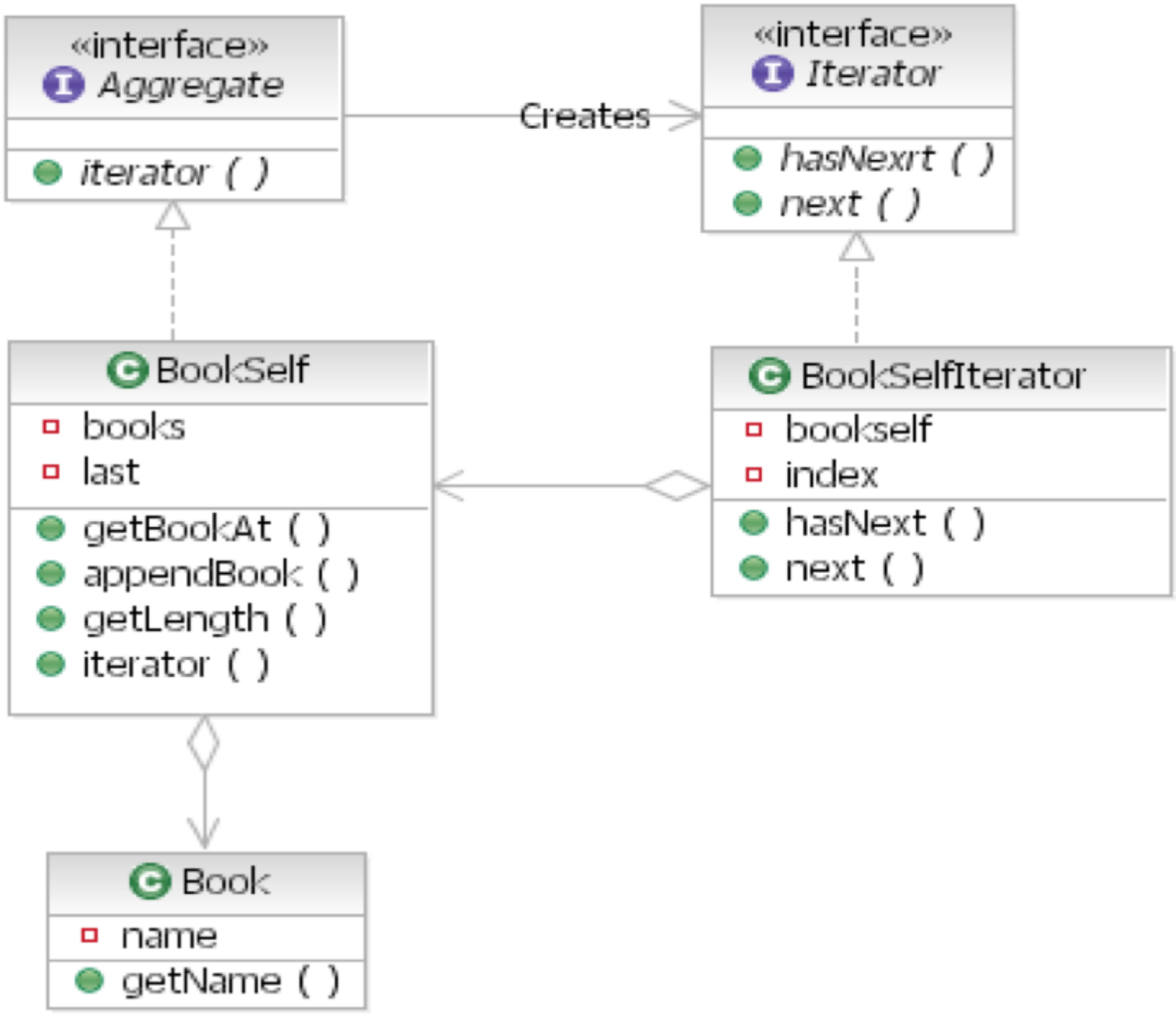
```java
package h8;

public class EmailMessageService implements MessageService{
    @Override
    public void sendMessage(String message) {
        System.out.println("sending email message:"+message);
    }
}
```

```java
package h8;

public class MyApplication {
    private final MessageService messageService;

    public MyApplication(MessageService messageService) {
        this.messageService = messageService;
    }

```

```
10      public void send(String message)
11      {
12          messageService.sendMessage(message);
13      }
14
15  }
```

4 Iterator迭代器

4.1 类图



4.2 代码

```java
package h9.q1;

public interface Aggregate {
    public Iterator iterator();
}
```

```java
package h9.q1;

public class Book {
   String bookName;

    public Book(String bookName) {
        this.bookName = bookName;
    }

    public String getBookName() {
        return bookName;
    }
}
```

```java
package h9.q1;

import java.util.ArrayList;
import java.util.List;

public class BookShelf implements Aggregate{

    List<Book> books=new ArrayList<>();
    public void append(Book book)
    {
        books.add(book);
    }

    public int getLength()
    {
        return books.size();
    }

    public Book getBookAt(int index)
    {
        return books.get(index);
    }

    @Override
    public Iterator iterator() {
        return new BookShelfIterator(this);
    }
}
```

```java
package h9.q1;

public class BookShelfIterator implements Iterator{
    private BookShelf bookShelf;
    private int index;
    public BookShelfIterator(BookShelf bookShelf) {
        this.bookShelf = bookShelf;
        this.index=0;
    }

    @Override
    public boolean hasNext() {
        if(index<bookShelf.getLength()) {
            return true;
        }
        return false;
    }

    @Override
    public Object next() {
        Book book = bookShelf.getBookAt(index);
        index++;
        return book;
    }
}
```

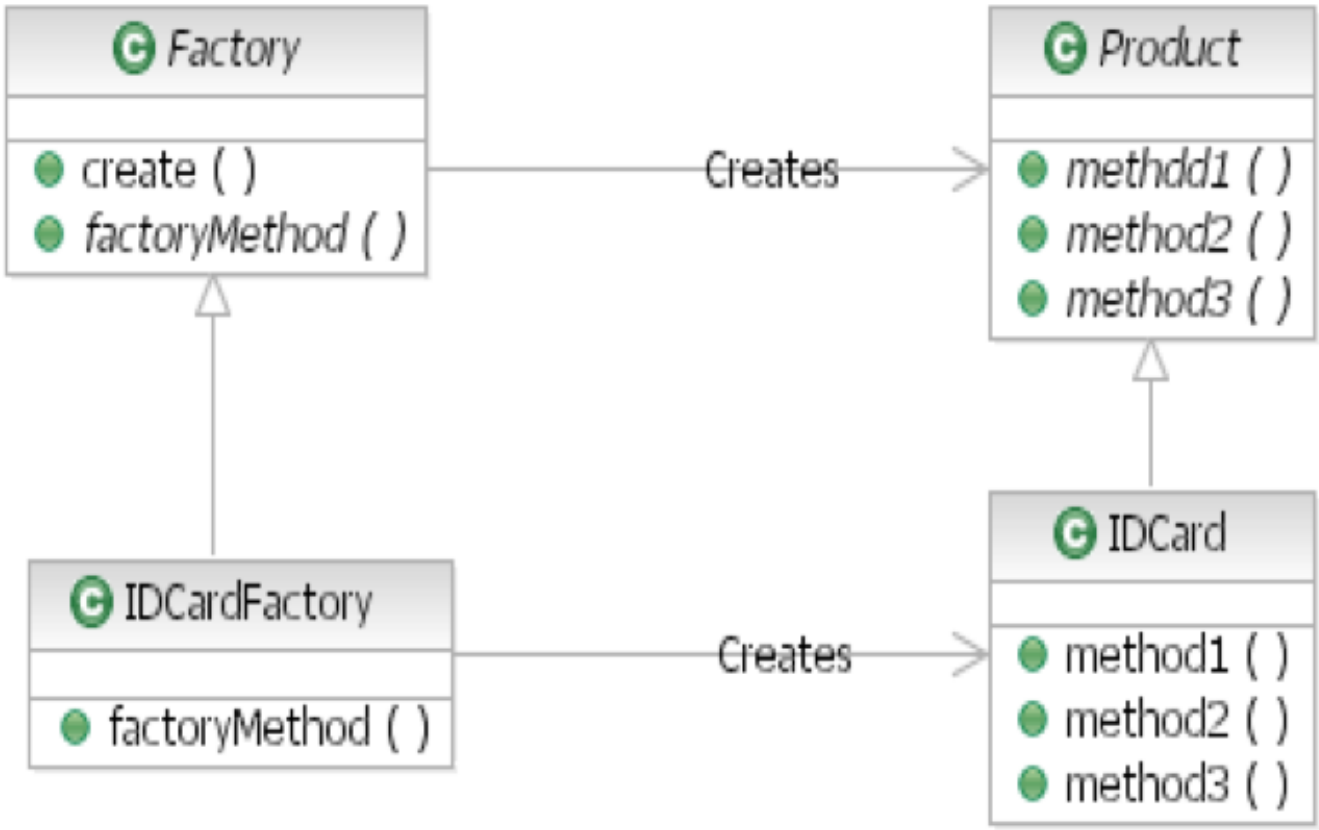
```java
package h9.q1;

public interface Iterator {
    public boolean hasNext();
    public  Object next();
}
```

```java
public static void main(String[] args) {
  Book b1=new Book("悲惨世界");
  Book b2=new Book("安娜");
  Book b3=new Book("三字经");
  BookShelf bookShelf=new BookShelf();
  bookShelf.append(b1);
  bookShelf.append(b2);
  bookShelf.append(b3);
  Iterator iterator=bookShelf.iterator();
  while (iterator.hasNext())
  {
    Book book = (Book) iterator.next();
    System.out.println(book.getBookName());
```

```
  }
}
```

5 Factory工厂方法

5.1 类图



5.2 代码

```java
package h10.framwork;

public abstract class Factory {
    public final Product create(String color)
    {
        Product product=createProduct(color);
        register(color);
        return product;
    }

    abstract protected Product createProduct(String color);
    abstract protected void register(String color);

}
```

```java
package h10.framwork;

public abstract class Product {
    public abstract void use();
}
```

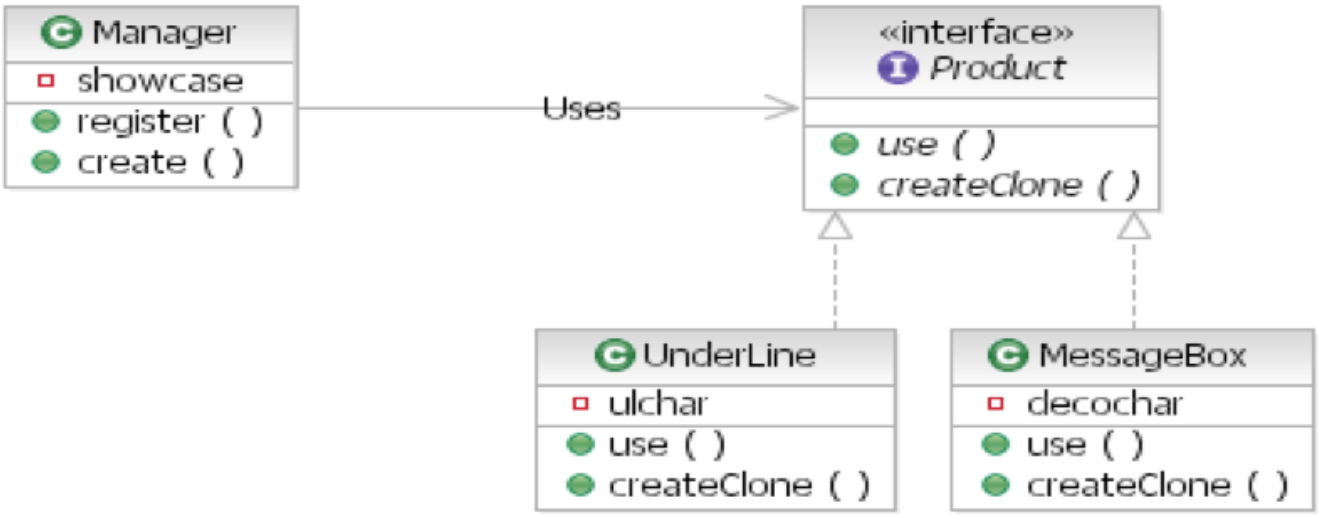
```java
package h10.toy;

import h10.framwork.Product;

public class Toy extends Product {
    String color;

    public Toy(String color) {
        this.color = color;
    }

    @Override
    public void use() {
        System.out.println(color);
    }

    public String getColor() {
        return color;
    }
}
```

```java
package h10.toy;

import h10.framwork.Factory;
import h10.framwork.Product;

import java.util.ArrayList;
import java.util.List;

public class ToyFactory extends Factory {

    List<String> colors=new ArrayList<>();
    @Override
    protected Product createProduct(String color) {
        return new Toy(color);
    }

    @Override
    protected void register(String color) {
        colors.add(color);
    }

```

```
22      List<String> getColors()
23      {
24          return colors;
25      }
26
27  }
```

6 ProtoType原型方法

6.1 类图



6.2 代码

```
1  package h11.q1;
2
3  public interface Shape extends Cloneable{
4      public void draw();
5      public Shape createClone() throws CloneNotSupportedException;
6  }
```

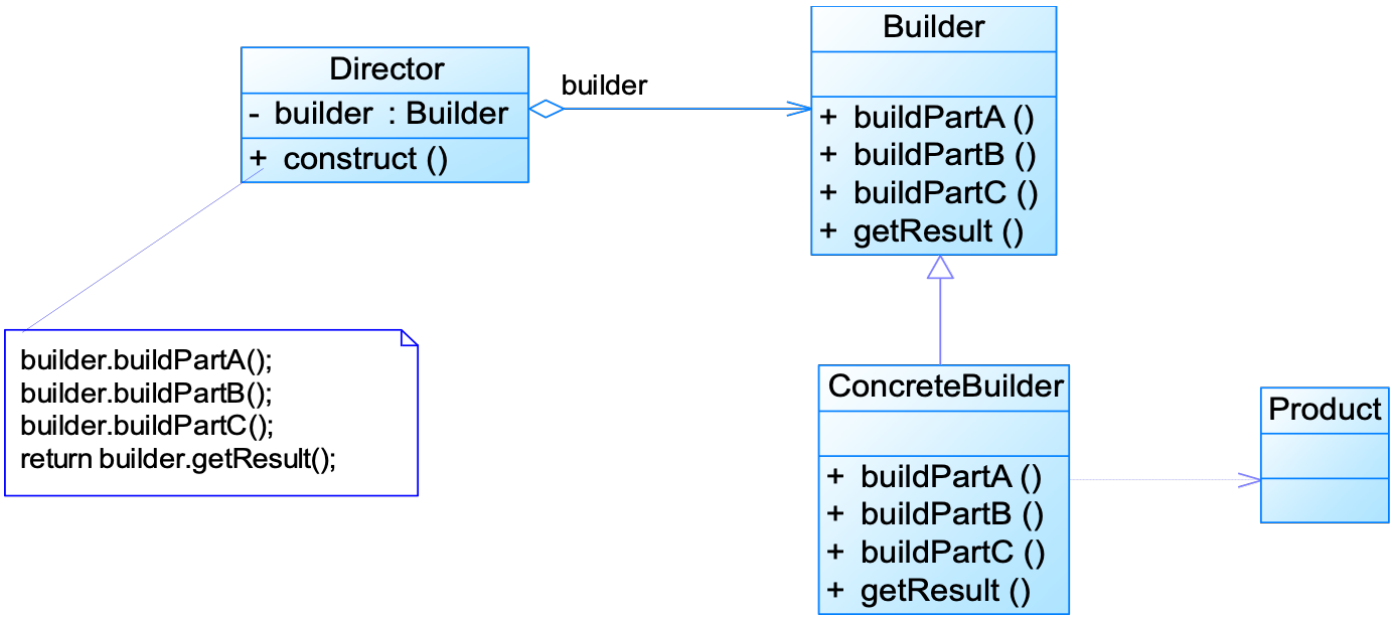
```
1  package h11.q1;
2
3  public class Rectangle implements Shape{
4      @Override
5      public void draw() {
6          System.out.println("画一个矩形");
7      }
8
9      @Override
10     public Shape createClone() {
11         Shape shape=null;
12         try {
```

```
            shape=(Shape) clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return shape;
    }
}
```

```
package h11.q1;

import java.util.HashMap;

public class Manager {

    HashMap hashMap=new HashMap();

    public void register(String name,Shape shape)
    {
        hashMap.put(name,shape);
    }

    public Shape create(String name)
    {
        Shape shape=null;
        shape= (Shape) hashMap.get(name);
        try {
            return shape.createClone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(e);
        }
    }

}
```

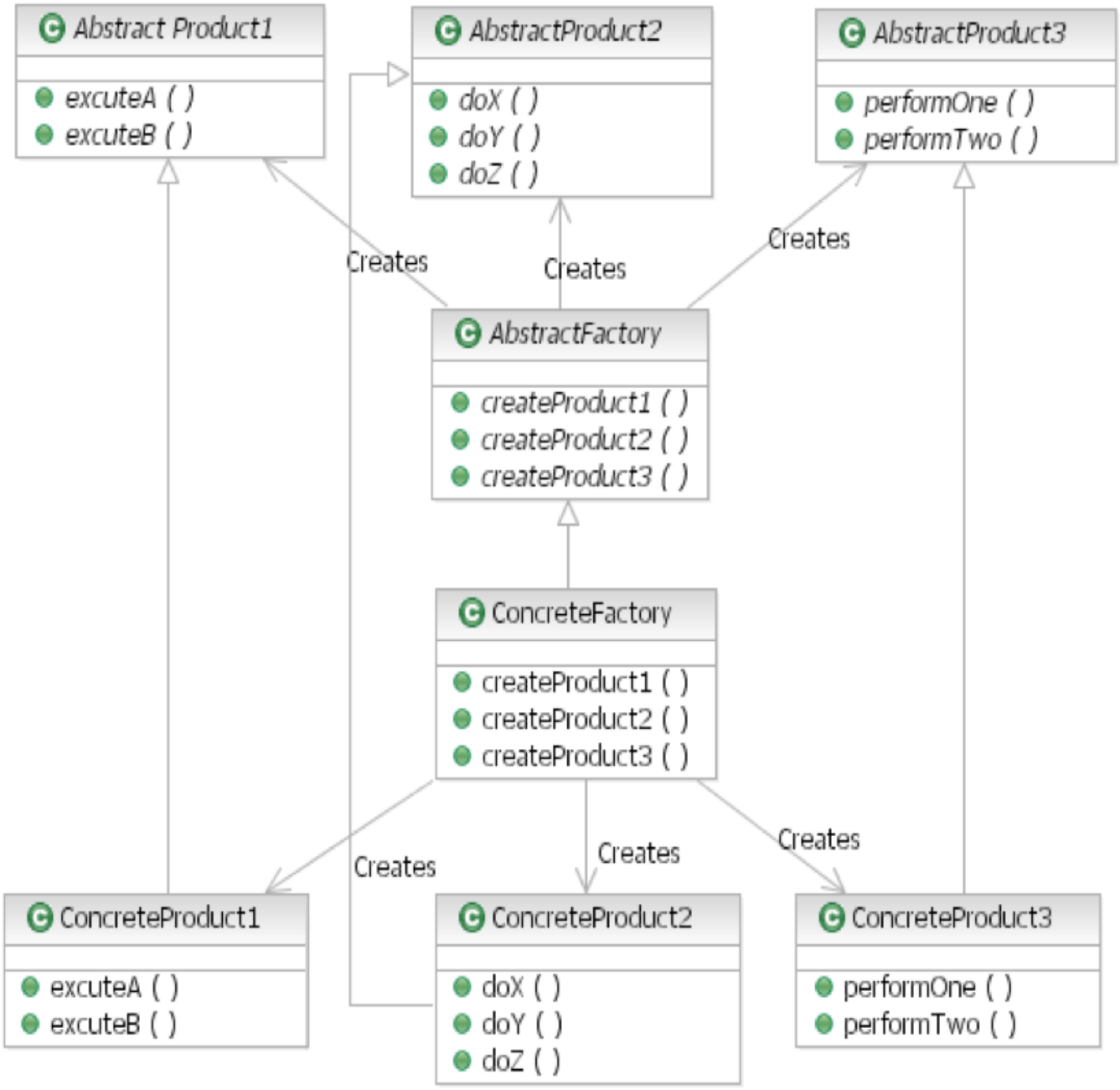
7 Builder创建者

7.1 类图



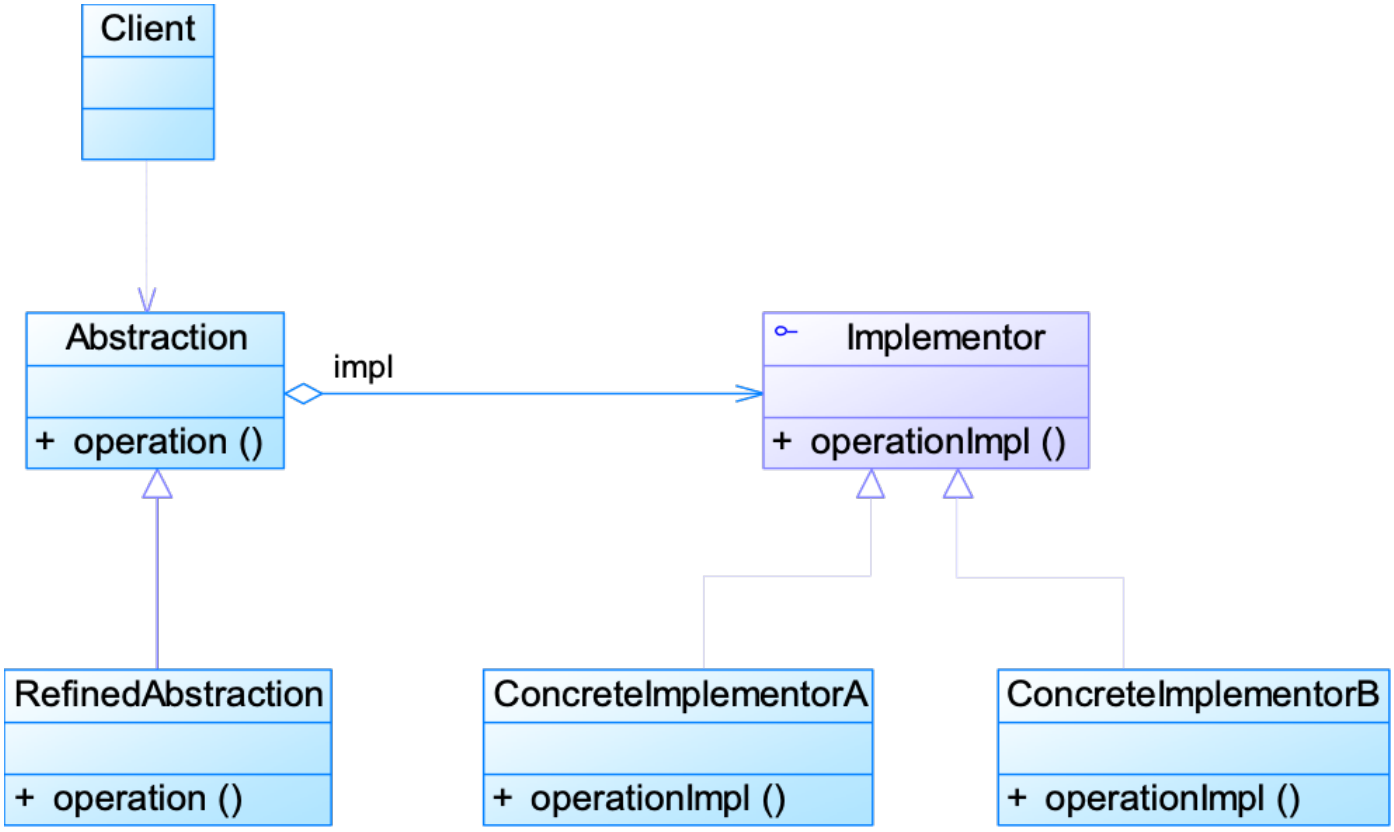
8 Abstrat FactoryMethod

8.1 类图



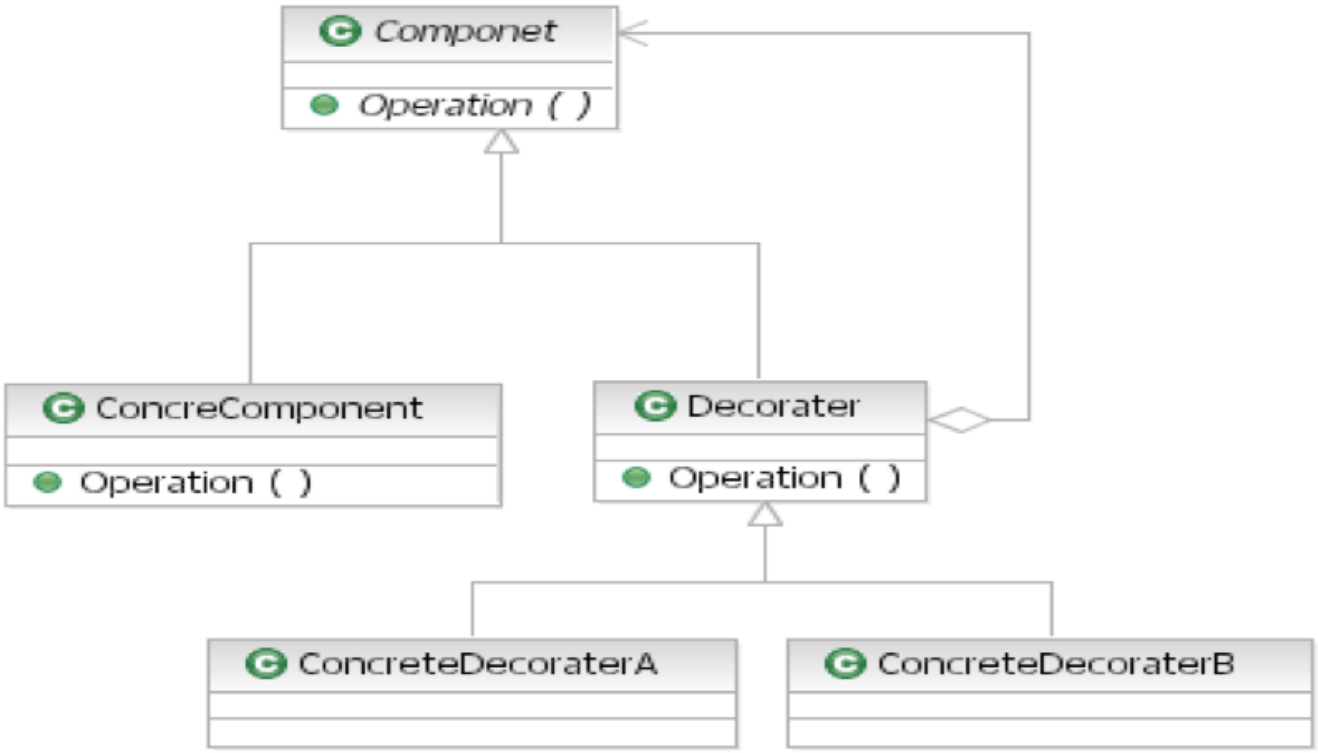
9 Bridge桥接器

9.1 类图



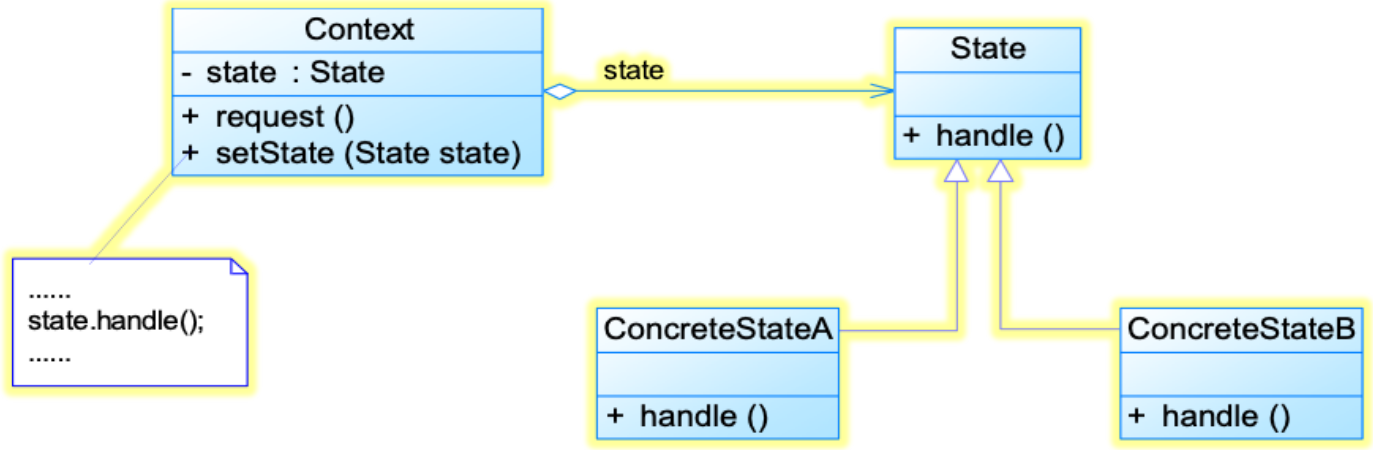
10 Decorator

10.1 类图



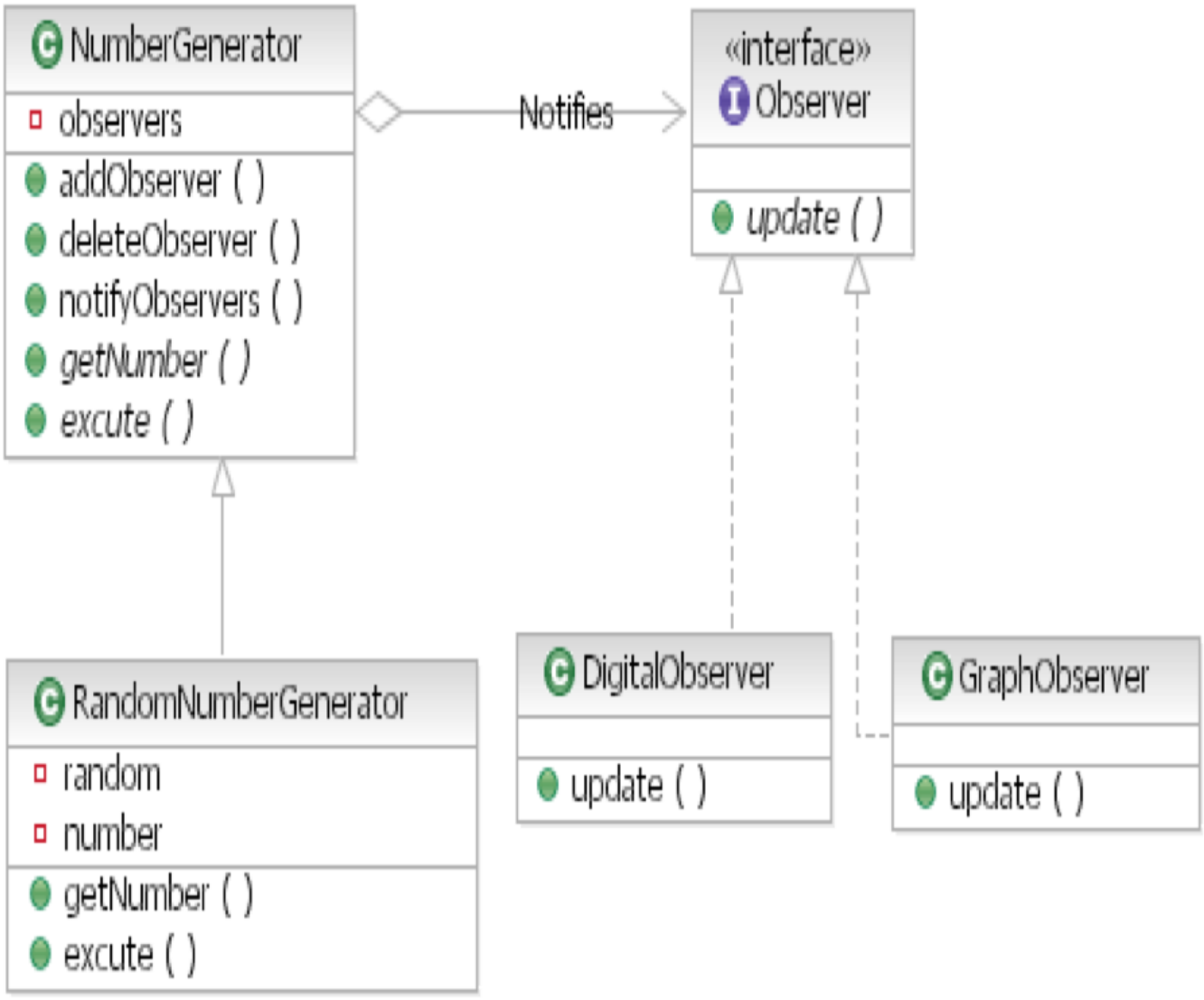
11 State状态模式

11.1 类图



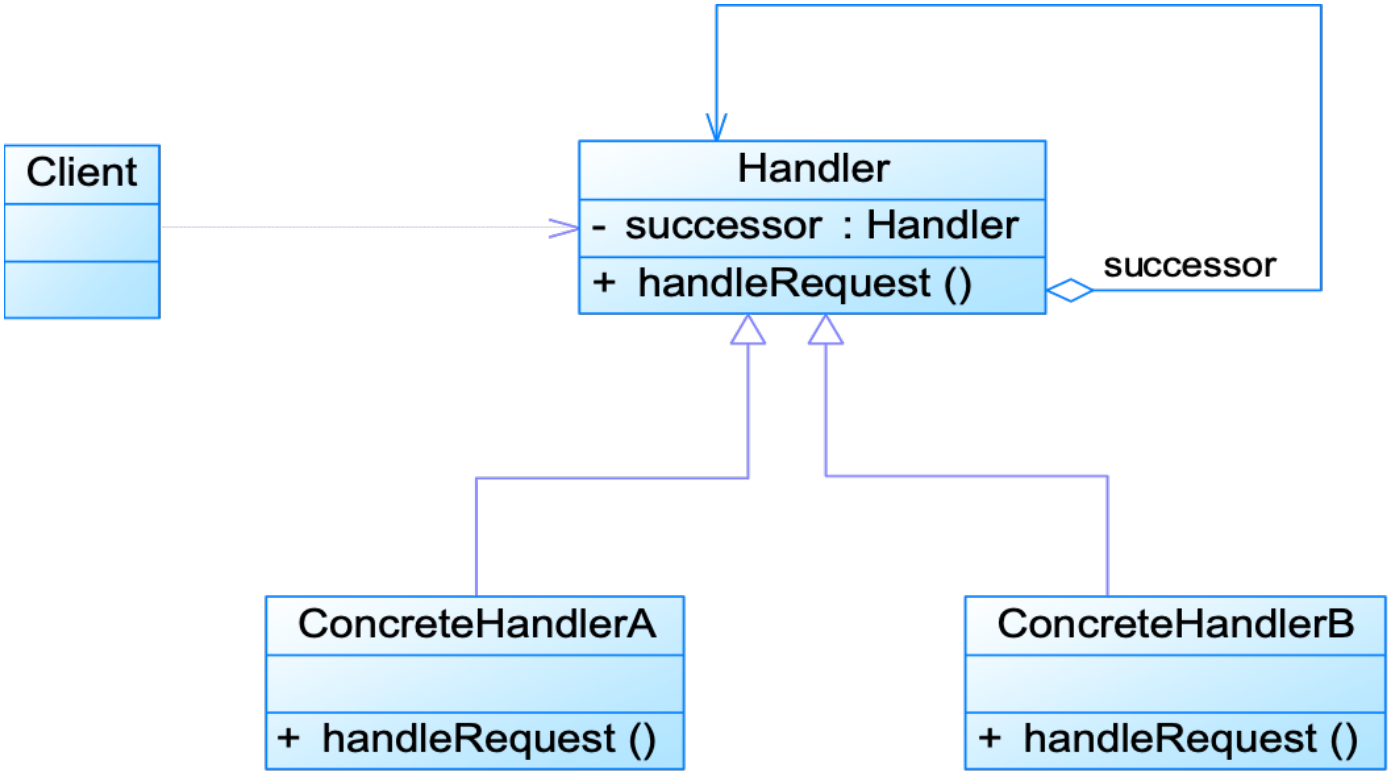
12 Observer观察者模式

12.1 类图



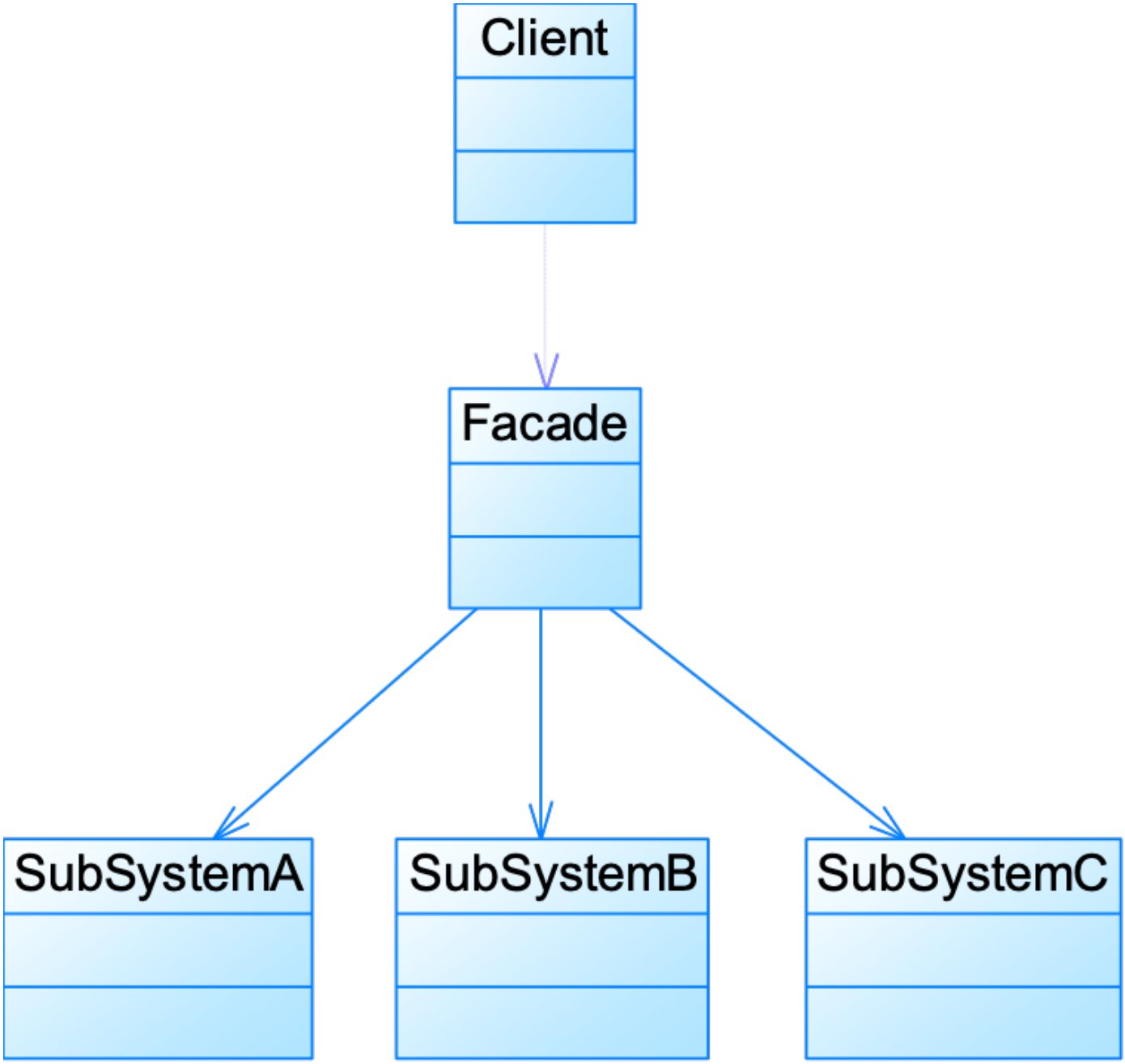
13 Chains of Responsibility

13.1 类图



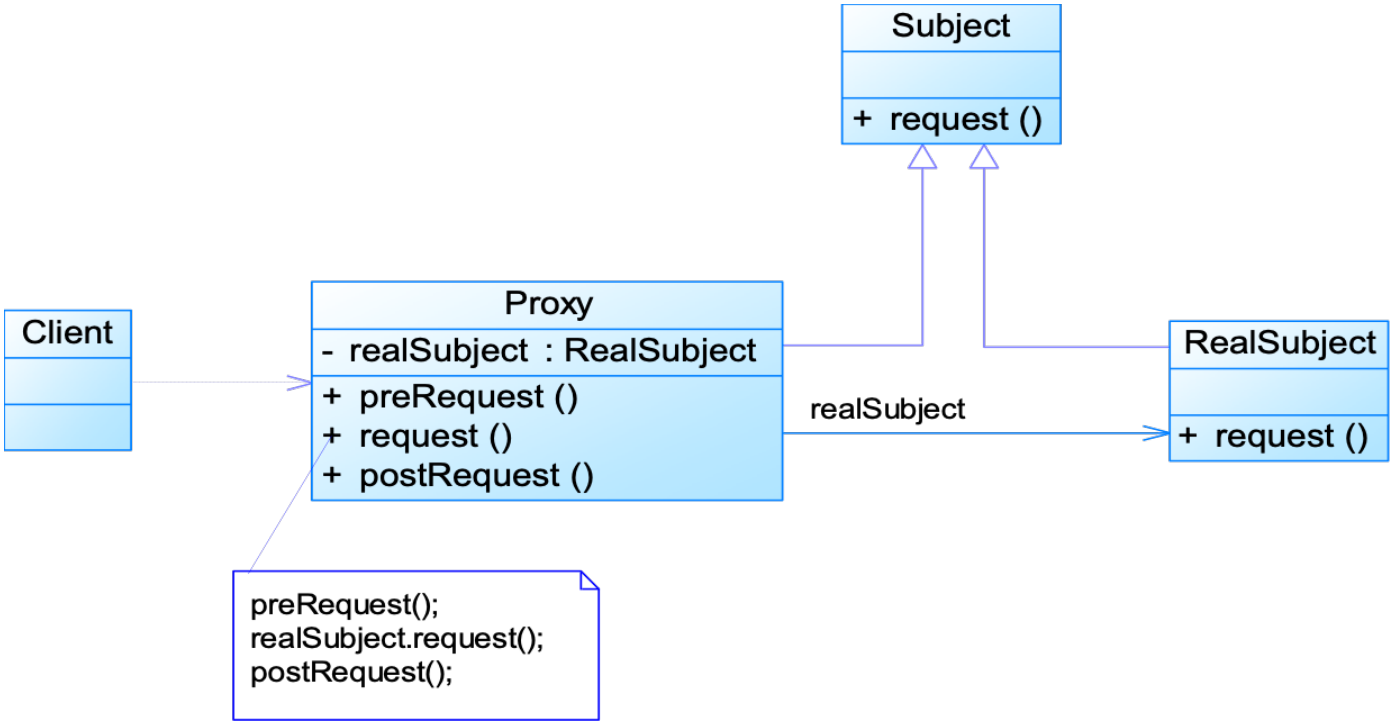
14 Facade外观模式

14.1 类图



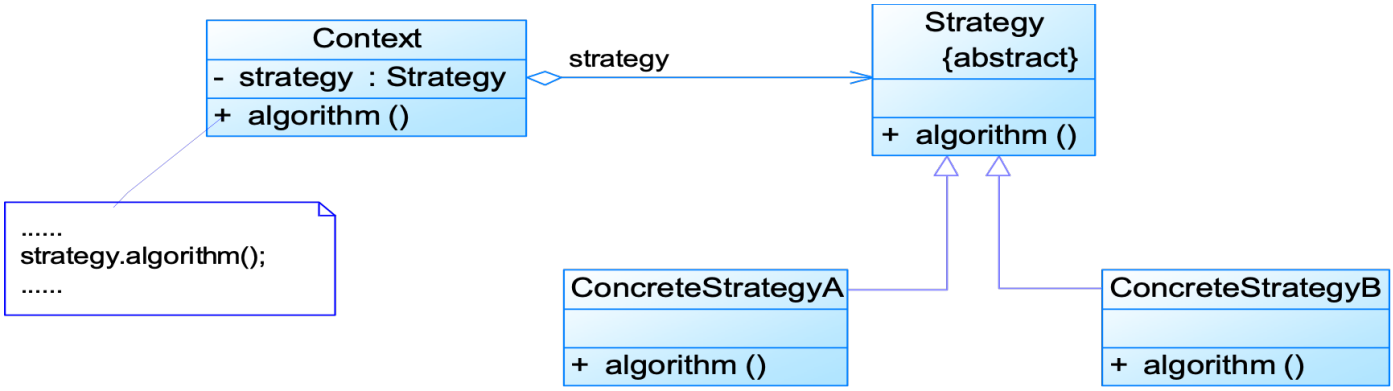
15 Proxy代理模式

15.1 类图

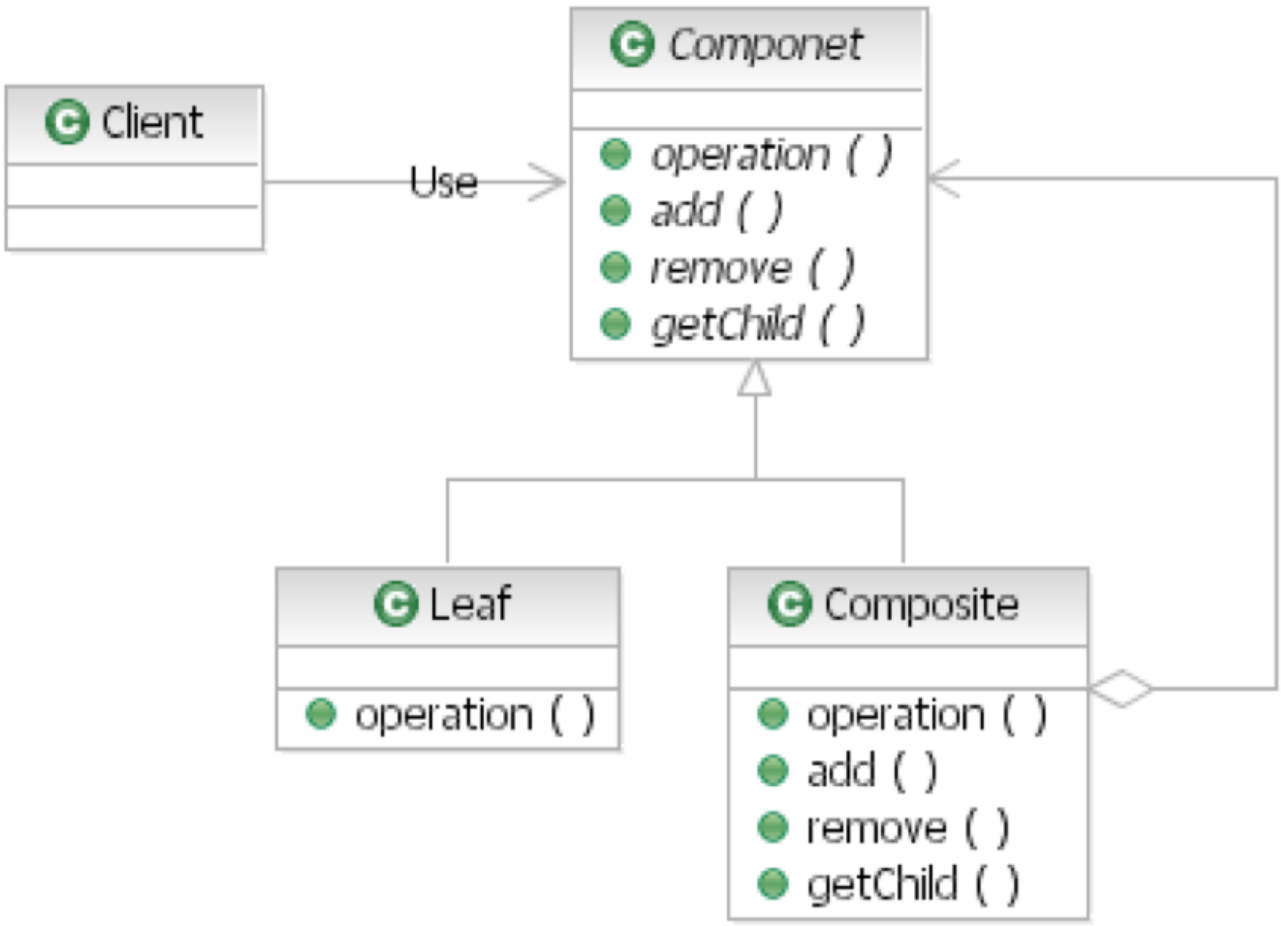


16 Strategy策略模式

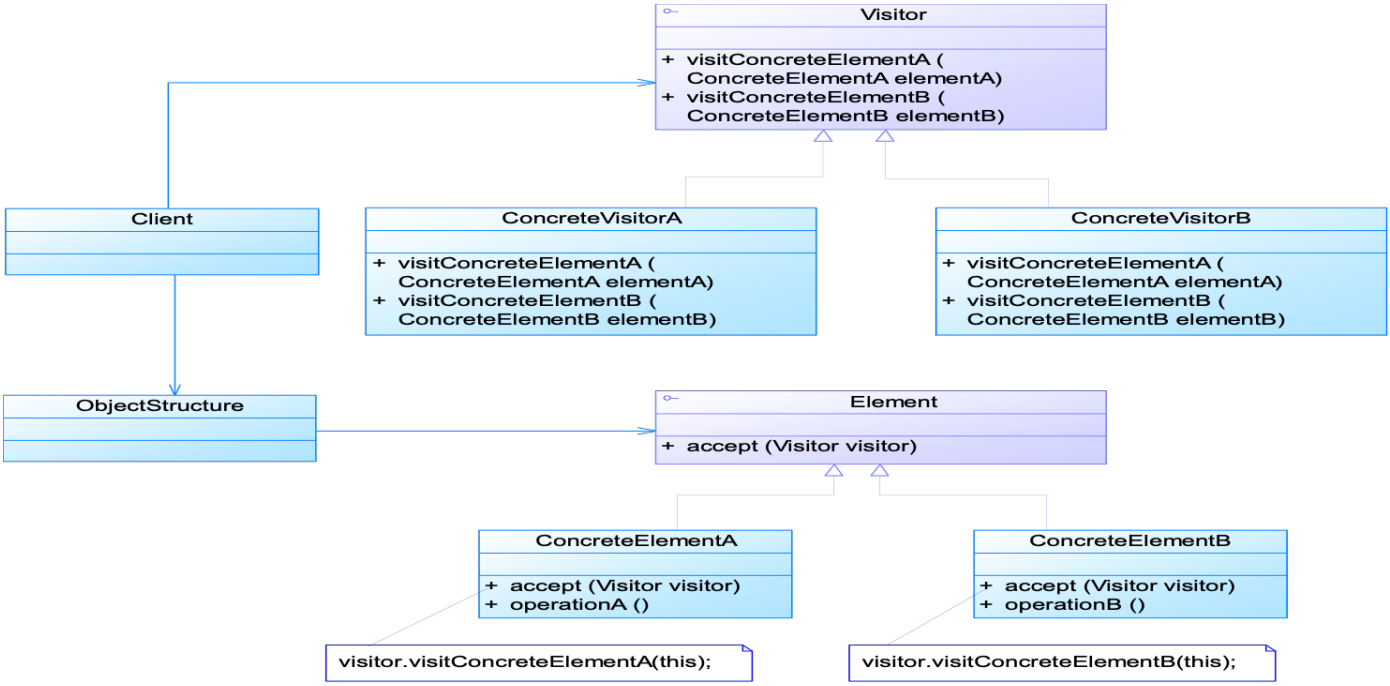
16.1 类图



17 Composite


Componet
Client
Use
operation ( )
add ( )
remove ( )
getChild ( )
Leaf
operation ( )
Composite
operation ( )
add ( )
remove ( )
getChild ( )

18 Visitor


Visitor
+ visitConcreteElementA (ConcreteElementA elementA)
+ visitConcreteElementB (ConcreteElementB elementB)
Client
ConcreteVisitorA
+ visitConcreteElementA (ConcreteElementA elementA)
+ visitConcreteElementB (ConcreteElementB elementB)
ConcreteVisitorB
+ visitConcreteElementA (ConcreteElementA elementA)
+ visitConcreteElementB (ConcreteElementB elementB)
ObjectStructure
Element
+ accept (Visitor visitor)
ConcreteElementA
+ accept (Visitor visitor)
+ operationA ()
ConcreteElementB
+ accept (Visitor visitor)
+ operationB ()
visitor.visitConcreteElementA(this);
visitor.visitConcreteElementB(this);