



哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

規格嚴格



功夫到家

设计模式实验 (1)

实验二：单例模式和工厂模式

2023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	2	4	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

01

实验目的

02

实验任务

03

实验步骤

04

作业提交



实验目的

- 理解单例模式和工厂模式的意义，掌握模式结构
- 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. 采用工厂模式创建普通和精英敌机以及三种道具。

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



选择题

根据**目的**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



实验步骤

1 英雄机应用场景分析

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，由玩家
通过鼠标控制移动。





实验步骤

1 英雄机应用场景分析

请思考：

1. 目前在哪个类创建英雄机？如何创建？
2. 是否符合面向对象设计原则？
3. 目前能否保证英雄机的唯一性？



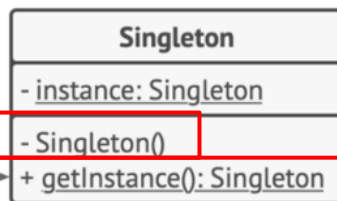
实验步骤

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。

关键代码1：构造函数是私有的

Client



关键代码2：提供一个访问该类唯一实例的方法

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。

```
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。



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

应用场景
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。





实验步骤

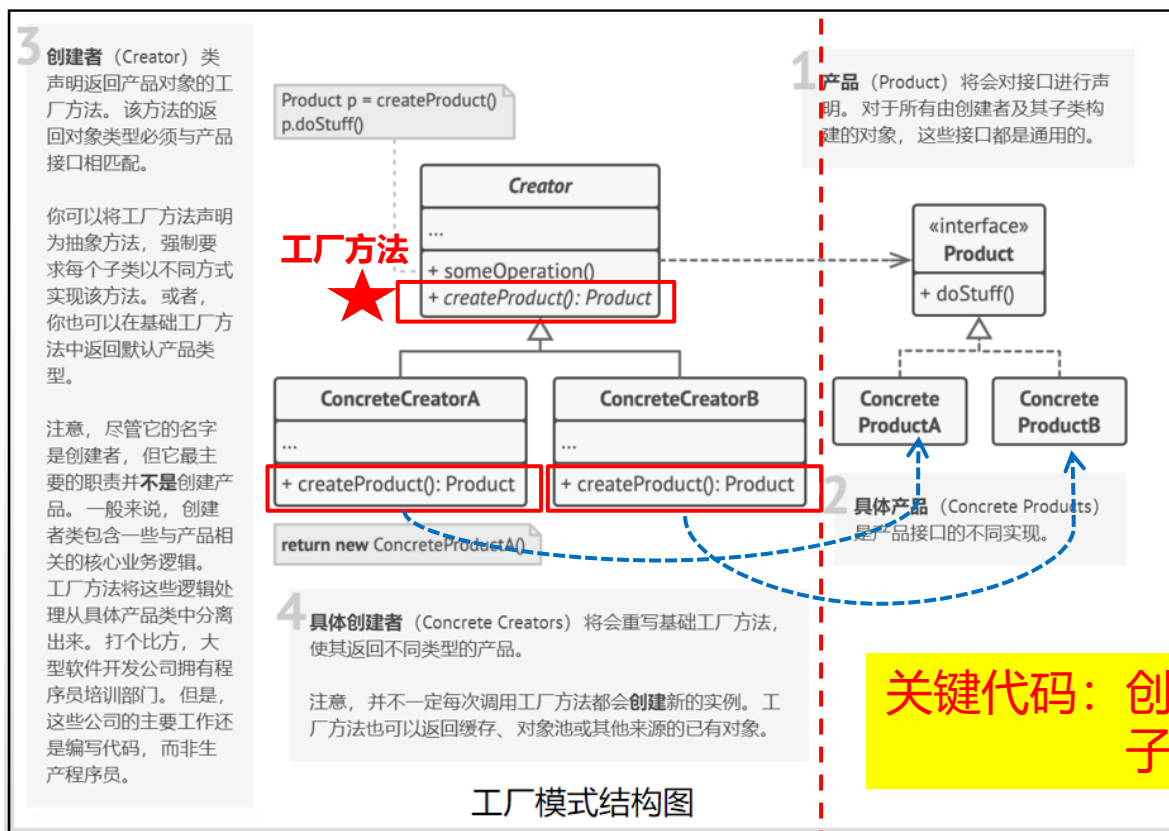
4 敌机和道具应用场景分析

请思考（以敌机为例）：

1. 目前在哪个类创建敌机？如何创建？是否合理？
2. 若敌机增加一种属性“防御力”，需要改动哪些代码？
3. 若增加Boss机或其它多种新型敌机，需要改动哪些代码？

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



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？



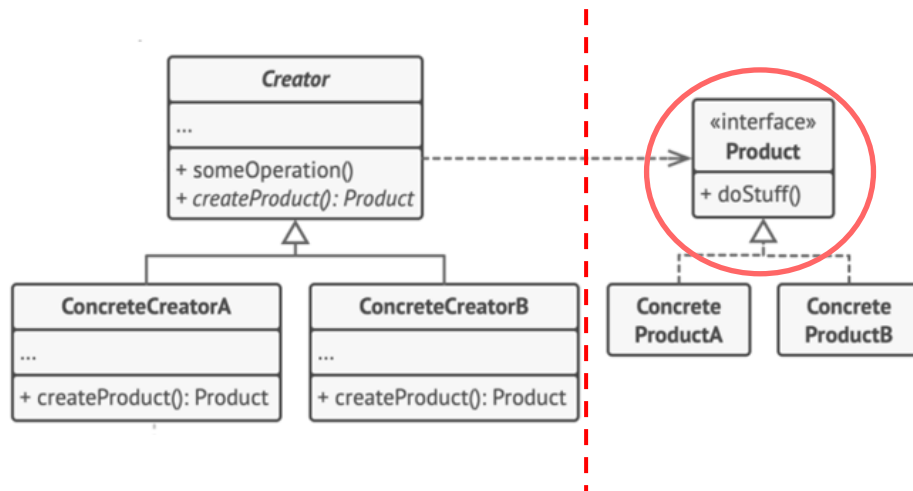


实验步骤

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？



产品系：创建一个 **Shape 接口** 和实现 Shape 接口的 **实体类**。

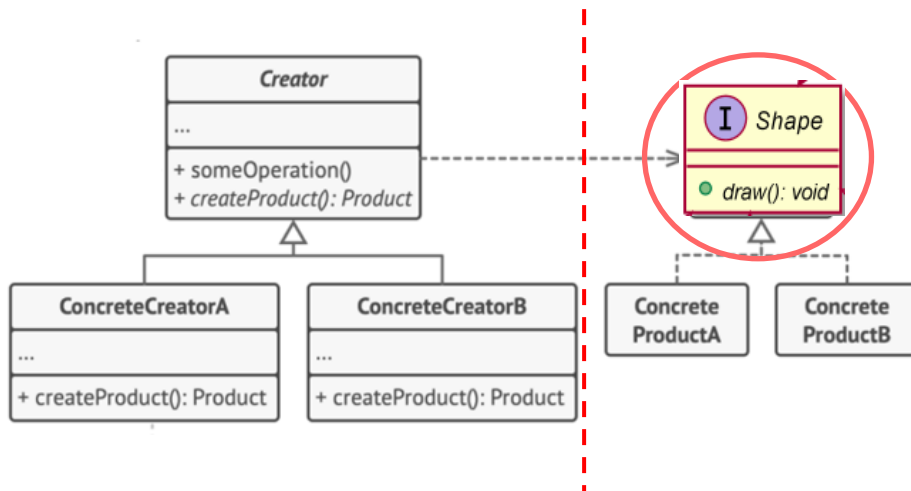


实验步骤

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？



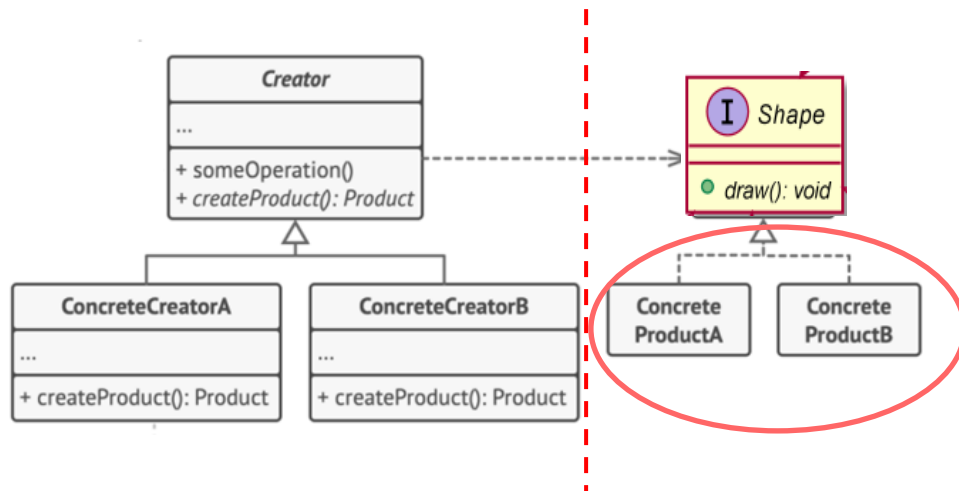


实验步骤

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？



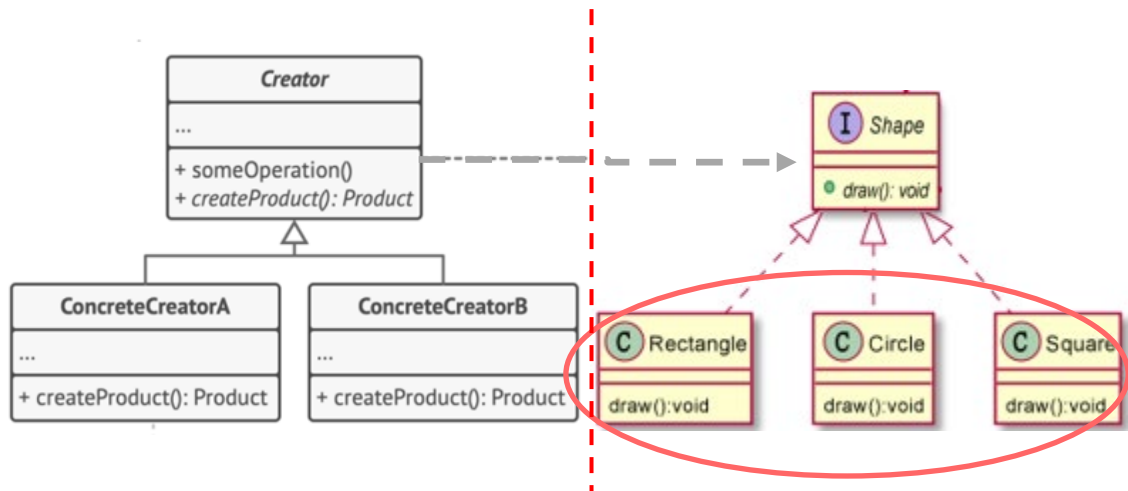


实验步骤

5

绘制工厂模式类图

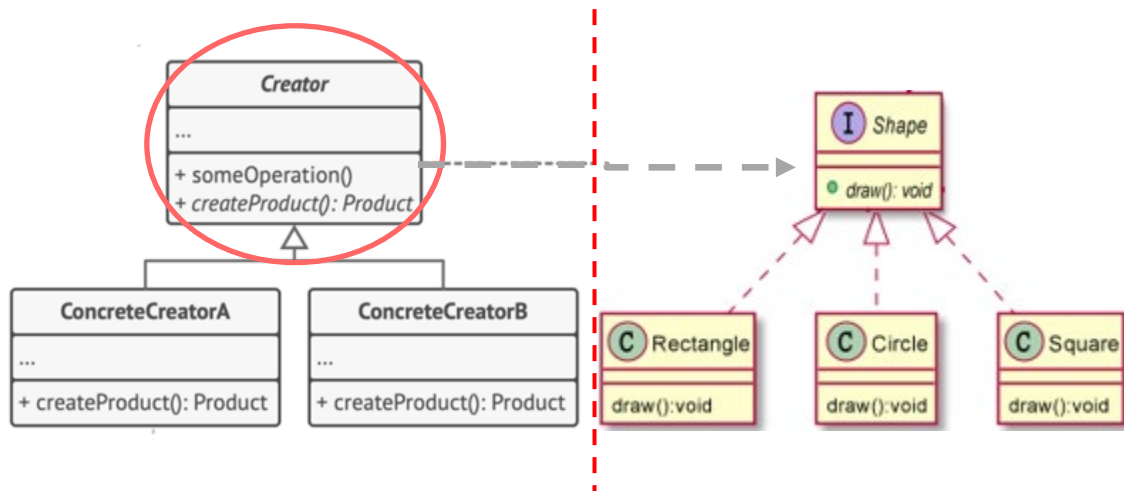
假如我们要建一个
图形工厂，生产3种产
品：圆形、长方形、
正方形。我们该如何
绘制UML类图？



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？



工厂系：定义**工厂接口** **ShapeFactory**和实现该接口的**具体工厂**实体类。

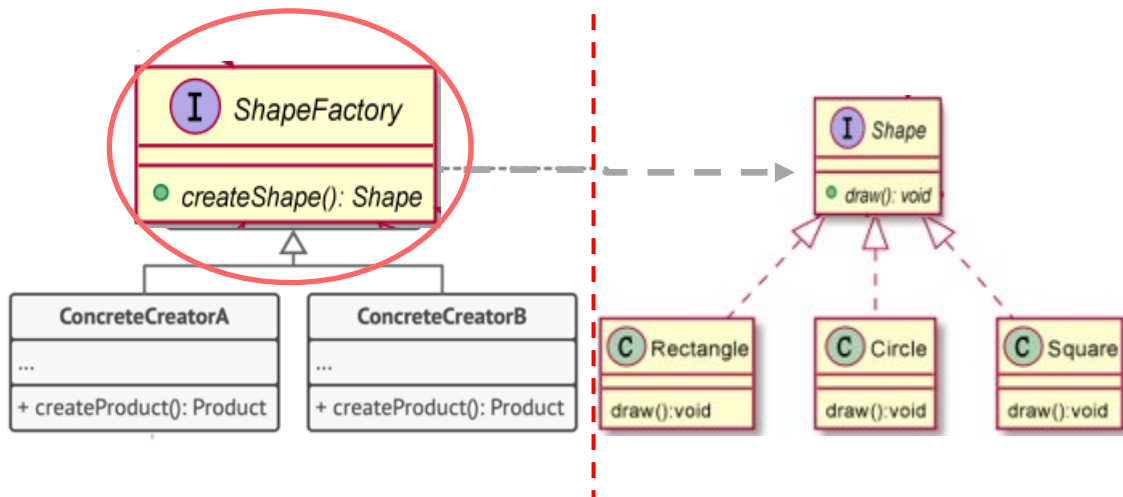


实验步骤

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？



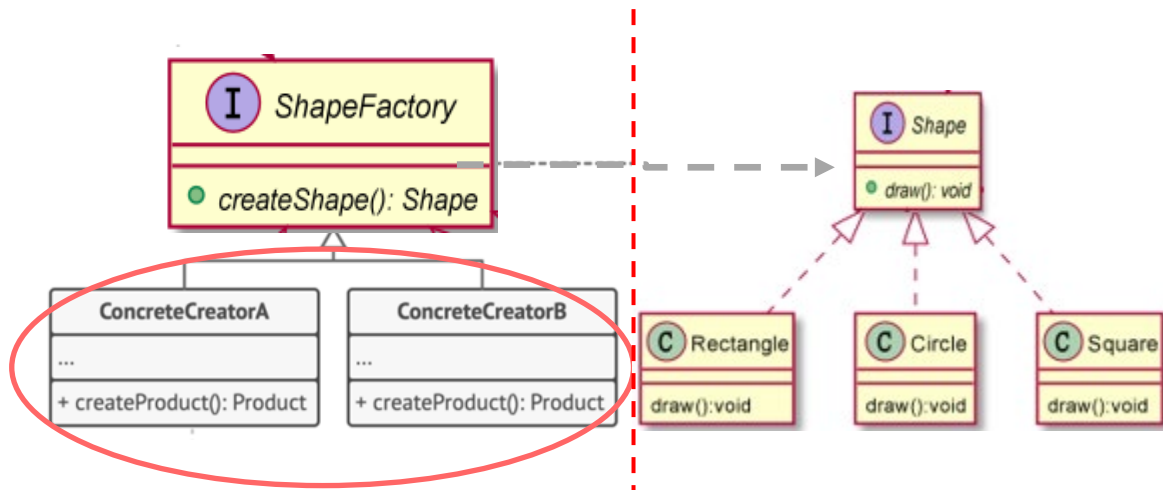


实验步骤

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？



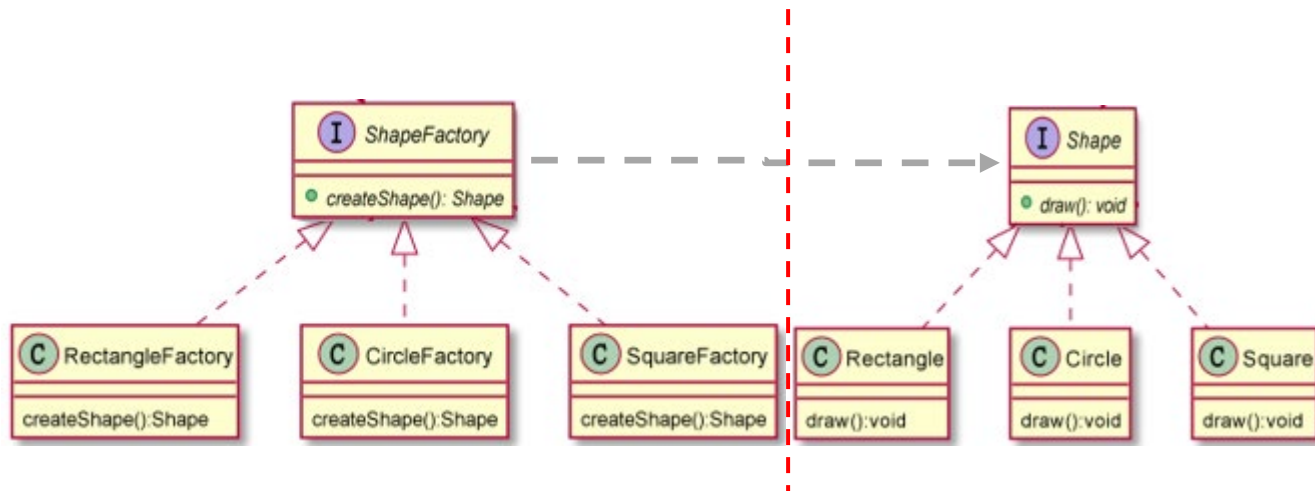


实验步骤

5

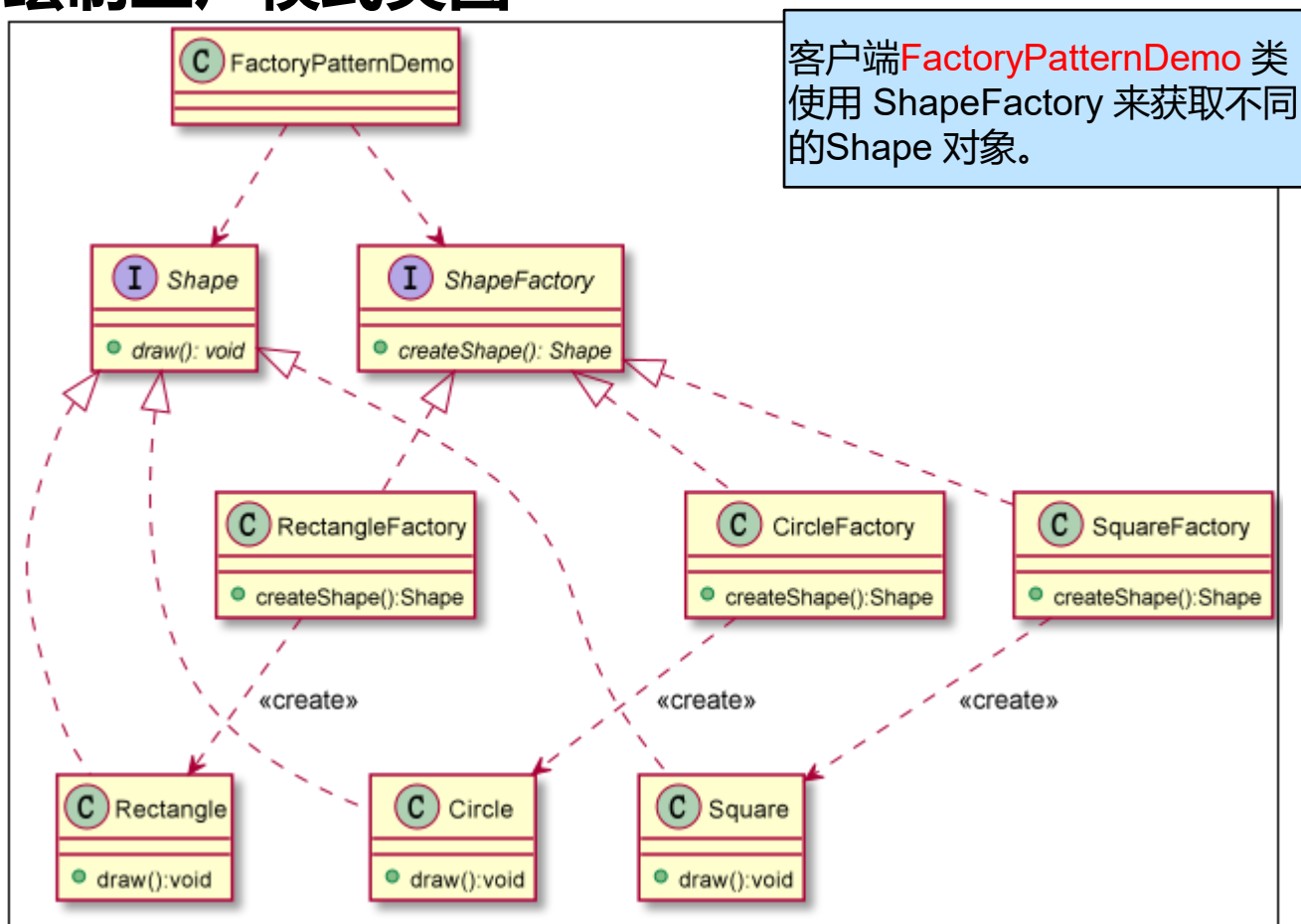
绘制工厂模式类图

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？



5

绘制工厂模式类图



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**创建普通、精英两种**敌机**，以及三种**道具**。





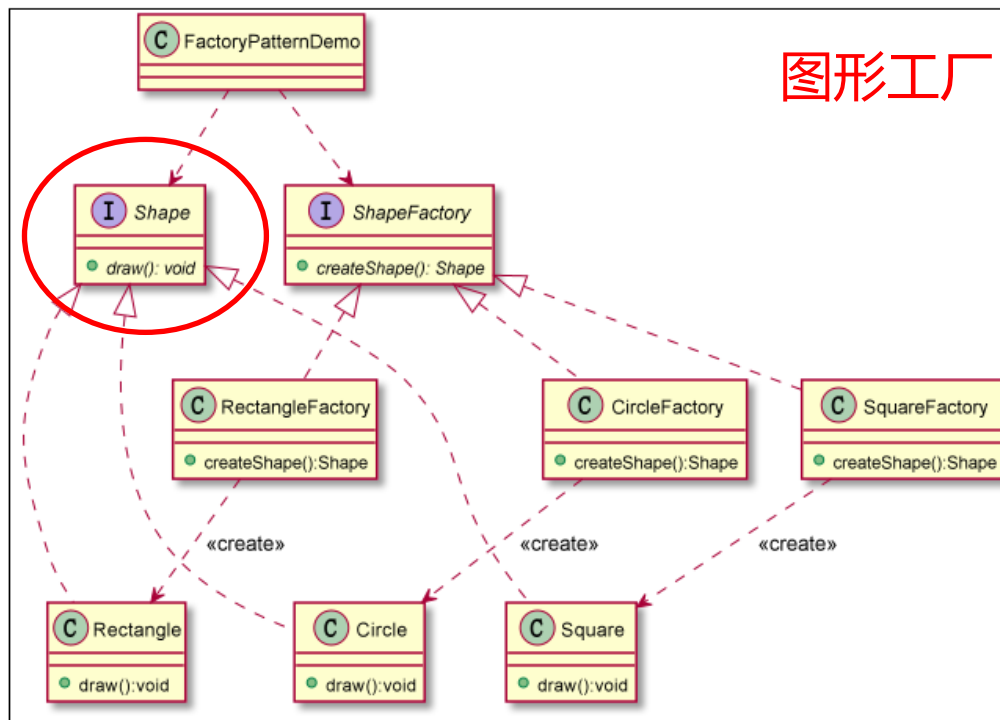
实验步骤

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，充当产品角色，也可用抽象类实现。

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



实验步骤

6 重构代码，实现工厂模式

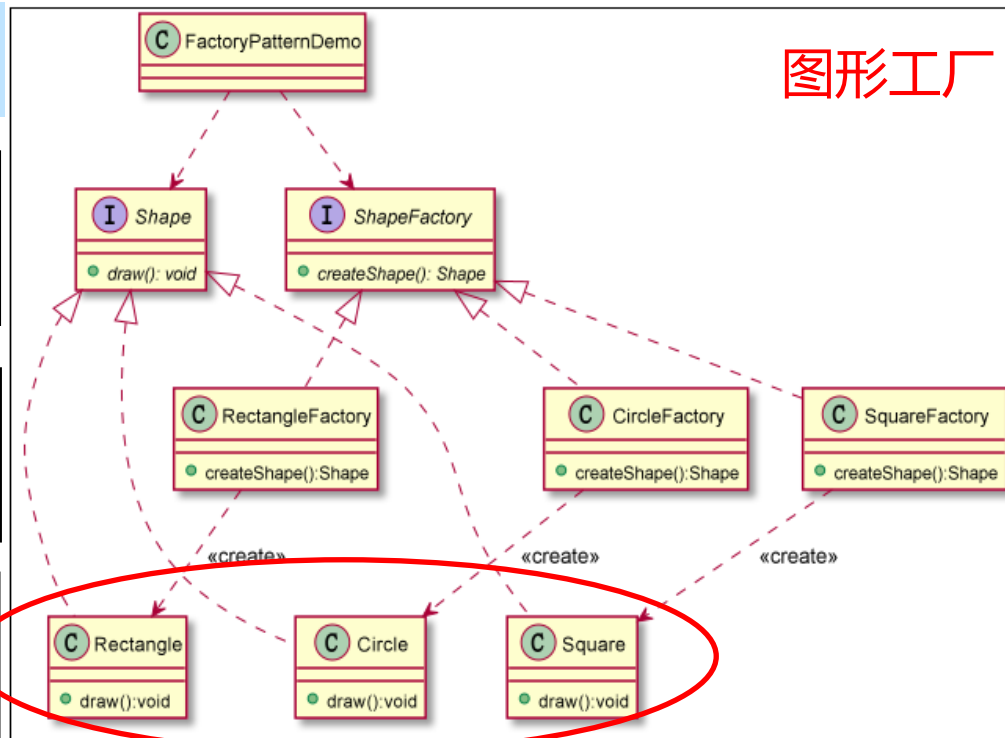
● 工厂模式代码示例：

② 创建实现Shape 接口的实体类，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.");  
    }  
}
```





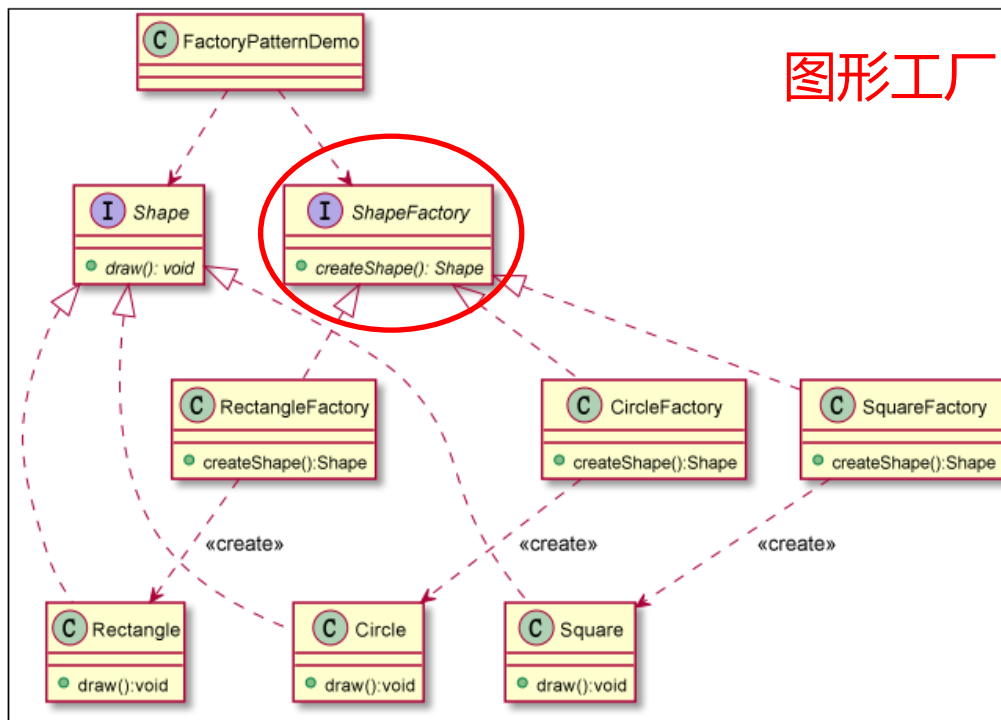
实验步骤

6 重构代码，实现工厂模式

- 工厂模式代码示例：

③ 创建一个工厂接口 **ShapeFactory**，充当创建者角色。

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



6 重构代码，实现工厂模式

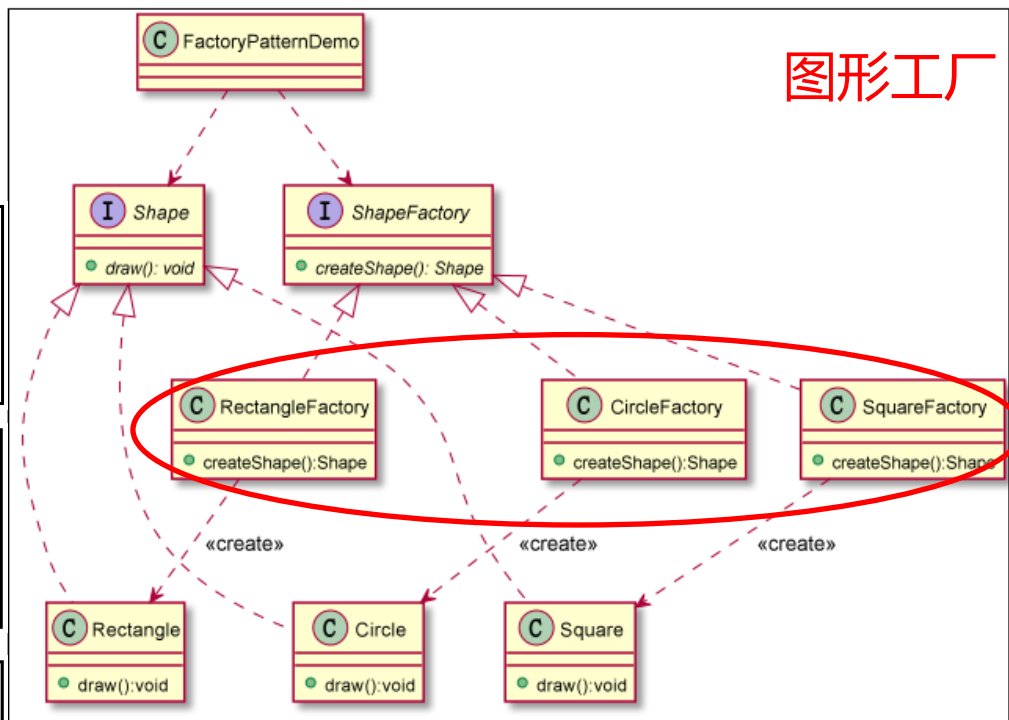
● 工厂模式代码示例：

④ 创建具体生产不同Shape的工厂类，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();  
    }  
}
```



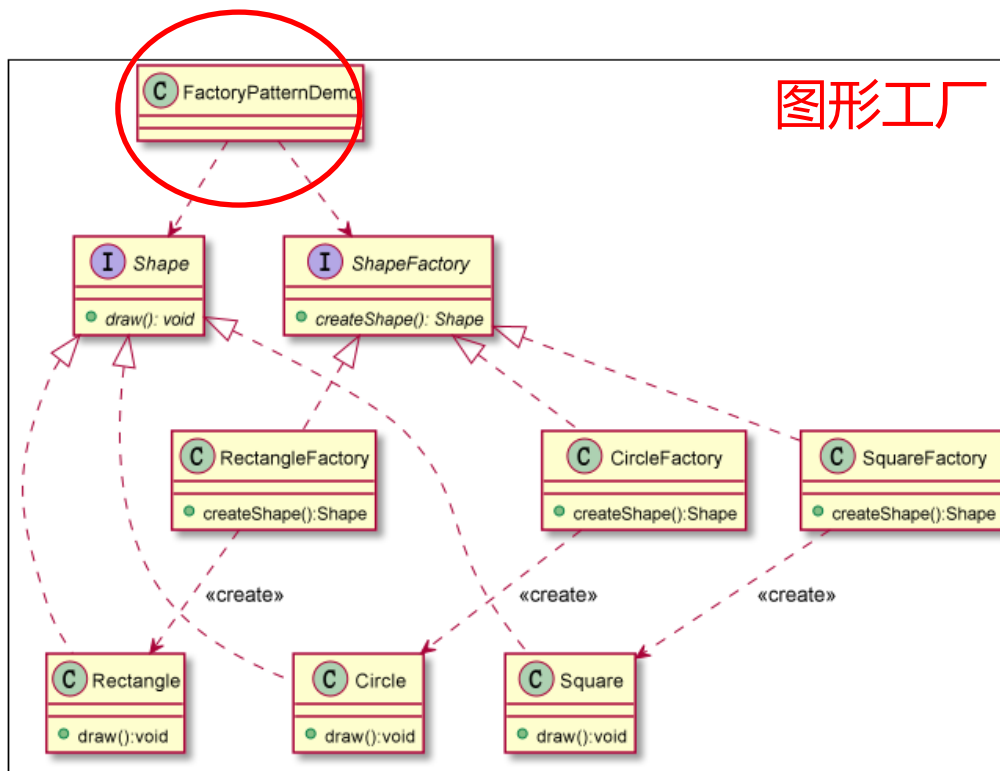
实验步骤

6 重构代码，实现工厂模式

● 工厂模式代码示例：

- 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();  
}
```

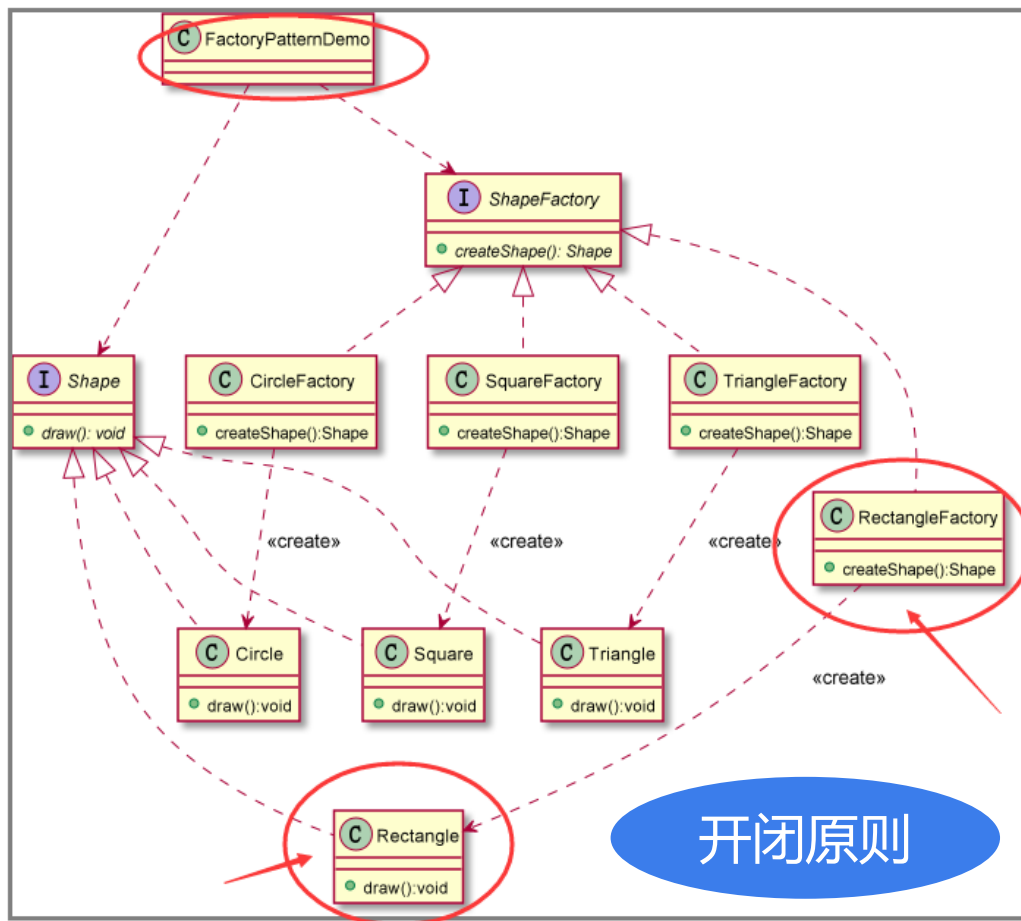


执行结果

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

实验步骤

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();

    }
}
```




思考题

若采用抽象工厂模式创建三种敌机和三种道具是否合适？

 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，一定要用报告模板）
- 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>



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，目前代码实现中存在的问题。

2. 解决方案

借鉴单例模式的解题思路，设计解决该场景问题的方案。

- 将 PlantUML 插件绘制的类图截图到此处
- 描述你设计的 UML 类图中的每个角色（类、接口），并对它的关键属性、方法和作用进行简要说明。



作业提交

为方便批改，请同学们将参数做如下修改：

- ① 增大道具的掉落几率，比如30%掉落火力道具、30%掉落加血道具、30%掉落炸弹道具，还有10%不掉落道具；
- ② 英雄机血量上限设置为1000。



**同学们
请开始实验吧！**