关于实验课

1. 使用腾讯课堂上课，如遇到技术故障将改用腾讯会议；
2. 为方便考勤，请同学们将昵称改成“学号-真实姓名”；
3. 上课不定时发起签到，请同学们不要迟到早退。

面向对象的软件构造导论

实验二：单例模式和工厂模式

2022春

哈尔滨工业大学（深圳）

本学期实验总体安排

实验项目	一	二	三	四	五	六
学时数	2	**2**	2	4	2	4
实验内容	飞机大战功能分析	**单例模式 工厂模式**	Junit与单元测试	策略模式 数据访问对象模式	Swing 多线程	模板模式 观察者模式
分数	4	**6**	4	6	6	14
提交内容	UML类图、代码	**UML类图、代码**	单元测试代码	UML类图、代码	代码	项目代码、实验报告

实验课程共16个学时，6个实验项目，总成绩为40分。

目录

实验目的

- 深入理解单例模式和工厂模式的意义，掌握各模式的结构
- 掌握绘制单例和工厂模式的UML类图
- 熟练使用代码实现单例和工厂模式

实验任务

绘制类图、重构代码，完成以下功能：

1. 采用单例模式创建英雄机；

2. 采用工厂模式创建三种敌机和三种道具。

实验步骤

1 英雄机应用场景分析

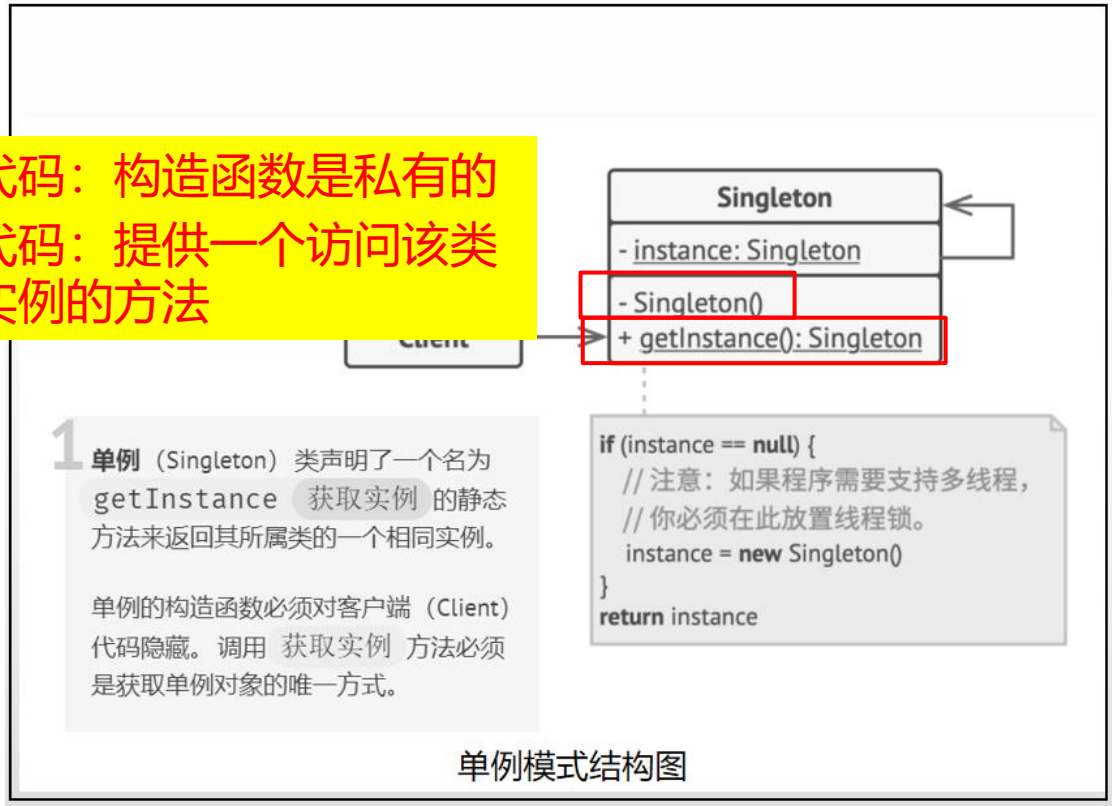
在飞机大战游戏中只有一种英雄机，且每局游戏只有一架英雄机。

单例模式

实验步骤

2 绘制单例模式类图

单例模式（Singleton Pattern）是一种创建型设计模式， 能够保证一个类只有一个实例， 并提供一个访问该实例的全局节点。



单例模式结构图

实验步骤

3 重构代码，实现单例模式

根据你所设计的UML类图，重构代码，采用单例模式创建英雄机。

实验步骤

3 重构代码，实现单例模式

- 单例模式代码示例（线程安全）：

① 饿汉式

```
public class EagerSingleton {
    private static EagerSingleton instance = new EagerSingleton ();
    private EagerSingleton (){}
    public static EagerSingleton getInstance() {
        return instance;
    }
}
```

② 懒汉式

```
public class LazySingleton {
    private static LazySingleton instance = null;
    private LazySingleton (){}
    public static synchronized LazySingleton getInstance() {
    if (instance == null) {
        instance = new LazySingleton();
    }
    return instance;
    }
}
```

实验步骤

3 重构代码，实现单例模式

- 单例模式代码示例（线程安全）：

③ 双重检查锁定（DCL，即 double-checked locking）

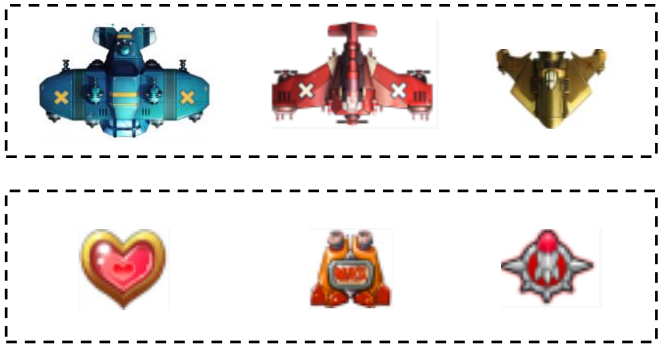
```
public class Singleton {
    private volatile static Singleton singleton;
    private Singleton (){}
    public static Singleton getSingleton() {
        if (singleton == null) {
            synchronized (Singleton.class) {
                if (singleton == null) {
                    singleton =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

实验步骤

4 敌机和道具应用场景分析

游戏中有3种类型敌机：Boss敌机、精英敌机、普通敌机。

游戏中有3种类型道具：加血道具、火力道具、炸弹道具。

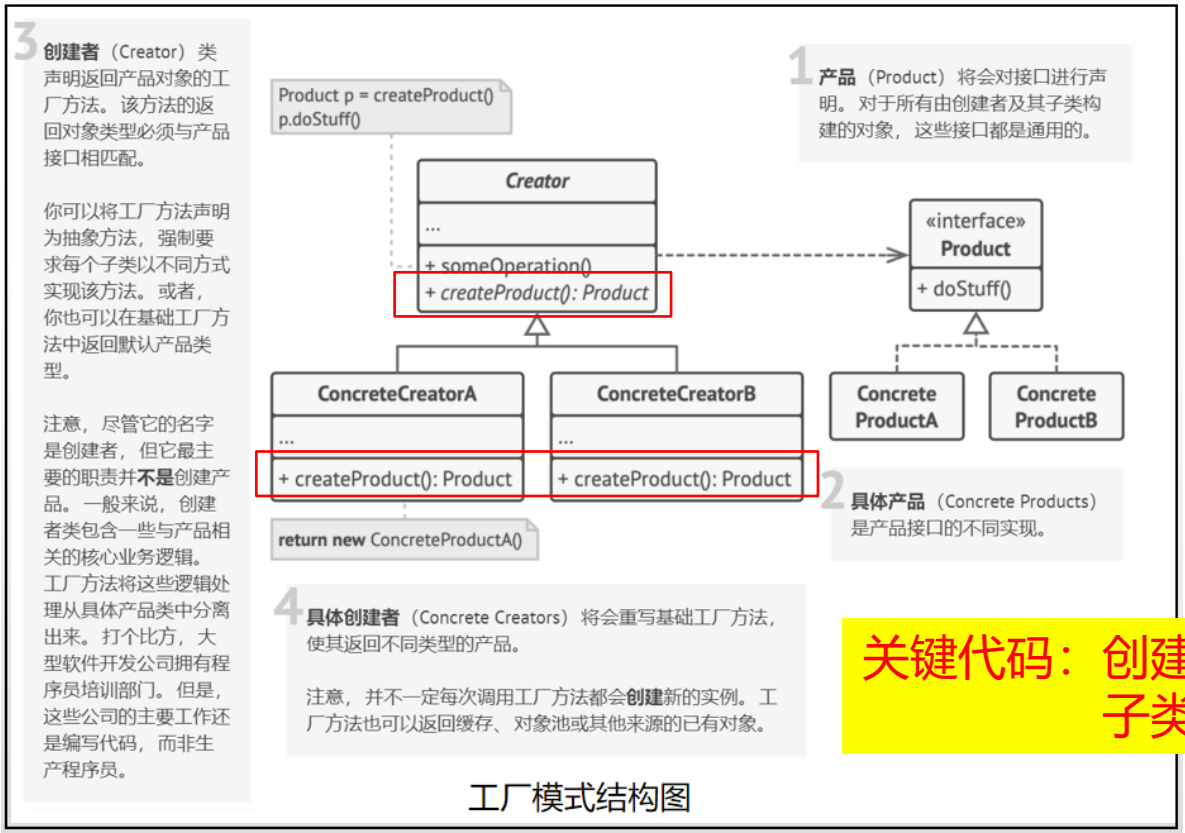


工厂模式

实验步骤

5 绘制工厂模式类图

工厂模式（Factory Pattern）也是一种创建型设计模式， 其在父类中提供一个创建对象的方法， 由子类决定实例化对象的类型。



工厂模式结构图

关键代码：创建过程在其子类执行

实验步骤

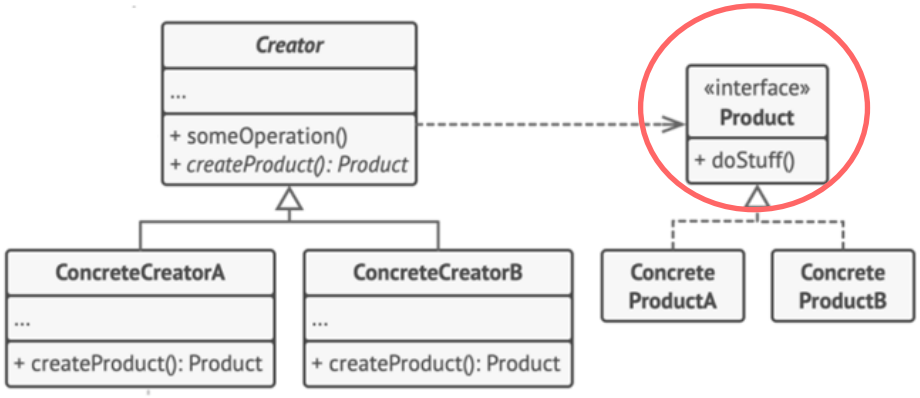
5 绘制工厂模式类图

假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?

实验步骤

5 绘制工厂模式类图

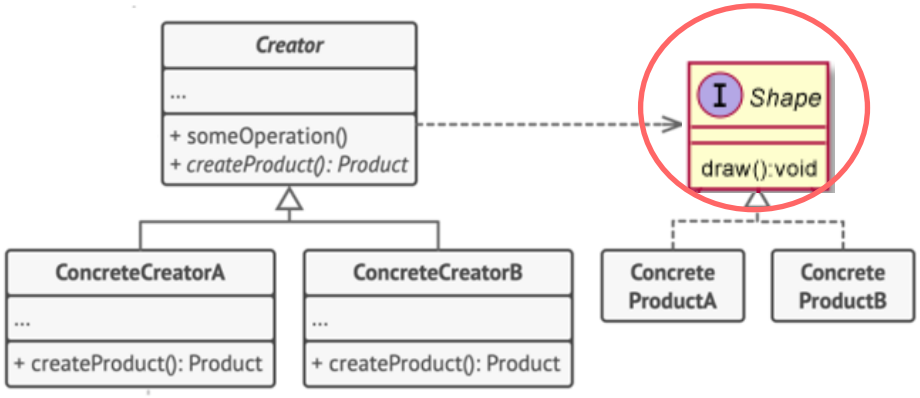
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

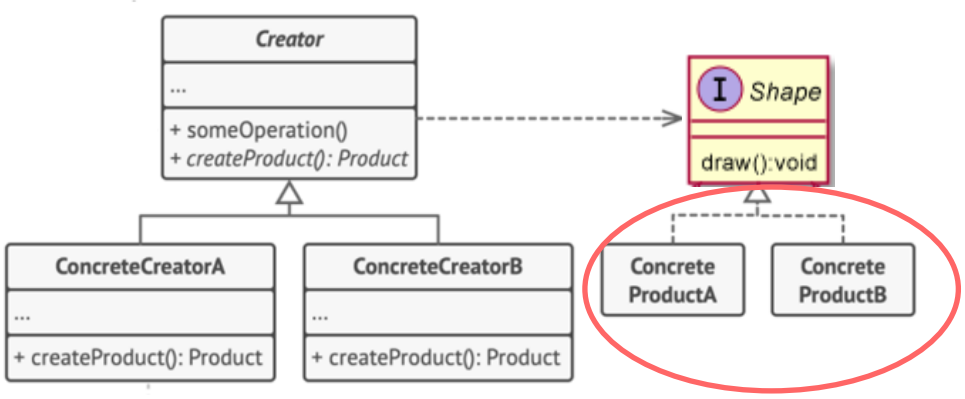
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

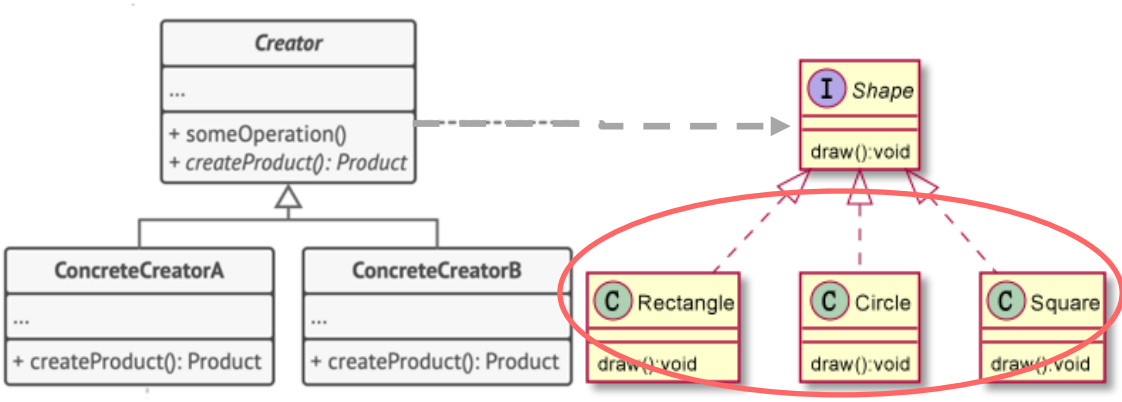
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

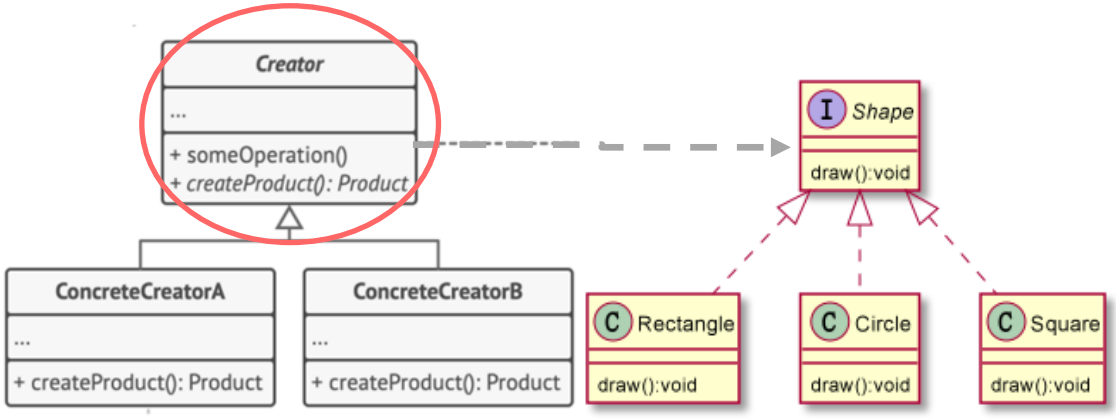
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

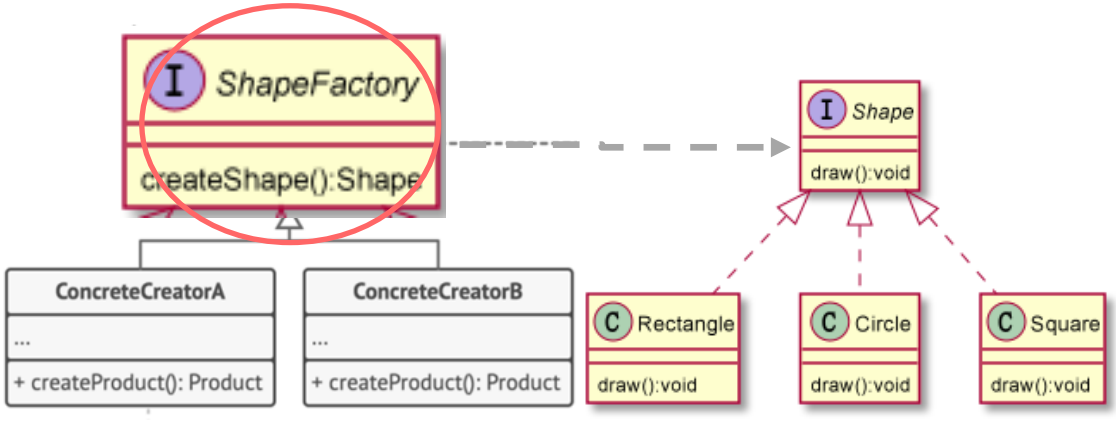
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

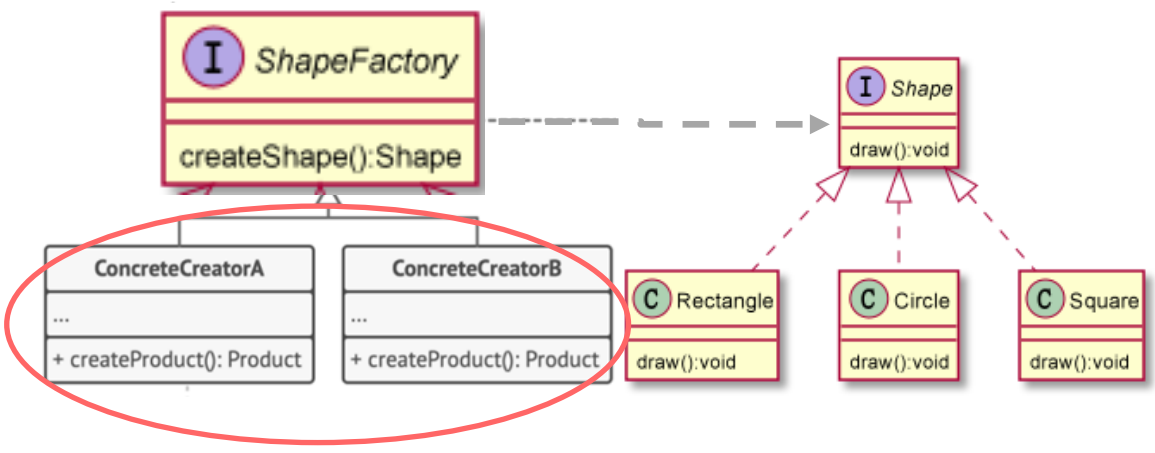
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

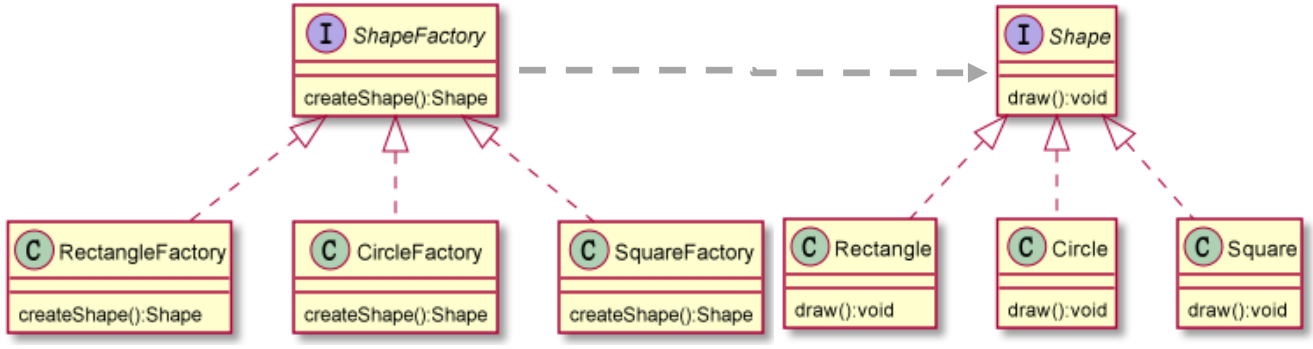
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

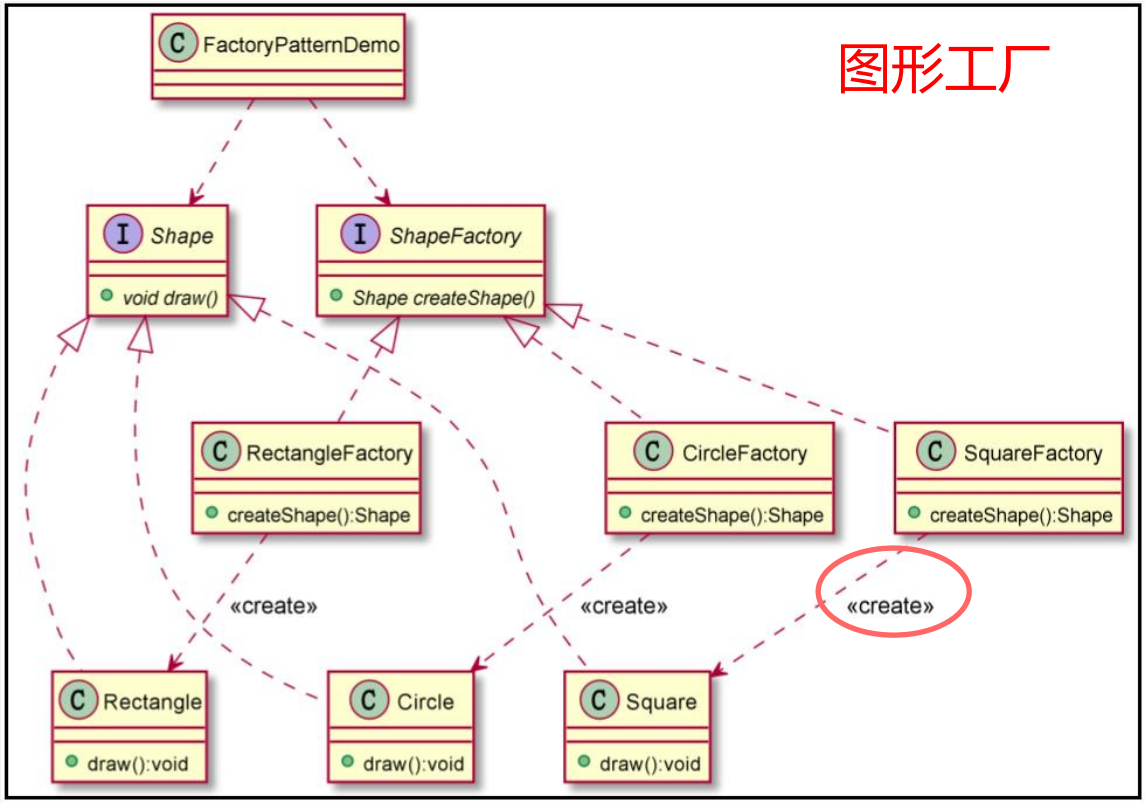
假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



实验步骤

5 绘制工厂模式类图

假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



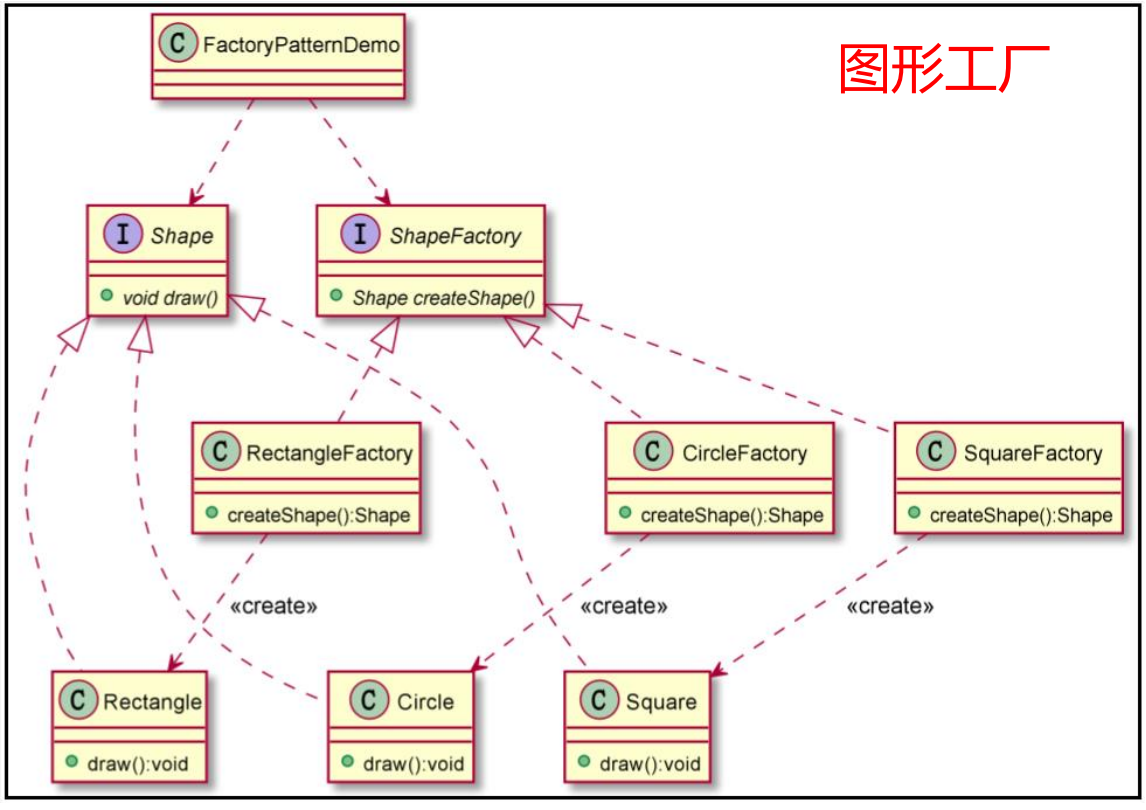
A、泛化　　B、实现

C、依赖　　D、关联

实验步骤

5 绘制工厂模式类图

假如我们要建一个图形工厂，生产3种产品：圆形、长方形、正方形。我们该如何绘制UML类图?



思考：结合飞机大战，我们该如何设计我们的敌机工厂和道具工厂?

实验步骤

6 重构代码，实现工厂模式

根据你所设计的UML类图重构代码，采用工厂模式创建三种敌机和三种道具。

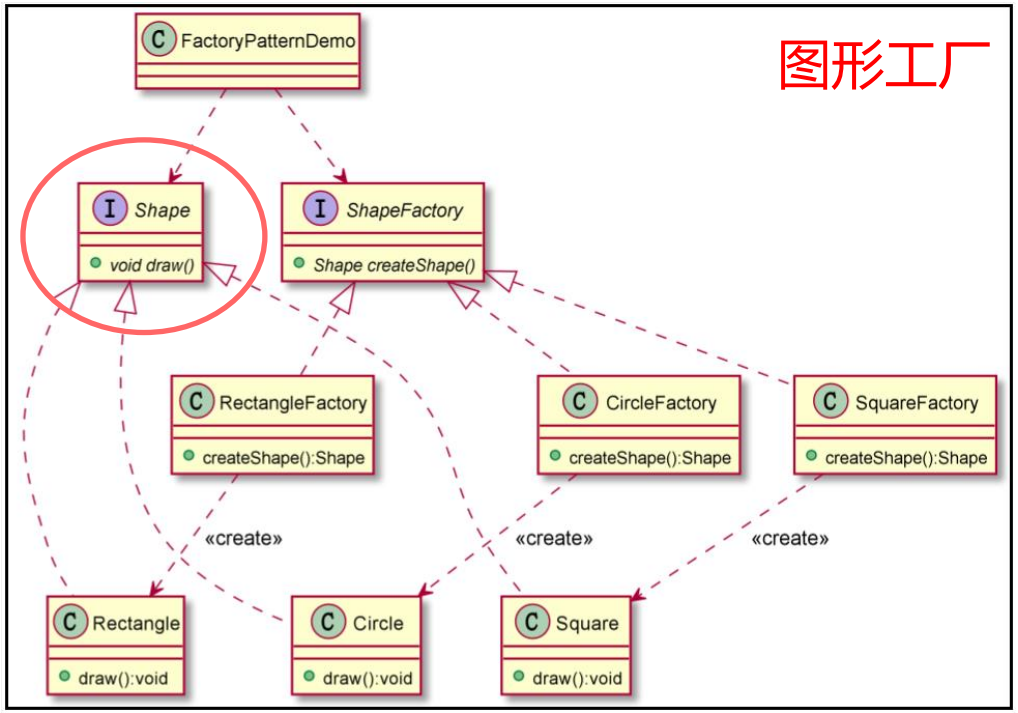
实验步骤

6 重构代码，实现工厂模式

- 工厂模式代码示例：

1、创建一个 Shape 接口

```
public interface Shape {
   void draw();
}
```



实验步骤

6 重构代码，实现工厂模式

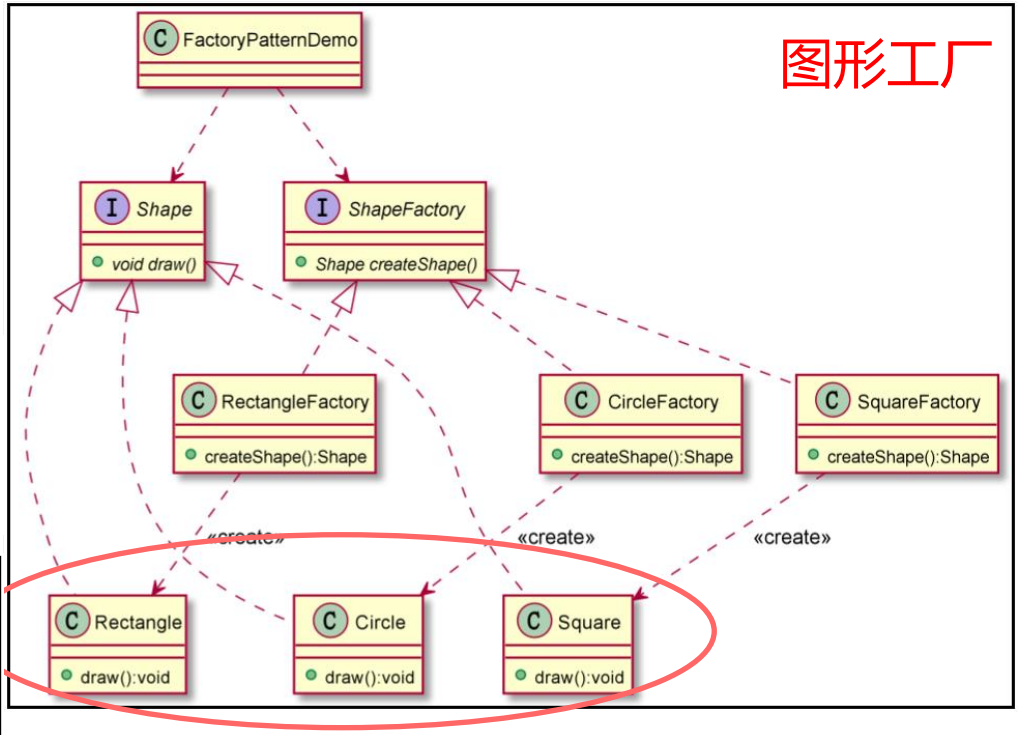
- 工厂模式代码示例：

2、创建实现 Shape 接口的实体类

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

```
public class Square implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

```
public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}
```



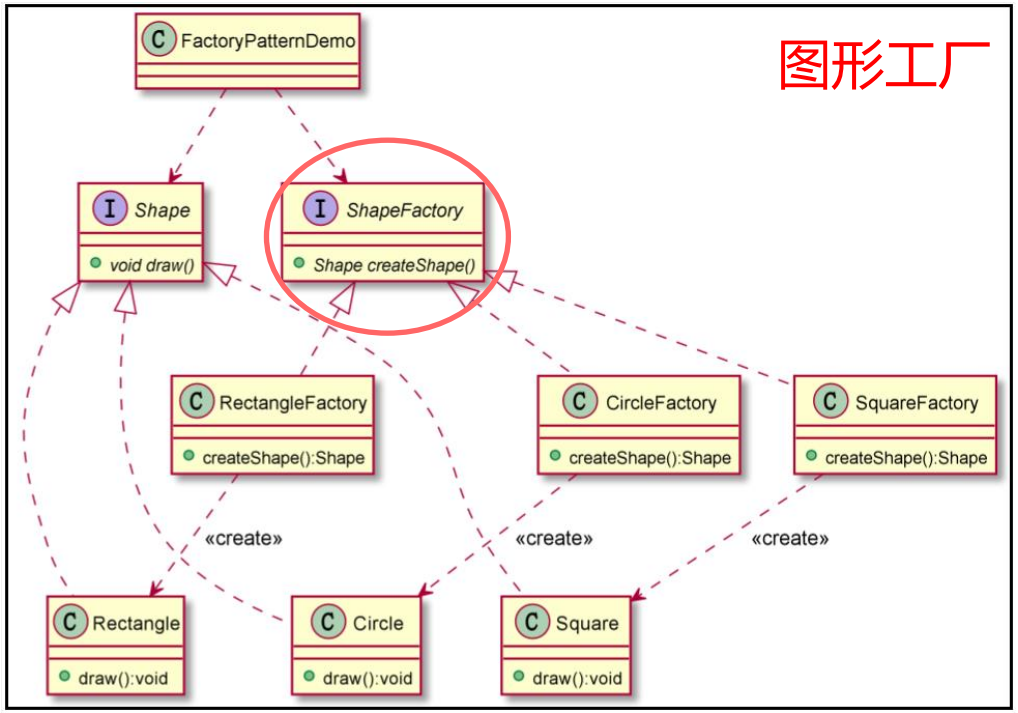
实验步骤

6 重构代码，实现工厂模式

- 工厂模式代码示例：

3、创建一个工厂ShapeFactory

```
public interface ShapeFactory {
    public Shape createShape();
}
```



实验步骤

6 重构代码，实现工厂模式

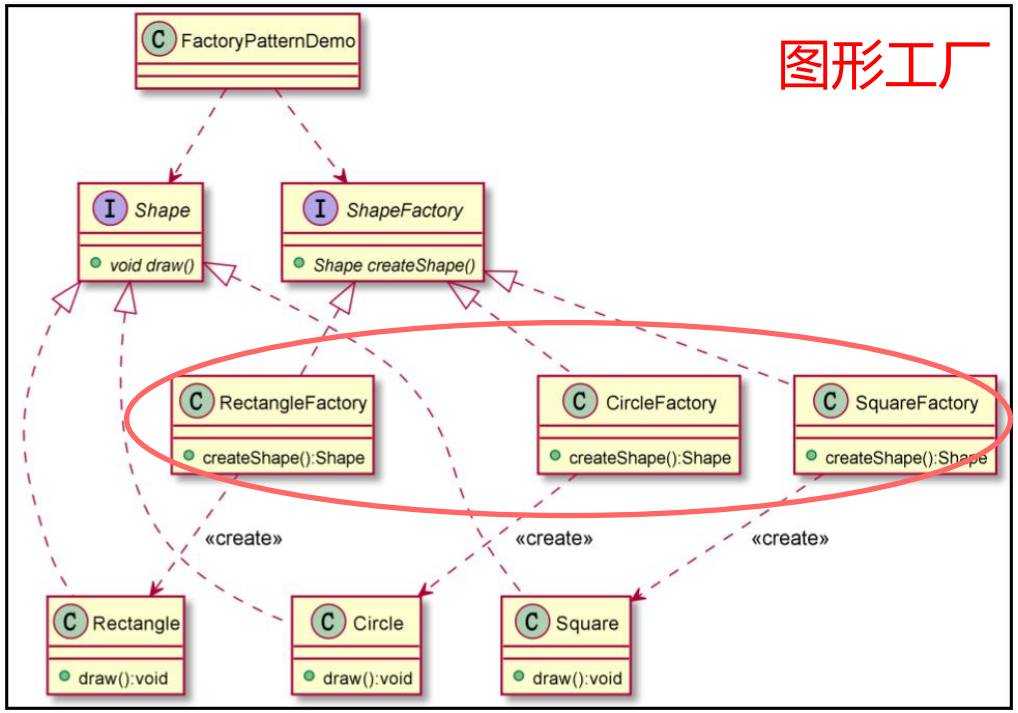
- 工厂模式代码示例：

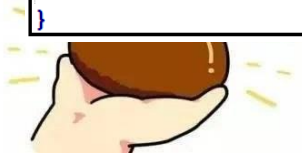
4、创建具体生产不同Shape的工厂类

```
public class RectangleFactory implements ShapeFactory {

   @Override
     public Shape createShape() {
         return new Rectangle();
     }
}
```

```
public class SquareFactory implements ShapeFactory {

   @Override
     public Shape createShape() {
         return new Square();
     }
}
```

```
public class CircleFactory implements ShapeFactory {

   @Override
     public Shape createShape() {
         return new Circle();
     }
}
```





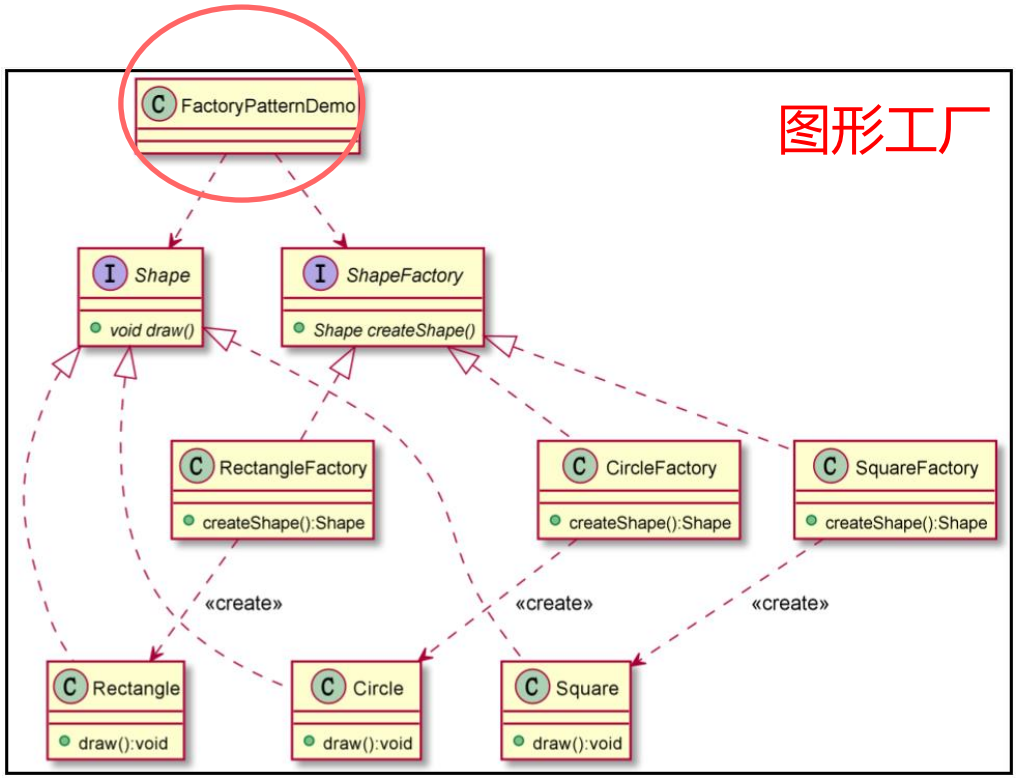
实验步骤

6 重构代码，实现工厂模式

- 工厂模式代码示例：

5、 FactoryPatternDemo 类使用 ShapeFactory 来获取不同的 Shape对象

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

   ShapeFactory shapeFactory ;
   Shape shape;

   //获取 Circle 的对象，并调用它的 draw 方法
   shapeFactory = new CircleFactory();
   shape = shapeFactory.createShape();
   shape.draw();

   //获取 Rectangle 的对象，并调用它的 draw 方法
   shapeFactory = new RectangleFactory();
   shape = shapeFactory.createShape();
   shape.draw();

   //获取 Square 的对象，并调用它的 draw 方法
   shapeFactory = new SquareFactory();
   shape = shapeFactory.createShape();
   shape.draw();

}
```




```
Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
```

作业提交

- 提交内容

① 项目压缩包（整个项目压缩成zip包提交，包含代码、uml图等）

② 截图报告

本实验无新增功能，重点考察对代码的重构。

- 截止时间

实验课后一周内提交至HITsz Grader 作业提交平台，具体截止日期参考平台发布。

登录网址：： http://grader.tery.top:8000/#/login

作业提交

实验二报告

一、 单例模式

1. 应用场景分析

描述飞机大战游戏中哪个应用场景需要用到此模式，目前实验一代码中存在的问题。

2. 解决方案

将 PlantUML 插件绘制的类图截图到此处，并对 UML 类图中每个类、接口，以及其关键属性和方法进行简单说明。

选择题

根据目的分类，单例模式和工厂模式属于哪种类型？

A．创建型模式

B．结构型模式

C．行为型模式

我就是看看你们有没有认真听课

补充说明

大家在开始实验二前先完成以下修改：

1、把basic下的FlyingObject改成AbstractFlyingObject；

2、把bullet下的AbstractBullet 改名为 BaseBullet；

3、按照刚才的命名修改UML类图，并把类图中原来构造函数的返回值都去掉。

实验二前需要完成的修改说明.pdf

同学们
请开始实验吧！