



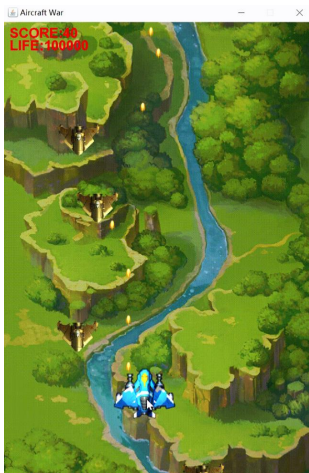
实验二：设计模式实验（1）

单例模式和工厂模式

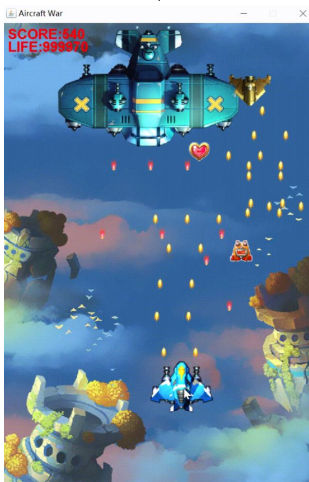
实验与创新实践教育中心 · 计算机与数据技术实验教学部

本学期实验总体安排

初始版本



最终版本



游戏主界面
英雄机移动
英雄机子弹直射
碰撞检测
统计得分和生命值

重构代码，采用**单例模式**
创建英雄机
重构代码，采用**工厂模式**
创建敌机和道具

重构代码，采用**策略模式**
实现不同弹道发射
采用**数据访问对象模式**
实现得分排行榜

采用**观察者模式**
实现炸弹道具生效
采用**模板模式**
实现三种游戏难度

初始版本

01

绘制UML类图

创建精英敌机并直射子弹
精英敌机随机掉落三种道具
加血道具生效

02

03

添加JUnit单元测试
创建Boss和其它敌机

04

05

使用**Swing**添加游戏难度选择和
排行榜界面
使用**多线程**实现音效的开启/关闭、
及火力道具

06

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	2	4	4
实验内容	飞机大战 功能分析	单例模式 工厂模式	Junit 单元测试	策略模式 数据访问对 象模式	Swing 多线程	观察者模式 模板模式
分数	4	6	4	6	6	14 (6+8)
提交内容	UML类图、 代码	UML类图、 代码	测试报告、 代码	UML类图、 代码	代码	项目代码、实 验报告、展示 视频

实验课程共**16**个学时，**6**个实验项目，总成绩为**40分**。

CONTENTS

目录

01 实验目的

02 实验任务

03 实验原理

04 实验步骤

实验目的

难度	知识点
理解	单例模式和工厂模式的模式动机和意图
掌握	单例和工厂模式UML结构图的绘制方法
熟练	使用Java语言，编码实现单例和工厂模式

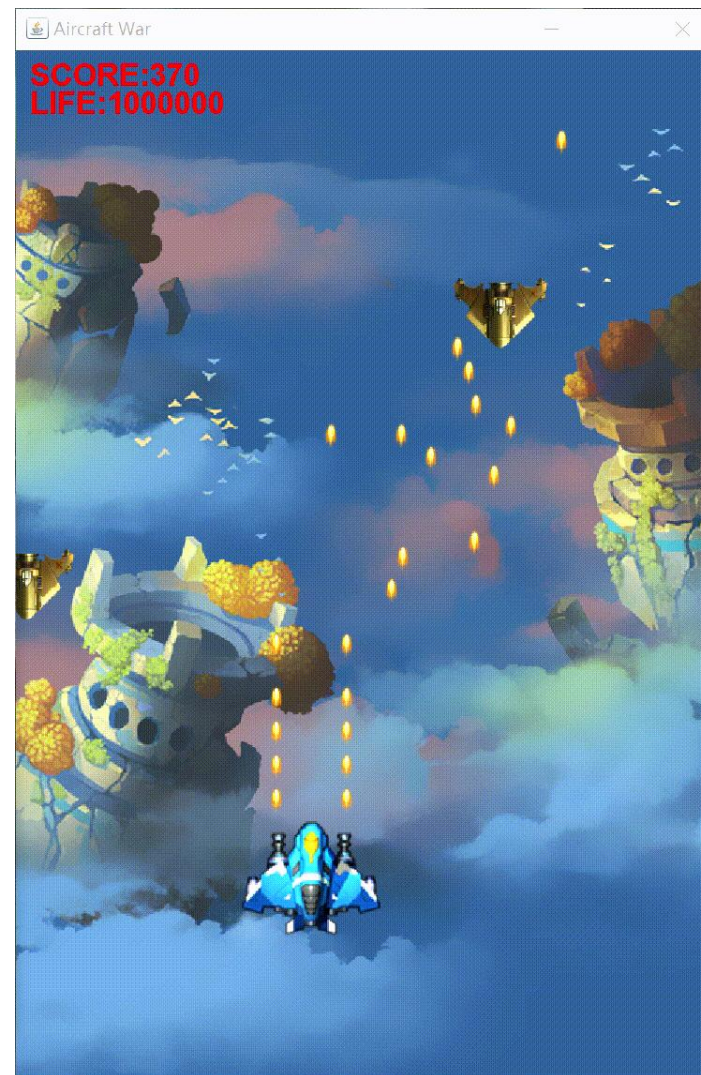


实验任务

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. 采用工厂模式创建普通、精英敌机以及三种道具。

注意：先“设计”再编码！请结合飞机大战实例，完成模式UML类图设计后，再进行编码。



课前小测



请选择：

若根据模式**目的**来分类，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

答案：A

创建型模式关注对象的创建过程，目标是**将对象的创建和使用分离**，降低耦合度。

实验原理：场景分析（1）

英雄机 创建场景 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**，由玩家通过鼠标控制其移动。



实验原理：场景分析（1）



请思考：

1. 目前代码在哪个类创建英雄机？如何创建？是否符合面向对象设计原则？

```
public Game() {  
    heroAircraft = new HeroAircraft(  
        locationX: Main.WINDOW_WIDTH / 2,  
        locationY: Main.WINDOW_HEIGHT - ImageManager.HERO_IMAGE.getHeight() ,  
        speedX: 0, speedY: 0, hp: 1000);  
}
```

违反
单一职责

X

2. 目前能否保证英雄机的唯一性？

不能，外部程序可以随意用new方法创建一个实例。

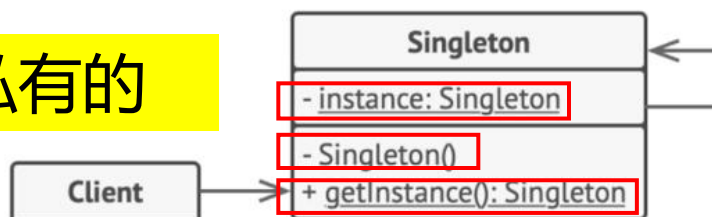
实验原理：单例模式结构图

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。

关键代码1：构造函数是私有的

1 单例 (Singleton) 类声明了一个名为 `getInstance` 获取实例 的静态方法来返回其所属类的一个相同实例。

单例的构造函数必须对客户端 (Client) 代码隐藏。调用 获取实例 方法必须是获取单例对象的唯一方式。



关键代码2：提供一个访问该类唯一实例的方法

```
if (instance == null) {
    // 注意：如果程序需要支持多线程，
    // 你必须在此放置线程锁。
    instance = new Singleton()
}
return instance
```

单例模式结构图

实验原理：单例模式代码实现

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;  
    }  
}
```

③ 双重检查锁定 (DCL)

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



只有敲代码才能
感受到温暖

实验原理：场景分析（2）

敌机、道具
创建场景
分析

游戏中有**3种类型敌机**：普通敌机、精英敌机、Boss敌机。



游戏中有**3种类型道具**：火力道具、炸弹道具、加血道具。



实验原理：场景分析（2）



请思考：

1. 目前在哪个类创建敌机？如何创建？是否符合面向对象设计原则？

```
if (enemyAircrafts.size() < enemyMaxNumber) {  
    enemyAircrafts.add(new MobEnemy(  
        (int) (Math.random() * (Main.WINDOW_WIDTH - ImageManager.MOB_ENEMY_IMAGE.getWidth())),  
        (int) (Math.random() * Main.WINDOW_HEIGHT * 0.05),  
        speedX: 0,  
        speedY: 10,  
        hp: 30  
    ));  
}
```

Game类

违反
单一职责

X

违反
开闭原则

X

违反
依赖倒转

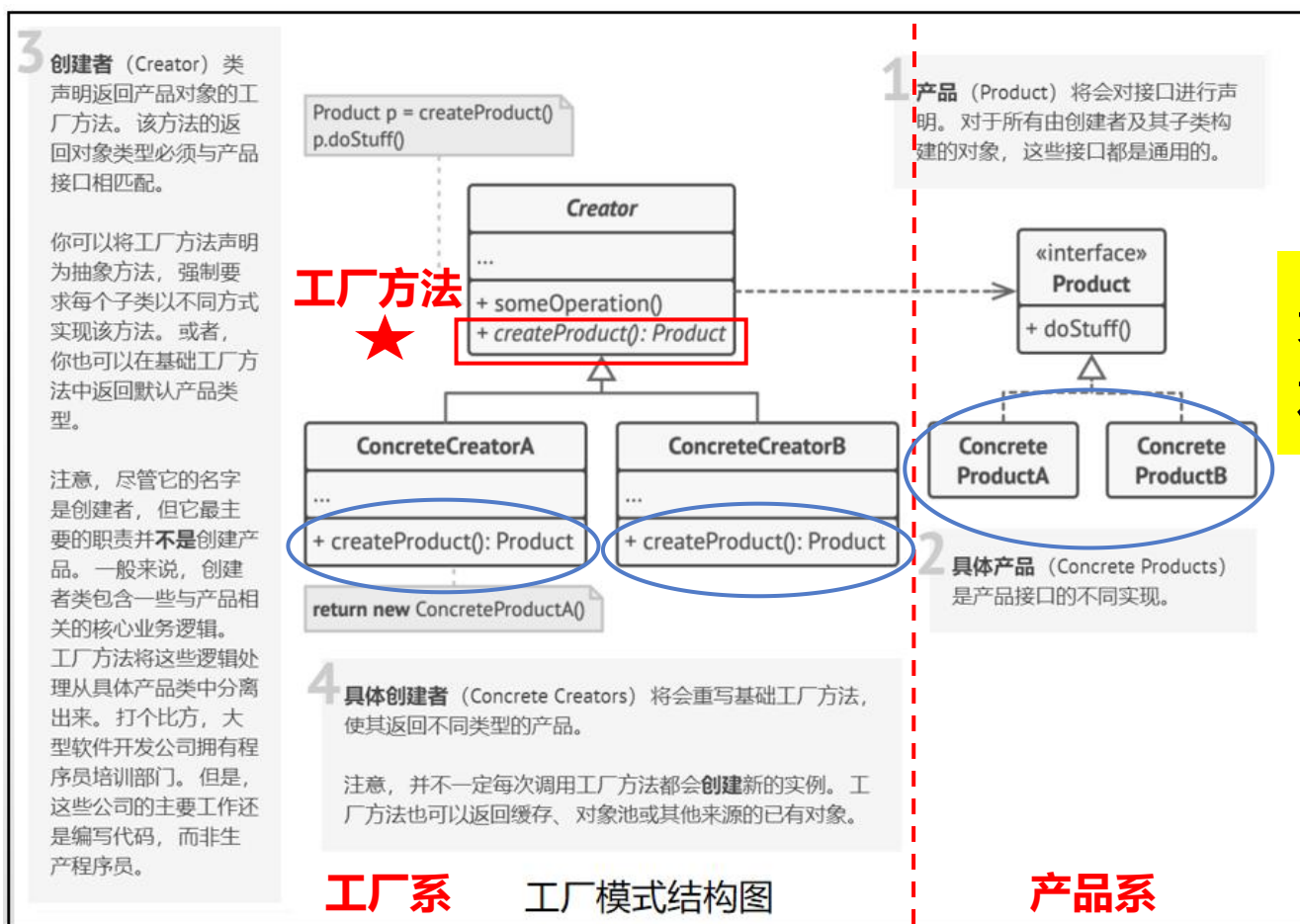
X

针对**接口**编程, 而不是针对实现编程!

3. 若增加Boss机或其它多种新型敌机，需要改动哪些代码？

实验原理：工厂模式结构图

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。



实验步骤：工厂模式



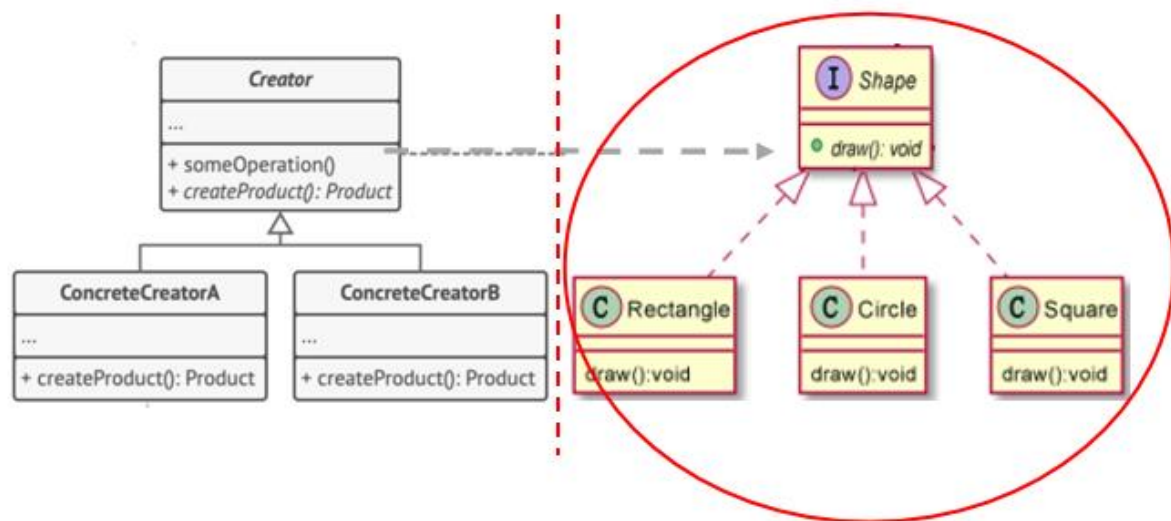
实验步骤：图形工厂举例（工厂模式）

假如我们要开发一个**图形工厂**，
可生产3种类型的图形产品：圆形、
长方形、正方形。我们该如何用工
厂模式实现呢？



实验步骤：图形工厂举例（1）

产品系： 创建 Shape 接口和实现该接口的三个图形实体类。



① 创建Shape 接口, 充当产品角色;

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

② 创建三个图形实体类, 充当具体产品角色;

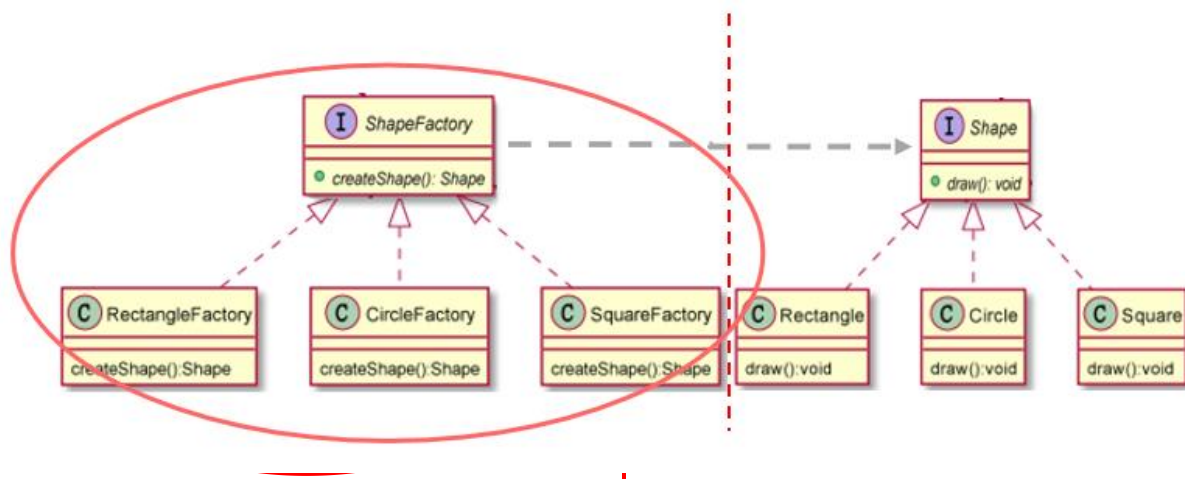
```
public class Rectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside Rectangle::draw() method.");
    }
}
```

```
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：图形工厂举例（2）

工厂系：创建工厂接口 ShapeFactory 和实现该接口的三个具体工厂实体类。



③ 创建工厂接口 ShapeFactory，充当创建者角色；

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

④ 创建三个工厂实体类，充当具体创建者角色。

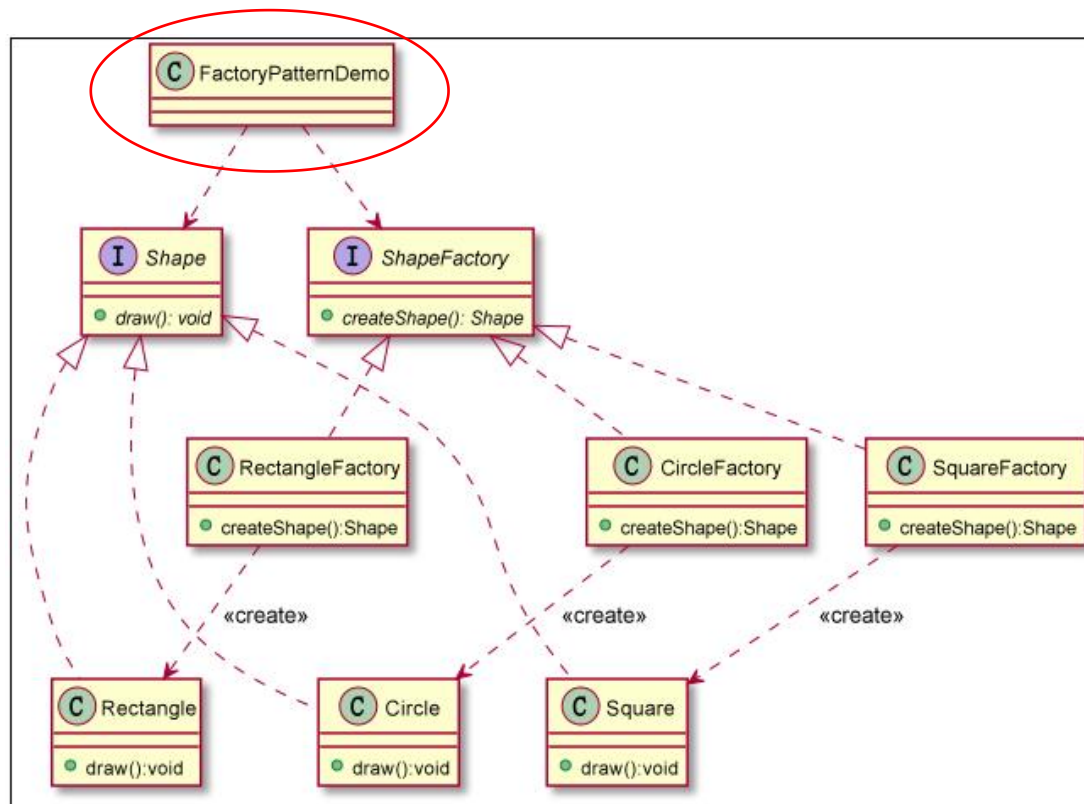
```
public class RectangleFactory implements ShapeFactory {  
    @Override  
    public Shape createShape() {  
        return new Rectangle();  
    }  
}
```

```
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：图形工厂举例（3）

⑤ 客户端 **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();
```

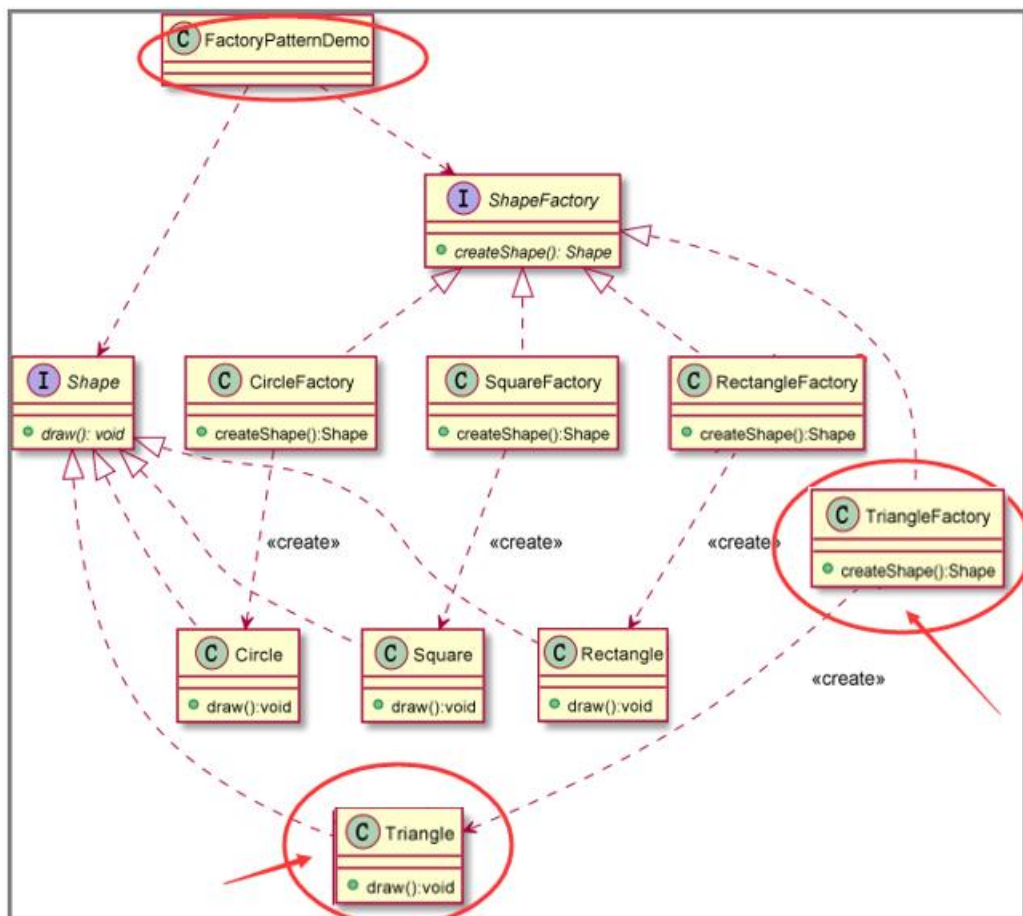
```
}
```

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

实验步骤：图形工厂举例（4）



请思考：如何添加一个三角形图形？



```
public class FactoryPatternDemo {

    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();

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

    }
}
```



开闭原则

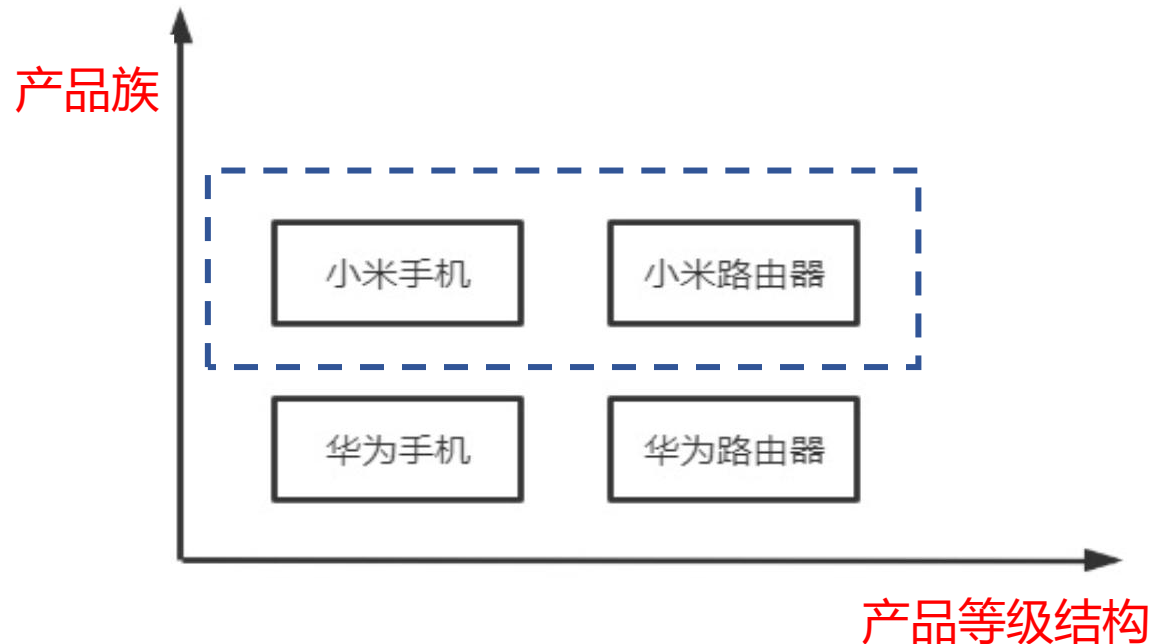
思考题



若采用**抽象工厂模式**创建敌机和道具是否合适？

☐ A . 合适

☐ B . 不合适



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 实验截图报告（设计模式类图和说明，请使用报告模板）

本实验无新增功能，重点考察类图绘制、代码重构。

- 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>



同学们，请开始实验吧！

THANK YOU