哈尔滨工业大学（深圳）

面向对象的软件构造导论
实验指导书

实验二 设计模式实验（1）

—— 单例模式和工厂模式

2024 春

目录

1. 实验目的

1. 理解单例模式和工厂模式的模式动机和意图，掌握模式结构；
2. 结合实例，熟练绘制单例和工厂两种模式的 UML 结构图；
3. 重构代码，熟练使用代码实现单例和工厂两种模式。

2. 实验环境

1. Windows 10
2. IntelliJ IDEA 2023.3.4
3. OpenJDK 20

3. 实验内容（2 学时）

（1）结合实例，绘制单例模式的 UML 结构图；
（2）根据类图，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；
（4）根据类图，重构代码，采用工厂模式创建普通和精英两种敌机，以及三种道具。

4. 实验步骤

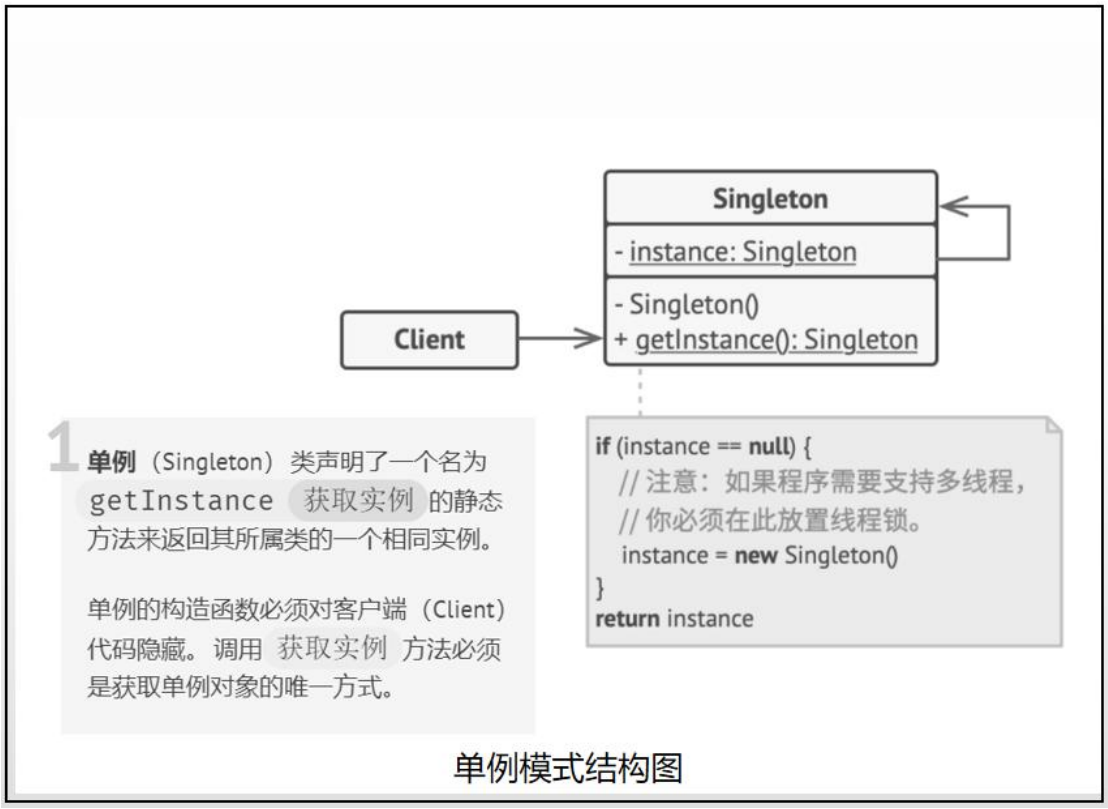
4.1 结合飞机大战实例，绘制单例模式的 UML 结构图

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 时判定游戏失败。

请结合该实例场景，为创建英雄机绘制单例模式的 UML 结构图，要求给出设计模式的名称，类名、方法名和属性名可自行定义。

单例模式

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。



单例模式结构图

请参考以上 UML 结构图，绘制飞机大战中的单例模式。

4.2 根据设计的类图，重构代码，实现单例模式

根据 4.1 中你所设计的 UML 类图，重构代码，采用单例模式创建英雄机。

单例模式的代码示例（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;
        }
    }
```

（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;
    }
}
```

4.3 结合飞机大战实例，绘制工厂模式的 UML 结构图

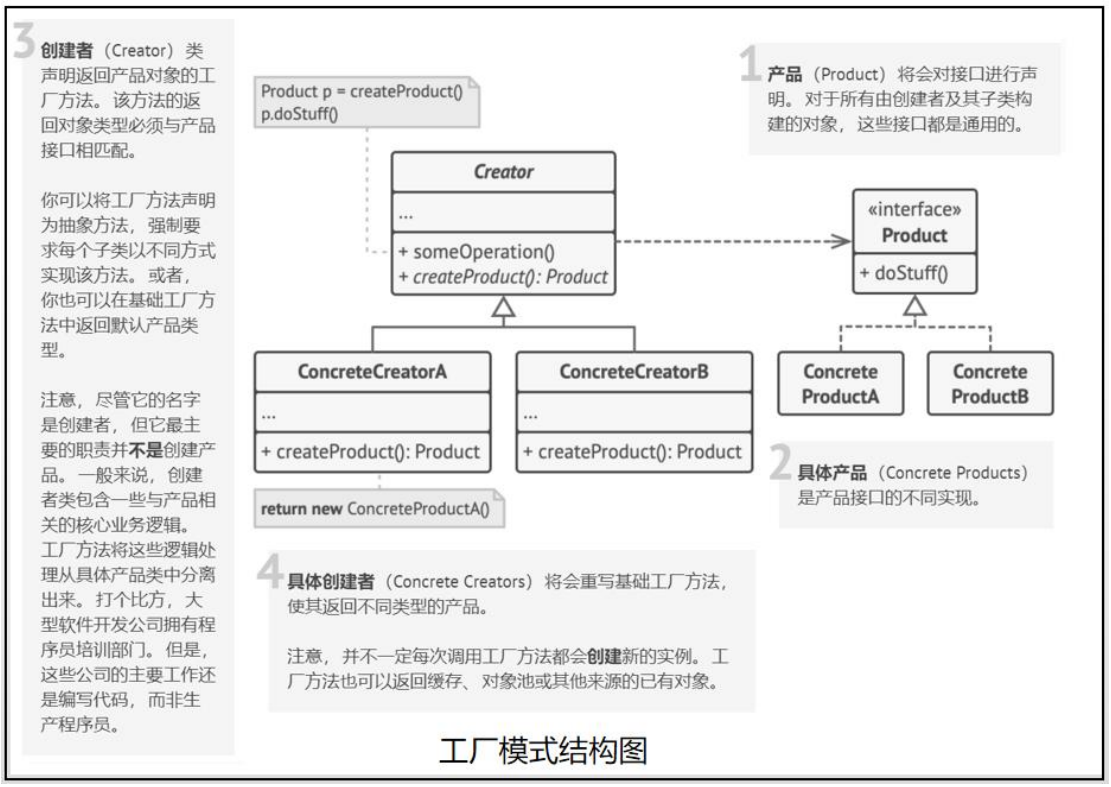
在飞机大战游戏中有 3 种类型的敌机：普通敌机、精英敌机、Boss 敌机。普通敌机和精英敌机以一定频率在界面随机位置出现并向屏幕下方移动。Boss 敌机悬浮于界面上方，直至被消灭。敌机通过生命值（血）生存，被英雄机子弹击中损失部分生命值，生命值为 0 时坠毁。

游戏中还有 3 种类型的道具：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。道具以一定速度向屏幕下方移动，与英雄机碰撞或移动至界面底部则消失。

请结合以上两个实例场景，为创建普通和精英两种敌机，以及三种道具绘制工厂模式的 UML 结构图，要求给出设计模式的名称，类名、方法名和属性名可自行定义。

工厂模式

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。



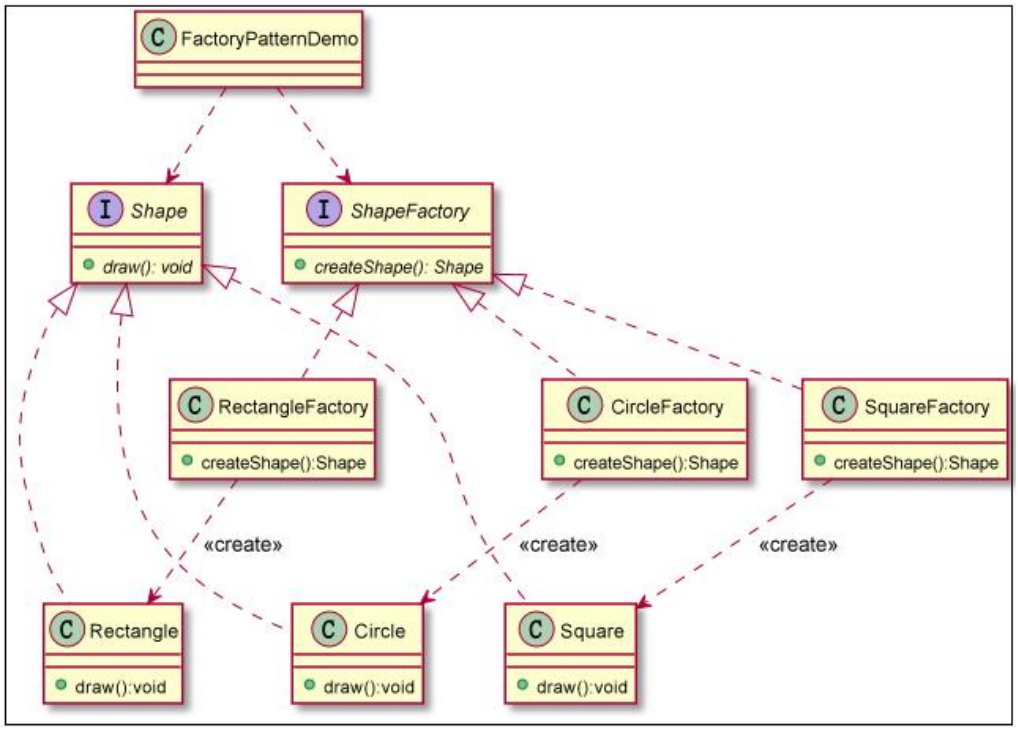
请参考以上 UML 结构图，绘制飞机大战中的工厂模式。

4.4 根据设计的类图，重构代码，实现工厂模式

根据 4.3 中你所设计的 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。

工厂模式代码示例：

我们将创建一个 Shape 接口和实现 Shape 接口的实体类。下一步是定义工厂类 ShapeFactory。客户端 FactoryPatternDemo 类使用 ShapeFactory 来获取不同的 Shape 对象。



步骤 1： 创建一个接口，充当产品角色。

Shape.java

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

步骤 2：创建实现接口的实体类，充当具体产品角色。

Rectangle.java

```
public class Rectangle implements Shape {

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

Square.java

```
public class Square implements Shape {

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

Circle.java

```
public class Circle implements Shape {

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

步骤 3：创建一个工厂接口，充当创建者角色。

ShapeFactory.java

```
public interface ShapeFactory {

    public abstract Shape createShape();

}
```

步骤 4：创建实现工厂接口的具体工厂类，充当具体创建者角色。

RectangleFactory.java

```
public class RectangleFactory implements ShapeFactory {

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

SquareFactory.java

```
public class SquareFactory implements ShapeFactory {

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

CircleFactory.java

```
public class CircleFactory implements ShapeFactory {

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

步骤 5：使用 FactoryPatternDemo 来演示工厂模式的用法。

FactoryPatternDemo.java

```
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();

    }
}
```

步骤 6：执行程序，输出结果：

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

5. 作业提交

实验课前，预习并理解单例模式和工厂模式的基本要素，包括模式名称、问题描述、解决方案和应用效果。

✧ **提交内容**

包括：

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。