# 设计模式实验（1）

# 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

# 实验二任务

1. 结合实例，绘制单例模式的UML结构图（类图）。
2. 重构代码，采用单例模式创建英雄机。
3. 结合实例，绘制工厂模式的UML结构图（类图）。
4. 重构代码，采用工厂模式创建三种敌机和三种道具。

# 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

# 实验步骤

## 1 绘制单例模式结构图

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。英雄机通过生命值（血）生存，被敌机子弹击中损失部分生命值，被敌机碰撞则全部损失。英雄机生命值为0时判定游戏结束。

# 实验步骤

## 1 绘制单例模式结构图

**单例模式**（Singleton Pattern）是一种创建型设计模式， 让你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：构造函数是私有的


单例模式结构图

# 实验步骤

## 2 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。

- 单例模式代码实现方式示例（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

## 2 重构代码，实现单例模式

- 单例模式的代码实现方式示例（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

## 3 绘制工厂模式结构图

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。普通敌机和精英敌机以一定频率在界面随机位置出现并向屏幕下方移动。Boss敌机悬浮于界面上方，直至被消灭。

游戏中还有3种类型道具：火力道具、炸弹道具、加血道具。敌机坠毁后，以较低概率随机掉落某种道具。英雄机通过碰撞道具后，道具自动触发生效。

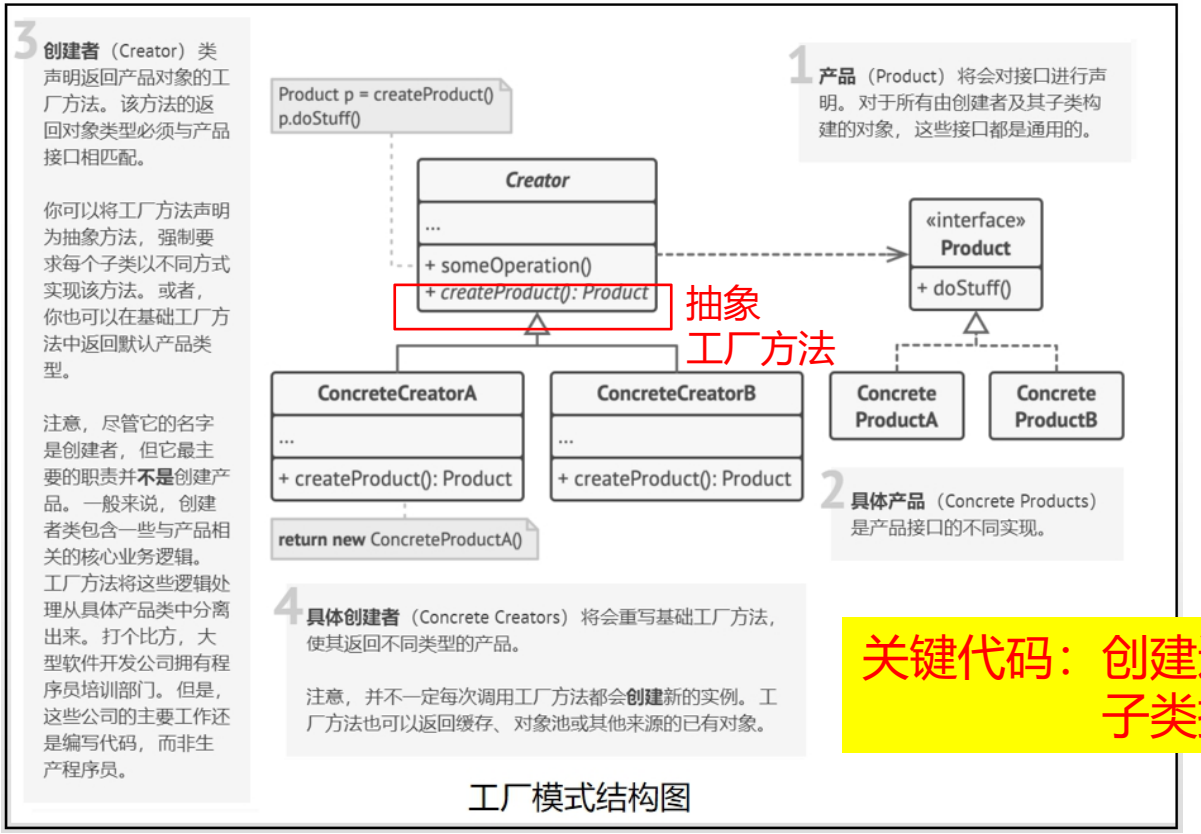





# 实验步骤

## 3 绘制工厂模式结构图

**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， 允许子类决定实例化对象的类型。



工厂模式结构图

# 实验步骤

## 4 重构代码，实现工厂模式

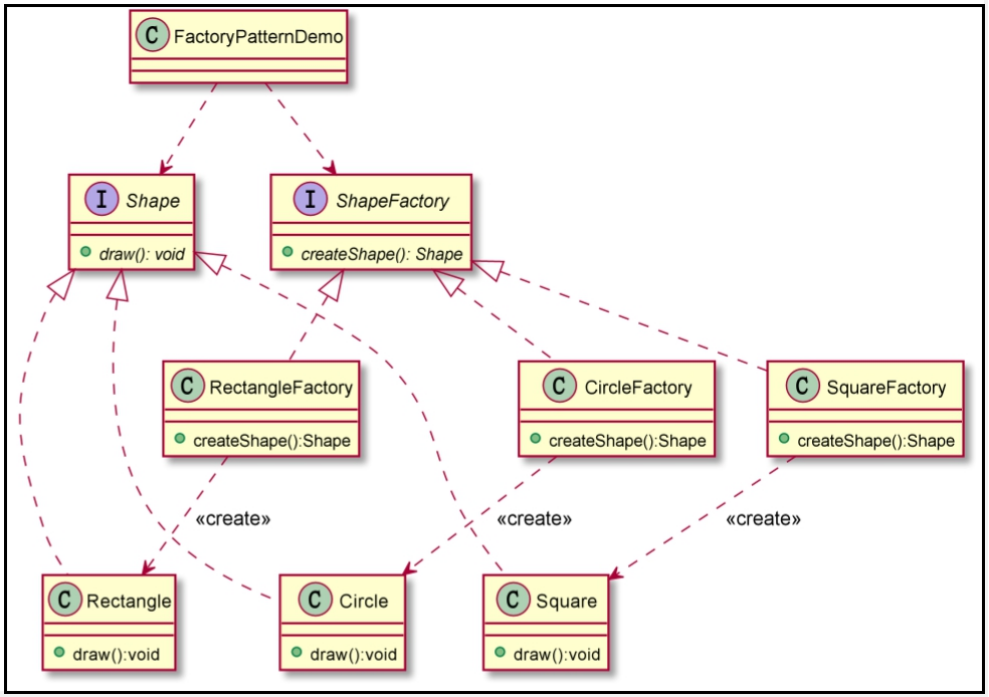
根据你所设计的UML类图，重构代码，采用工厂模式创建三种敌机和三种道具。

- 工厂模式代码实现方式示例：

创建一个 Shape 接口和实现 Shape 接口的实体类。

下一步是定义工厂接口 ShapeFactory。

FactoryPatternDemo 类使用 ShapeFactory 来获取不同的Shape 对象。



# 实验步骤

## 4 重构代码，实现工厂模式

- 工厂模式的代码实现方式示例：

① 创建一个接口，充当产品角色。(也可以用抽象类)

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

② 创建实现接口的实体类，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

## 4 重构代码，实现工厂模式

- 工厂模式的代码实现方式示例：

③ 创建一个工厂接口，充当创建者角色。

```
public interface ShapeFactory {

    public abstract Shape createShape();

}
```

④ 创建实现工厂接口的具体工厂类，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

## 4 重构代码，实现工厂模式

- 工厂模式的代码实现方式示例：

⑤ 使用 FactoryPatternDemo 来演示工厂模式的用法。

⑥ 执行程序，输出结果。

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

```
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();

}
```

# 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 结合飞机大战实例，提交你所绘制的单例模式和工厂模式UML类图（随代码提交并截图至报告模板中）。

② 提交重构后的代码，代码运行正确。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。

# 同学们
# 请开始实验吧！