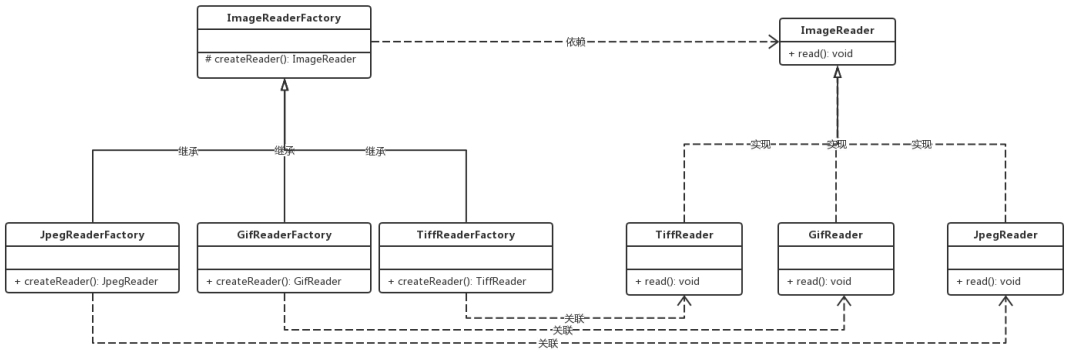
2019年设计模式考试题

1. 设计一个程序来读取多种不同存储格式的图片，针对每一种图片格式都设计一个图片读取器(ImageReader)，如GIF格式图片读取器(GifReader)用于读取GIF格式的图片，JPEG格式图片读取器(JpegReader)用于读取JPEG格式的图片，Tiff格式图片读取器(TiffReader)用于读取Tiff格式的图片。图片读取器对象通过图片读取器工厂(ImageReaderFactory)来创建，ImageReaderFactory是一个抽象类，用于定义创建图片读取器的工厂方法，其子类GifReaderFactory、JpegReaderFactory和TiffReaderFactory用于创建具体的图片读取器对象。请使用工厂方法模式实现该程序的设计，并使用UML画出该模式的类图。




```java
package first;

public interface ImageReader {
    public void read();
}
```

```java
package first;

public class GifReader implements ImageReader {
    @Override
    public void read() {
        System.out.println("读取Gif图片");
    }
}
```

```java
package first;

public class JpegReader implements ImageReader {
    @Override
    public void read() {
        System.out.println("读取Jpeg图片");
    }
}
```

```java
package first;

public class TiffReader implements ImageReader {
    @Override
    public void read() {
        System.out.println("读取Tiff图片");
    }
}
```

```java
package first;

public abstract class ImageReaderFactory {
    public abstract ImageReader createReader();
}
```

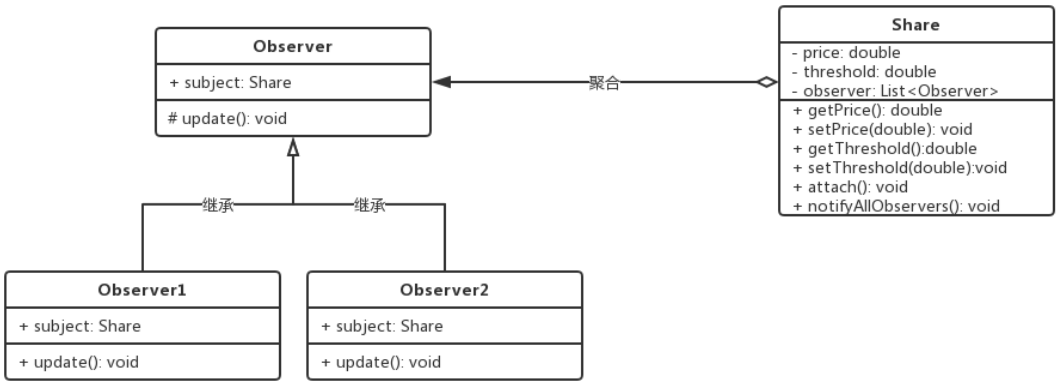
```java
package first;

public class GifReaderFactory extends ImageReaderFactory {
    @Override
    public ImageReader createReader() {
        return new GifReader();
    }
}
```

```java
package first;

public class JpegReaderFactory extends ImageReaderFactory {
    @Override
    public ImageReader createReader() {
        return new JpegReader();
    }
}
```

```java
package first;

public class TiffReaderFactory extends ImageReaderFactory {
    @Override
    public ImageReader createReader() {
        return new TiffReader();
    }
}
```

```java
package first;

public class Main {
    public static void main(String[] args) {
        ImageReaderFactory factory = new JpegReaderFactory();
        ImageReader jpegReader = factory.createReader();
        jpegReader.read();
        ImageReaderFactory factory1 = new GifReaderFactory();
        ImageReader gifReader = factory1.createReader();
        gifReader.read();
    }
}
// output:读取Jpeg图片
//        读取Gif图片
```

2. 某在线股票软件需要提供如下功能：当股票购买者购买的某只股票价格变化幅度达到5%时，系统将自动发送通知(包括新价格)给购买该股票的股民。现使用观察者模式设计该系统，绘制类图并编成实现。




```java
package second;

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

public class Share {
    // 股票价格
    private double price;
    // 股票波动通知用户的阈值
    private double threshold;
    // 订阅本支股票的所有用户
    private List<Observer> observer = new ArrayList<>();

    public Share() {
        this.price = 100;
        this.threshold = 0.05;
    }

    public Share(double price, double threshold) {
        this.price = price;
        this.threshold = threshold;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        double temp = this.price;
        this.price = price;
        double range = Math.abs(temp - price);
        if (range > (this.threshold * this.getPrice())) {
            this.notifyAllObservers();
        }

    }

    public double getThreshold() {
```

```
        return threshold;
    }

    public void setThreshold(double threshold) {
        this.threshold = threshold;
    }

    public void attach(Observer observer) {
        this.observer.add(observer);
    }

    public void notifyAllObservers() {
        observer.forEach((observe) -> observe.update());
    }
}
```

```
package second;

public abstract class Observer {
    public Share share;

    public abstract void update();
}
```

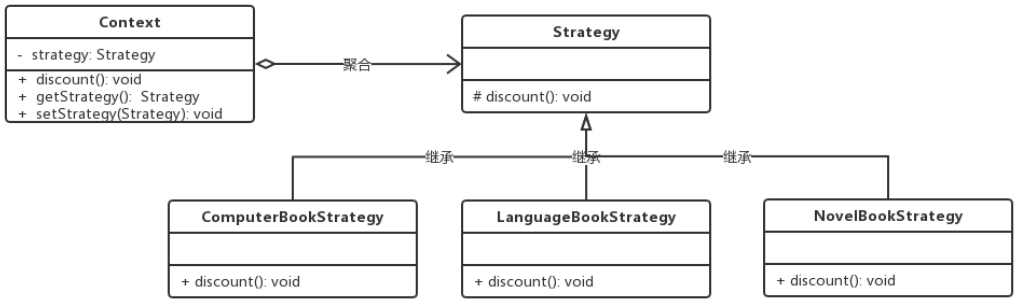
```
package second;

public class Observer1 extends Observer {

    @Override
    public void update() {
        System.out.println("Observer1收到股票变化通知信息,股票几个变为" +
share.getPrice());
    }

    public Observer1(Share share) {
        this.share = share;
        this.share.attach(this);
    }
}
```

```
package second;

public class Observer2 extends Observer {

    @Override
    public void update() {
        System.out.println("Observer2收到股票变化通知信息,股票几个变为" +
share.getPrice());
    }

    public Observer2(Share share) {
        this.share = share;
        this.share.attach(this);
    }
}
```

```
package second;
```

```java
public class Main {
    public static void main(String[] args) {
        // 定义股票 股票初始值100,变化幅度为0.05
        Share share = new Share(100, 0.05);
        // 定义用户，即观察者,并订阅股票
        Observer1 ob1 = new Observer1(share);
        Observer2 ob2 = new Observer2(share);
        // 股票发生波动
        share.setPrice(120);
        share.setPrice(140);
        share.setPrice(142);
    }
}
// output:  Observer1收到股票变化通知信息,股票几个变为120.0
//          Observer2收到股票变化通知信息,股票几个变为120.0
//          Observer1收到股票变化通知信息,股票几个变为140.0
//          Observer2收到股票变化通知信息,股票几个变为140.0
```

3. 设计一个网上书店，该系统中所有的计算机类图书(ComputerBook)每本都有10%的折扣，所有的语言类图书(LanguageBook)每本都有2元的折扣，小说类图书(NovelBook)每100元有10元的折扣。现使用策略模式来设计该系统，绘制类图并编成实现。




```java
package third;

public abstract class Strategy {
    public abstract void discount();
}
```

```java
package third;

public class ComputerBookStrategy extends Strategy {
    @Override
    public void discount() {
        System.out.println("每10本折后10%");
    }
}
```

```java
package third;

public class LanguageBookStrategy extends Strategy {
    @Override
    public void discount() {
        System.out.println("每本2元折扣");
    }
}
```

```java
package third;

public class NovelBookStrategy extends Strategy {
    @Override
    public void discount() {
        System.out.println("每100元折扣10元");
    }
}
```

```java
package third;

public class Context {
    private Strategy strategy;

    public Context() {
    }

    public Context(Strategy strategy) {
        this.strategy = strategy;
    }

    public Strategy getStrategy() {
        return strategy;
    }

    public void setStrategy(Strategy strategy) {
        this.strategy = strategy;
    }

    public void discount() {
        this.strategy.discount();
    }
}
```

```java
package third;

public class Main {
    public static void main(String[] args) {
        Context context = new Context();
        Strategy strategy = new ComputerBookStrategy();
        Strategy strategy2 = new LanguageBookStrategy();
        Strategy strategy3 = new NovelBookStrategy();
        context.setStrategy(strategy);
        context.discount();
        context.setStrategy(strategy2);
        context.discount();
        context.setStrategy(strategy3);
        context.discount();
    }
```

```
}
// output:每10本折后10%
//         每本2元折扣
//         每100元折扣10元
```

4，请查阅文献，并结合个人的学习和实践体会，举例谈谈你对某个特定设计模式的认识，以及该特定的设计模式(自选)在面向对象程序设计中的重要作用和应用实例。

单例模式是一种创建型模式，通过单例模式的方法创建的类在当前进程中只能有一个实例，提供一个全局访问点用来访问对象。单例模式是设计模式中最简单的形式之一，它的目的是使得类的一个对象成为系统中的唯一实例。因此需要用一种只允许生成对象类的唯一实例的机制，“阻止”想要生成对象的访问。使用工厂方法来限制实例化过程，且该方法应该是静态方法。

优点：

- 实例控制：阻止其他对象实例化自己的单例对象的副本，确保所有对象访问唯一的实例。
- 灵活性：控制了实例化过程，类可以灵活更改实例化过程。

缺点：

- 系统开销：虽然数量少，但如果每次对象请引用时都需要检查是否存在类的实例，需要额外花费开销。可使用静态方法解决此问题。
- 开发混淆：不能使用new关键字实例化对象，无法访问源代码时，会意外发现不能实例化对象。
- 对象生存期：不能产生删除单个对象的问题。在提供内存管理的语言中，只有单例类能导致实例被取消。在一些语言中其他类可以删除对象实例，会导致单例类出现引用悬浮。

单例模式的重点是阻止想要生成对象的访问，在java中有多数情况会破坏单例模式，下面介绍java语言中单例模式的特殊情况，并给出解决措施。

破坏单例模式：

- 线程同步，防止创建对象时线程切换,造成多创建
- 虚拟机优化，使用volatile防止对象不同步
- 反射调用私有方法，可在构造中判断this是否为空,为空抛异常)
- 反序列化
- 对象克隆

```
package fourth;
// 简单单例模式
public class Singleton {
    private static Singleton singleton;

    private Singleton() {

    }

    public static Singleton getInstance() {
        if(singleton == null){
            singleton = new Singleton();
        }
        return singleton;
    }
}
```

```
package fourth;
```

```java
import java.io.Serializable;

// 修改后的单例模式

// 使用线程同步创建，防止进程切换重复创建线程，
// 设置volatile关键字修饰，使读取singleton对象时能够获取最新状态
// 修改构造方法，防止反射创建对象
// 修改readResolve方法，防止反序列化对象时重新创建对象
// 重写克隆方法，防止对象克隆
public class Singleton2 implements Serializable, Cloneable {
    private static volatile Singleton2 singleton;

    private Singleton2() {
        if (singleton != null) {
            throw new RuntimeException("对象已被创建");
        }
    }

    public static Singleton2 getInstance() {
        if (singleton == null) {
            synchronized (singleton) {
                if (singleton == null)
                    singleton = new Singleton2();
            }
        }
        return singleton;
    }

    private Object readResolve() {
        return singleton;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return getInstance();
    }
}
```

要求：第1题、第2题和第3题中的实现代码，要能够通过编译运行测试并给出正确结果。

请在提交纸质答案时，将编程代码的电子版一并提交。