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

实验六 设计模式实验（3）

—— 观察者模式和模板模式

2022 春

目录

1. 实验目的

1. 理解观察者模式和模板模式的模式动机和意图，掌握模式结构
2. 结合实例，熟练绘制观察者和模板两种模式的 UML 结构图
3. 重构代码，学习如何使用代码实现观察者和模板两种模式

2. 实验环境

1. Windows 10
2. IntelliJ IDEA 2021.3.2
3. Java 11

3. 实验内容（4 学时）

(1) 结合实例，绘制观察者模式的 UML 结构图（类图）。
(2) 根据设计的 UML 类图，重构代码，采用观察者模式实现炸弹道具。
(3) 结合实例，绘制模板模式的 UML 结构图（类图）。
(4) 根据设计的 UML 类图，重构代码，采用模板模式实现简单、普通和困难三种游戏难度。

4. 实验步骤

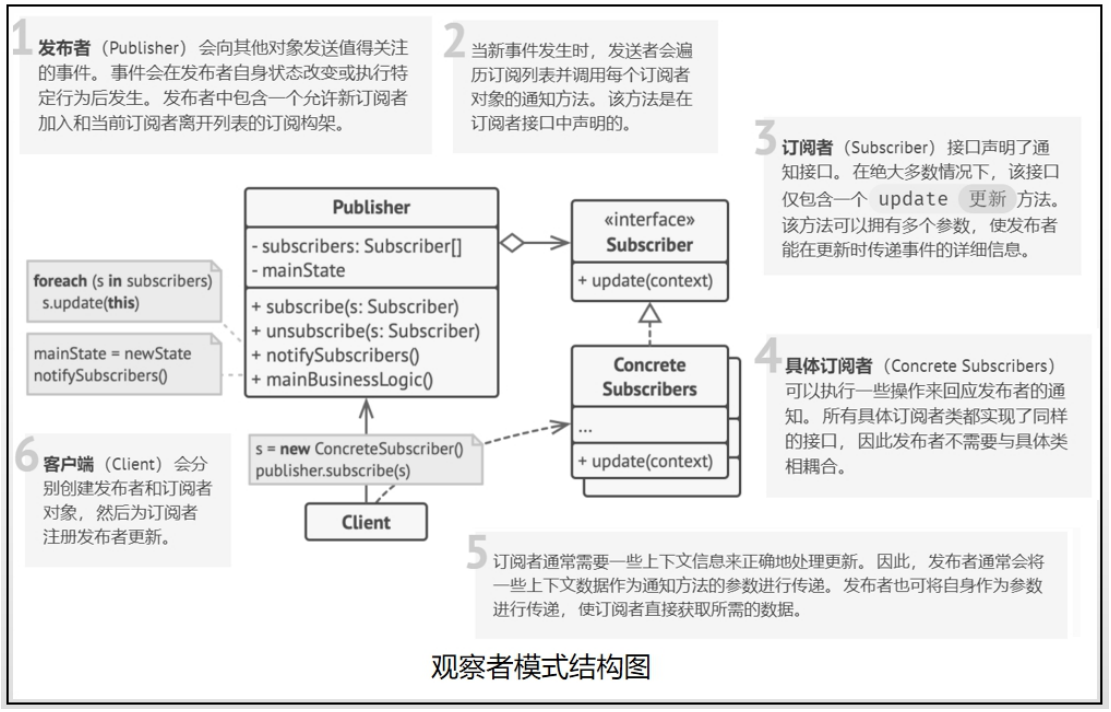
4.1 结合实例，绘制观察者模式的 UML 结构图

在飞机大战游戏中，精英敌机和 Boss 机坠毁时会以较低概率掉落炸弹道具。英雄机碰撞炸弹道具后，道具生效，可清除界面上除 boss 机外的所有敌机和敌机子弹。

请结合以上该实例场景，为实现炸弹道具绘制观察者模式的 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。

观察者模式

观察者模式（Observer Pattern）也是一种行为设计模式， 允许你定义一种订阅机制， 可在对象事件发生时通知多个 “观察” 该对象的其他对象。



观察者模式结构图

请参考以上 UML 结构图，绘制飞机大战中的观察者模式。

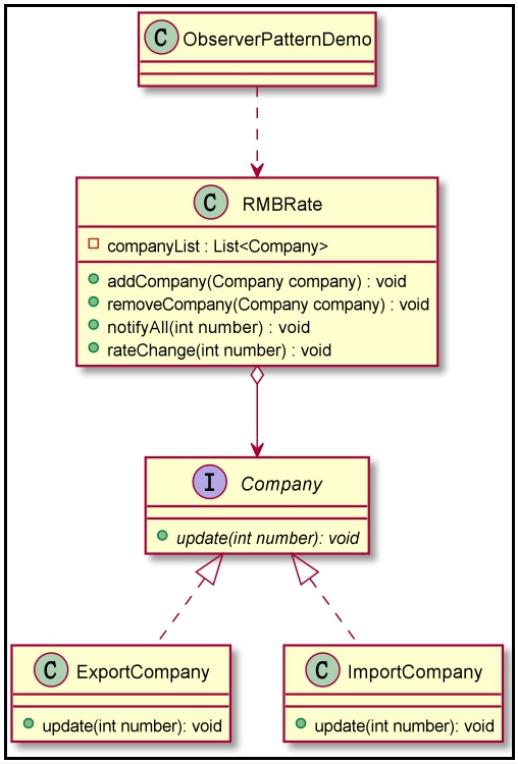
4.2 根据设计的类图，重构代码，实现观察者模式

根据 4.1 中你所设计的 UML 类图，重构代码，采用观察者模式实现炸弹道具。

观察者模式的代码实现方式示例（人民币汇率波动对进出口公司的影响）：

当“人民币汇率”升值时，进口公司的进口产品成本降低且利润率提升，出口公司的出口产品收入降低且利润率降低；当“人民币汇率”贬值时，进口公司的进口产品成本提升且利润率降低，出口公司的出口产品收入提升且利润率提升。

我们将创建 RMBRate 对象作为发布者，它带有绑定观察者的方法。创建 Company 接口和实现了该接口的实体类，充当订阅者角色。ObserverPatternDemo 使用 RMBRate 对象和 Company 实体类对象来演示观察者模式。



步骤 1：创建 RMBRate 类，充当发布者角色。

RMBRate.java

```
public class RMBRate {

    //观察者列表
    private List<Company> companyList = new ArrayList<>();

    //增加观察者
    public void addCompany(Company company) {
        companyList.add(company);
    }

    //删除观察者
    public void removeCompany(Company company) {
        companyList.remove(company);
    }

    //通知所有观察者
    public void notifyAll(int number) {
        for (Company company : companyList) {
            company.update(number);
        }
    }

    //人民币汇率改变
    public void rateChange (int number) {
        notifyAll(number);
    }
```

```
}
```

步骤 2: 创建 Company 接口，充当订阅者角色。

Company.java

```
public interface Company {
        /**
         * 对汇率的反应
         * @param number 汇率
         */
        void update(int number);
    }
```

步骤 3: 创建 Company 接口实体类，充当具体订阅者角色。

ImportCompany.java

```
public class ImportCompany implements Company {
    @Override
    public void update(int number) {
        System.out.print("进口公司收到消息：");
        if (number > 0) {
            System.out.println("人民币汇率升值" + number + "个基点，
                                进口产品成本降低，公司利润提升。");
        } else if (number < 0) {
            System.out.println("人民币汇率贬值" + (-number) + "个基点，
                                进口产品成本提高，公司利润降低。");
        }
    }
}
```

ExportCompany.java

```
public class ExportCompany implements Company {
    @Override
    public void update(int number) {
        System.out.print("出口公司收到消息：");
        if (number > 0) {
            System.out.println("人民币汇率升值" + number + "个基点，
                                出口产品收入降低，公司销售利润降低。");
        } else if (number < 0) {
            System.out.println("人民币汇率贬值" + (-number) + "个基点，
                                出口产品收入提高，公司销售利润降低提高。");
        }
    }
}
```

步骤 4: 使用 RMBRate 对象和 Company 实体类对象来演示观察者模式。

ObserverPatternDemo.java

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

```
        RMBRate rate = new RMBRate();
        Company company1 = new ImportCompany();
        Company company2 = new ExportCompany();

        rate.addCompany(company1);
        rate.addCompany(company2);

        System.out.println("人民币汇率改变：");
        rate.rateChange(10);

        System.out.println("人民币汇率改变：");
        rate.rateChange(-5);

        rate.removeCompany(company1);

        System.out.println("人民币汇率改变：");
        rate.rateChange(8);

    }
}
```

步骤 5: 执行程序，输出结果：

```
人民币汇率改变：
进口公司收到消息：人民币汇率升值 10 个基点，进口产品成本降低，公司利润提升。
出口公司收到消息：人民币汇率升值 10 个基点，出口产品收入降低，公司销售利润降低。

人民币汇率改变：
进口公司收到消息：人民币汇率贬值 5 个基点，进口产品成本提高，公司利润降低。
出口公司收到消息：人民币汇率贬值 5 个基点，出口产品收入提高，公司销售利润降低提高。

人民币汇率改变：
出口公司收到消息：人民币汇率升值 8 个基点，出口产品收入降低，公司销售利润降低。
```

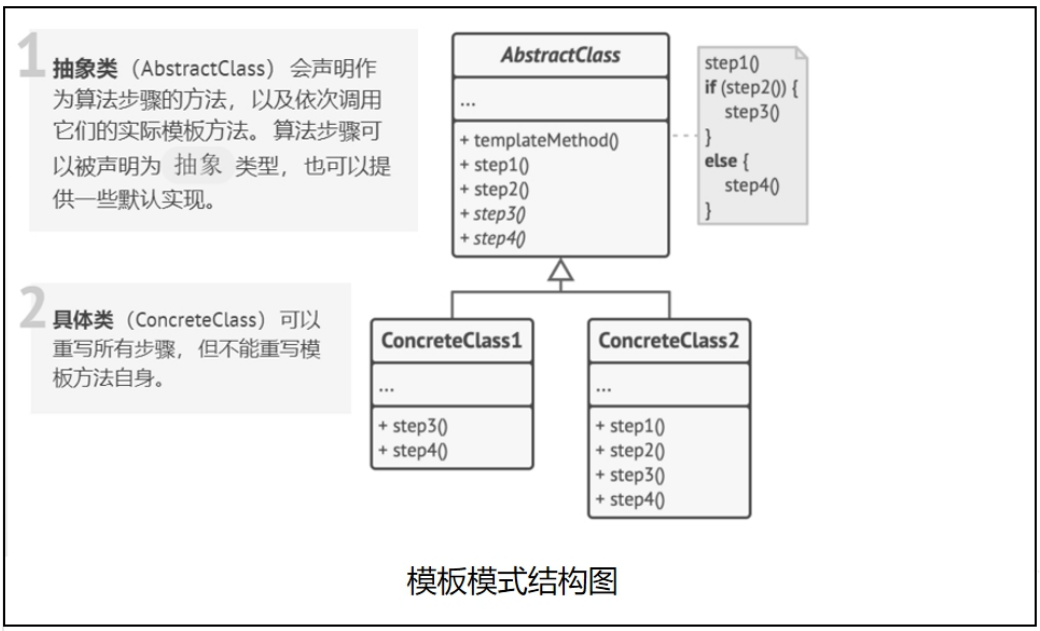
4.3 结合实例，绘制模板模式的 UML 结构图

用户进入飞机大战游戏界面后，可选择某种游戏难度：简单 / 普通 / 困难。当用户选择后，出现该难度对应的地图，且英雄机和敌机的战斗力会相应调整。普通和困难模式随着游戏时长增加而提升难度，如提升普通和精英敌机的速度和血量、缩短敌机产生周期等，且当得分每超过一次 bossScoreThreshold，则产生一次 boss 机。

请结合该实例场景，为创建不用的游戏难度绘制模板模式的 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。

模板模式

模板模式（Template Pattern）是一种行为设计模式， 它在超类中定义了一个算法的框架， 允许子类在不修改结构的情况下重写算法的特定步骤。



模板模式结构图

请参考以上 UML 结构图，绘制飞机大战中的模板模式。

4.4 根据设计的类图，重构代码，实现模板模式

根据 4.3 中你所设计的 UML 类图，重构代码，采用模板模式实现简单、普通和困难三种游戏难度。

	简单	普通	困难
Boss 敌机	无	有 每次召唤不改变 Boss 机血量	有 每次召唤提升 Boss 机血量
难度是否随时间增加	否	是	是

游戏难度设置可考虑如下因素（至少选择 5 个完成）：

- ✧ 游戏界面中出现的敌机数量的最大值
- ✧ 敌机的属性值，如血量、速度
- ✧ 英雄机的射击周期
- ✧ 敌机的射击周期
- ✧ 精英敌机的产生概率
- ✧ 普通和精英敌机的产生周期
- ✧ Boss 敌机产生的分数阈值
- ✧ Boss 敌机的血量
- ✧ ...

模板模式的代码实现方式示例（银行业务办理流程）：

在银行办理业务时，一般都包含几个基本步骤，首先需要取号排队，然后办理具体业务，最后需要对银行工作人员进行评分。无论具体业务事取款、存款还是转账，其基本流程都一样。我们将创建一个定义操作的 BankTemplateMethod 抽象类，其中，模板方法设置为 final，这样它就不会被重写。Deposit 、Transfer 和 Withdraw 是扩展了 BankTemplateMethod 的实体类，它们重写了抽象类的方法。TemplatePatternDemo 是我们的演示类，用于演示模板模式的用法。

步骤 1: 创建一个抽象类，它的模板方法被设置为 final。

BankTemplateMethod.java

```
public abstract class BankTemplateMethod {

    public final void takeNumber()
    {
        System.out.println("取号排队");
    }

    public abstract void transact();

    public void evaluate()
    {
        System.out.println("反馈评分");
    }

    //模板方法
    public final void process()
    {
        this.takeNumber();
        this.transact();
        this.evaluate();
    }
}
```

步骤 2: 创建扩展了上述类的实体类，它们重写了抽象类的某些方法。

Deposit.java

```
public class Deposit extends BankTemplateMethod {

    @Override
    public void transact() {
        System.out.println("存款");
    }
}
```

Transfer.java

```
public class Transfer extends BankTemplateMethod {
```

```
    @Override
    public void transact() {
        System.out.println("转账");
    }
}
```

Withdraw.java

```
public class Withdraw extends BankTemplateMethod {

    @Override
    public void transact() {
        System.out.println("取款");
    }
}
```

步骤 3: 使用 BankTemplateMethod 的模板方法 process() 来演示模板模式。

TemplatePatternDemo.java

```
public class TemplatePatternDemo {

    public static void main(String[] args) {

        BankTemplateMethod bank;
        System.out.println("顾客 1： ");
        bank = new Deposit();
        bank.process();

        System.out.println("顾客 2： ");
        bank = new Withdraw();
        bank.process();

        System.out.println("顾客 3： ");
        bank = new Transfer();
        bank.process();

    }
}
```

步骤 4: 执行程序，输出结果：

```
顾客 1：
取号排队
存款
反馈评分

顾客 2：
取号排队
取款
反馈评分

顾客 3：
取号排队
```

转账
反馈评分

5. 实验要求

1. 实验课前，预习并理解模板方法模式和观察者模式的基本要素，包括模式名称、问题描述、解决方案和应用效果。
2. 结合飞机大战实例，提交绘制的模板模式和观察者模式的 UML 结构图（整合在实验报告中）。
3. 提交重构后完整的项目代码，代码运行正确。
4. 录制一段视频（小于 2min），展示你的飞机大战游戏的所有功能点。
5. 根据模板，完成整个项目的实验报告。

本次实验提交版本需完成以下功能：

✓ 设计并实现三种游戏难度，其中普通和困难模式随着游戏时长增加而提升难度（控制台输出），且当得分每超过一次阈值，则产生一次 Boss 敌机。

✓ 炸弹道具生效时清除界面上除 Boss 机外的所有敌机和敌机子弹。