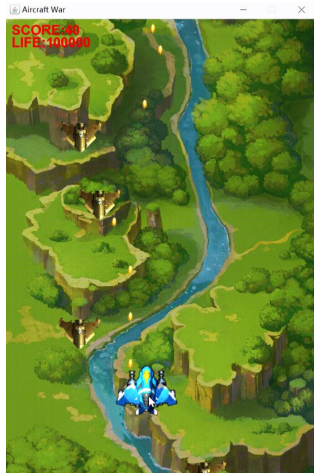
实验六：设计模式实验（3）
观察者模式和模板模式

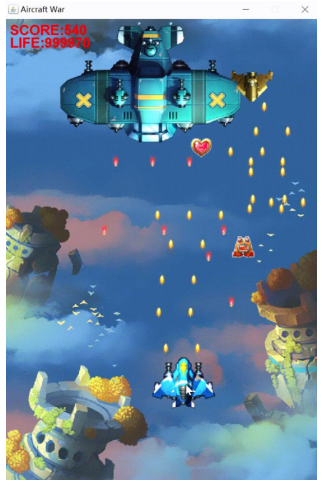
实验与创新实践教育中心 • 计算机与数据技术实验教学部

本学期实验总体安排

初始版本



最终版本



初始版本

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

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

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

03

添加**JUnit单元测试**
创建Boss和超级精英敌机

04

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

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

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

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**

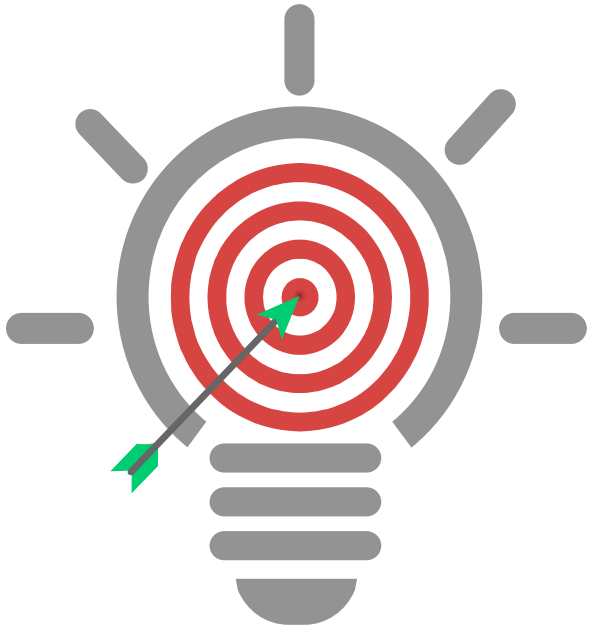
实验课程共16个学时，6个实验项目，总成绩为40分。

目录

CONTENTS

实验目的

难度	知识点
理解	观察者模式和模板模式的模式动机和意图
掌握	观察者模式和模板模式UML结构图的绘制方法
熟练	使用Java语言，编码实现观察者模式和模板模式

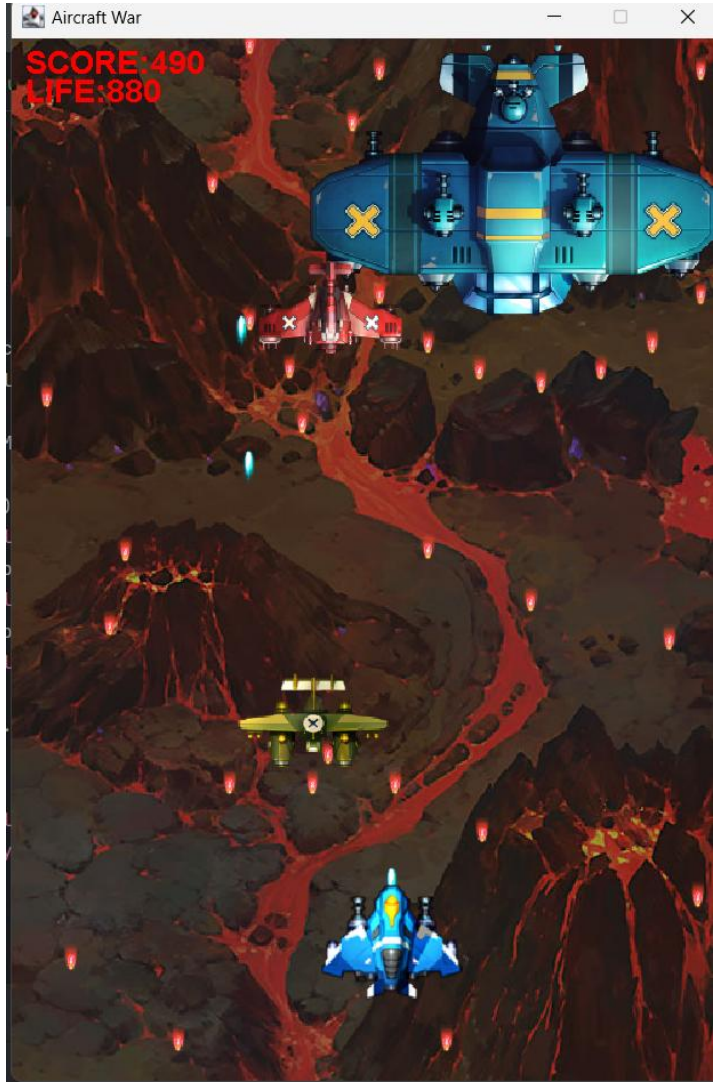


实验任务

绘制类图、重构代码，完成以下功能：

1. 采用观察者模式实现炸弹道具；

2. 采用模板模式实现简单、普通、困难三种游戏难度 。

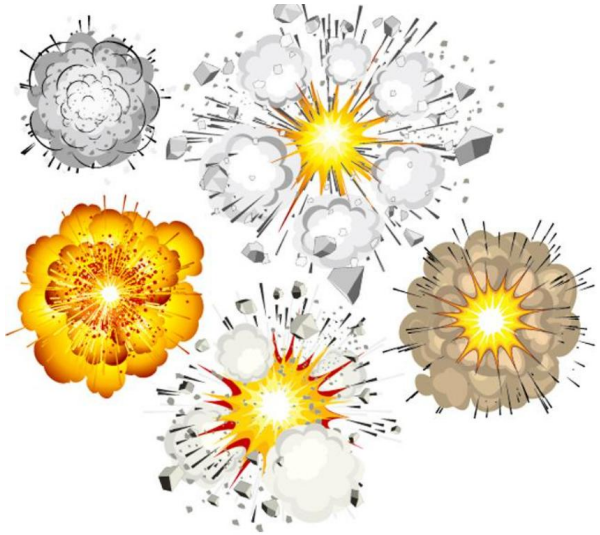


注意：先“设计”再编码！请结合飞机大战实例，完成模式UML类图设计后，再进行编码。

实验原理：场景分析（1）

炸弹场景分析

敌机坠毁时会以较低概率掉落炸弹道具。它可清除界面上的所有的普通、精英敌机和敌机子弹，超级精英敌机血量减少，Boss敌机不受影响。英雄机可获得坠毁的敌机分数。

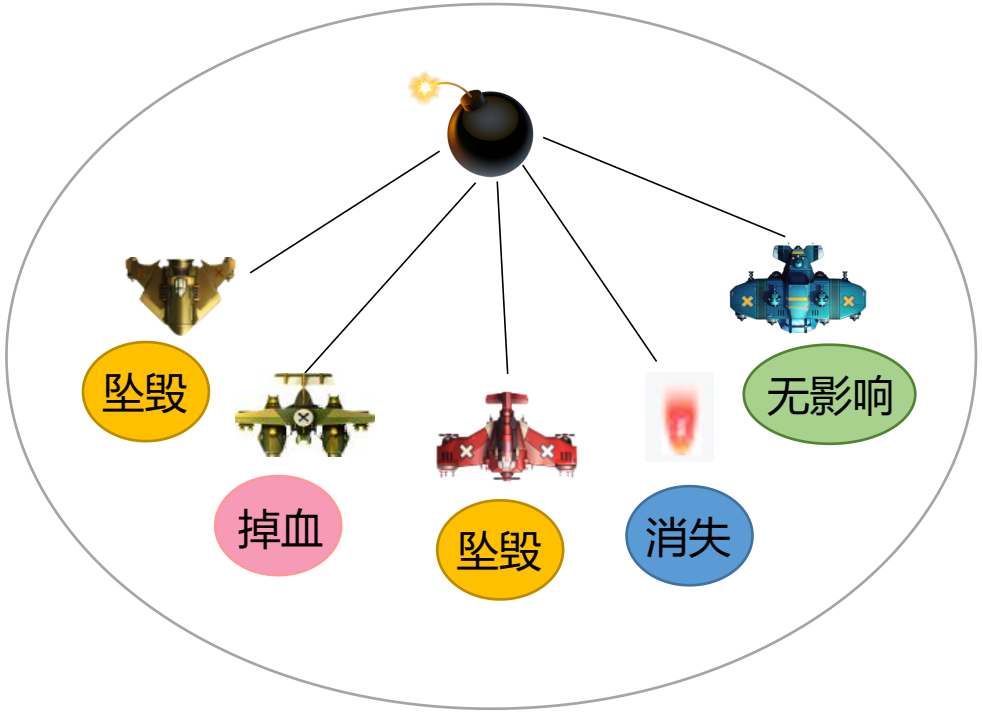


实验原理：场景分析（1）

请思考：

1. 目前在哪个类实现炸弹爆炸？有哪些角色对炸弹有响应？如何响应？

```java
public class BombSupply extends AbstractFlyingSupply{
    public BombSupply(int locationX, int locationY, int speedX, int speedY) {
        super(locationX, locationY, speedX, speedY);
    }

    @Override
    public void activate() {
        System.out.println("BombSupply active!");
    }
}
```



实验原理：场景分析（2）

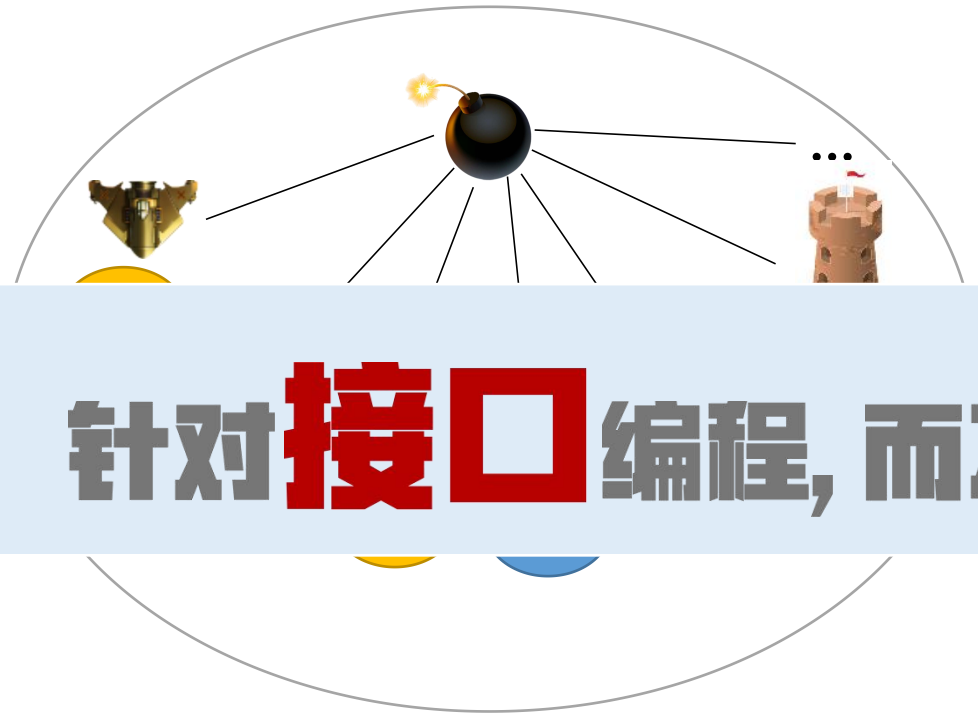
请思考：

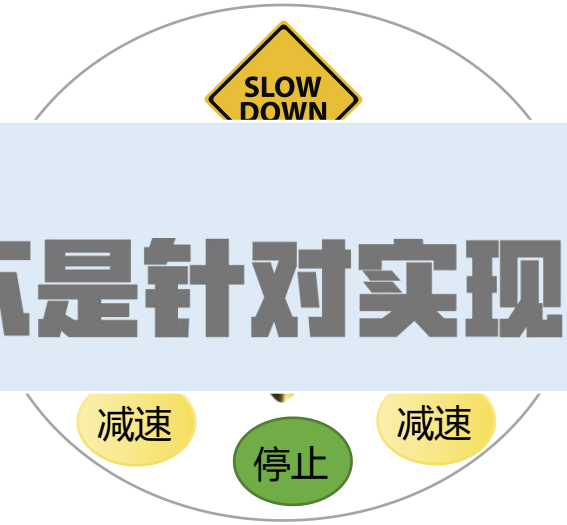
2. 如何增加一个对炸弹有响应的新角色，如敌方炮台？

3. 如何增加一个新型道具，如减速道具？

违反
单一职责 X

违反
开闭原则 X



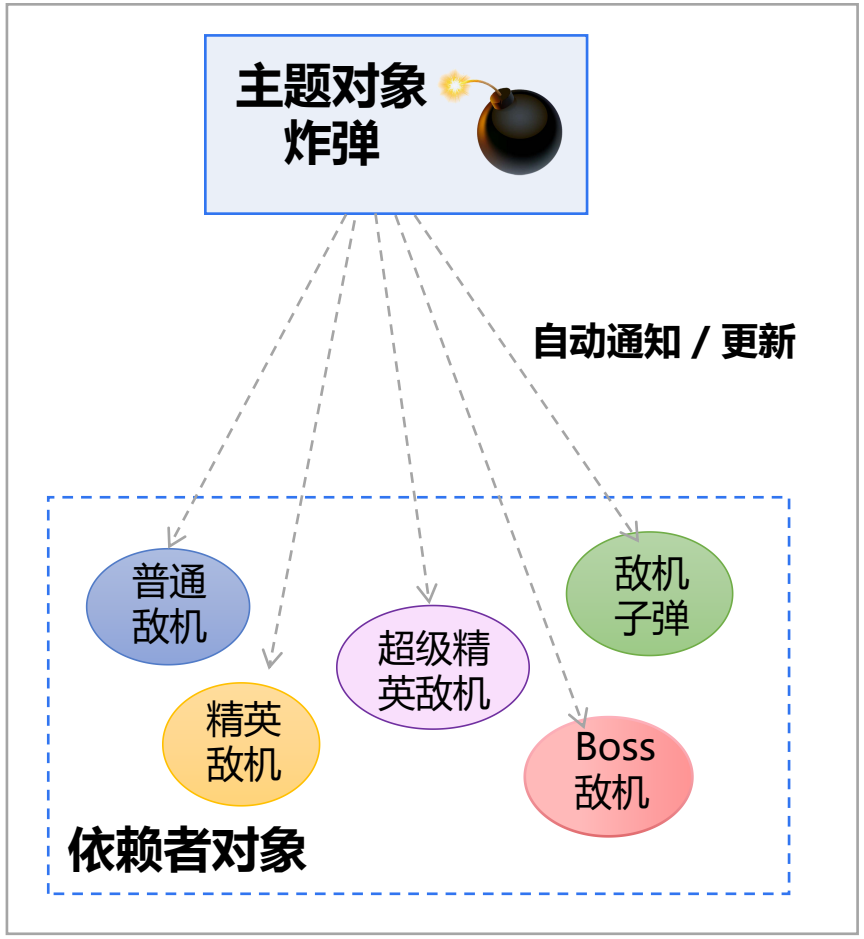


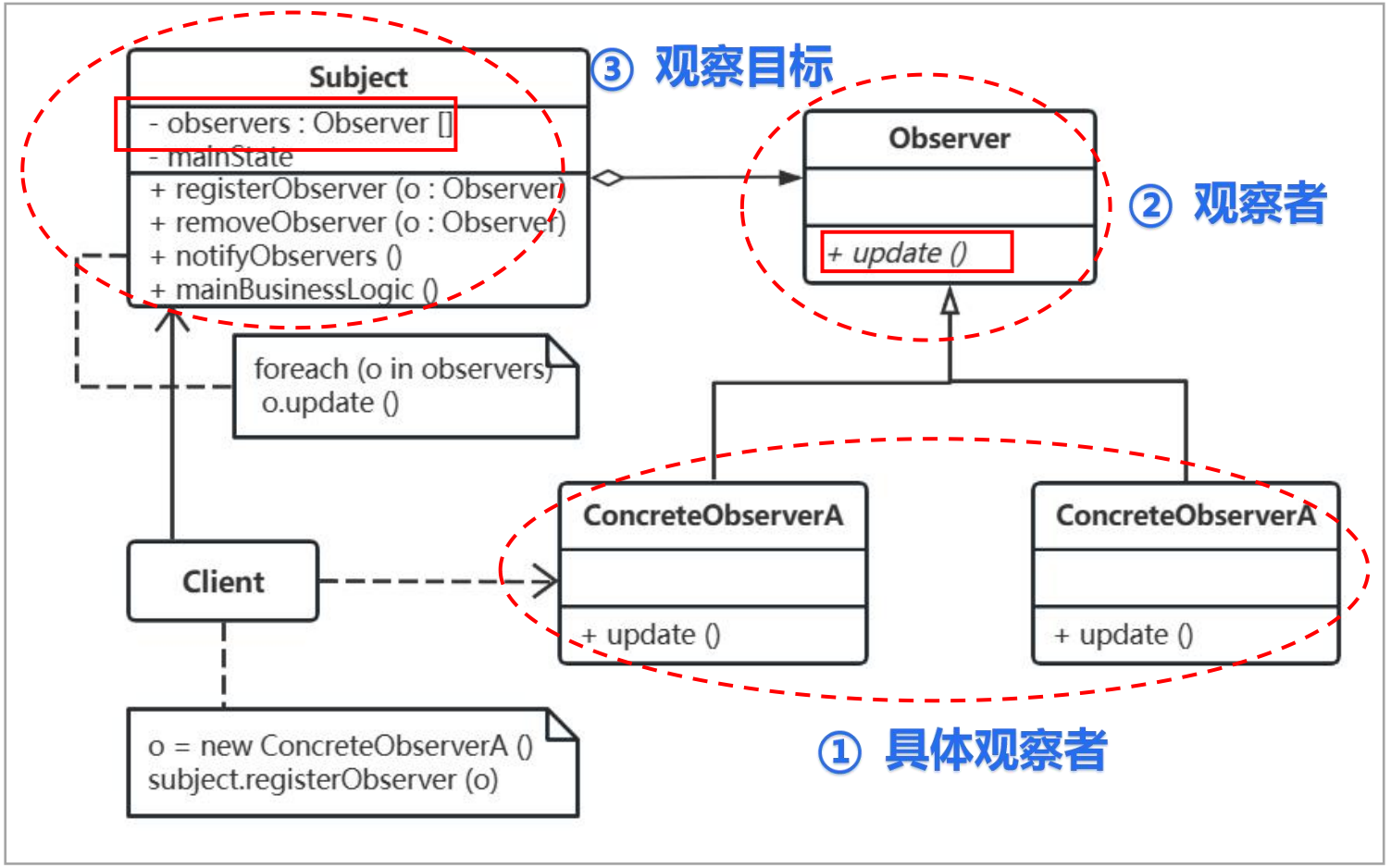
X

针对接口编程，而不是针对实现编程！

实验原理：观察者模式结构图

观察者模式（Observer Pattern）定义了对象之间的一对多依赖。当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。





实验步骤：观察者模式

01 分析

结合飞机大战游戏场景，分析用户需求及当前软件开发中的痛点。

02 设计

借鉴**观察者模式**的解题思路，设计解决方案并绘制UML类图。

03重构

编写代码，用观察者模式实现**炸弹道具**，解决开发痛点。

04 验证

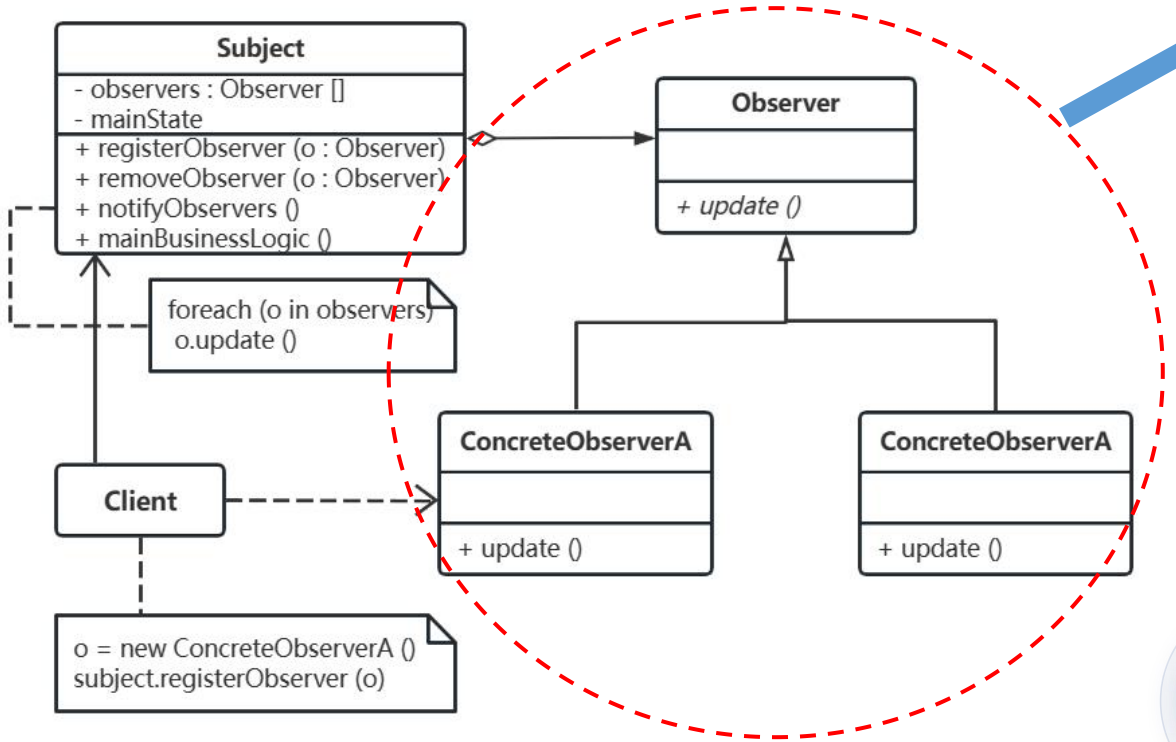
创建更多新型敌方角色，验证观察者模式的优越性。

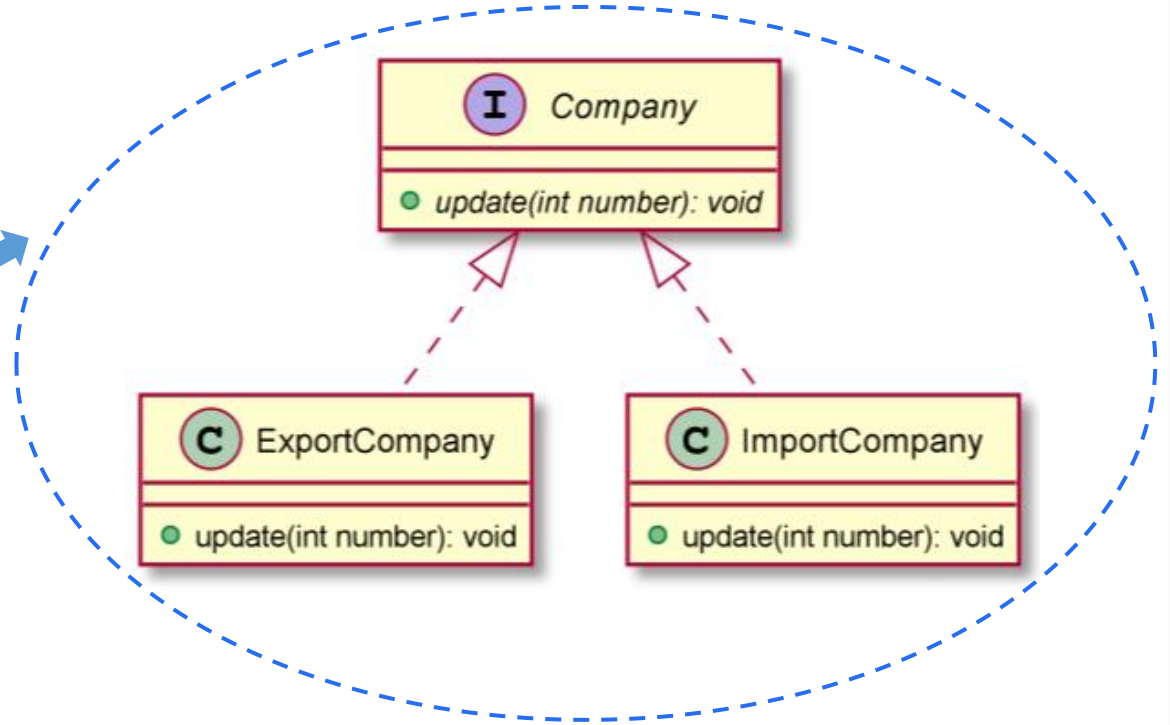
实验步骤：人民币汇率举例（观察者模式）

假如我们要实现一个功能：观察人民币汇率波动对进口和出口公司的影响。未来可能会支持更多类型的公司。我们该如何用观察者模式实现呢?

实验步骤：人民币汇率举例（1）

UML 类 图 设 计

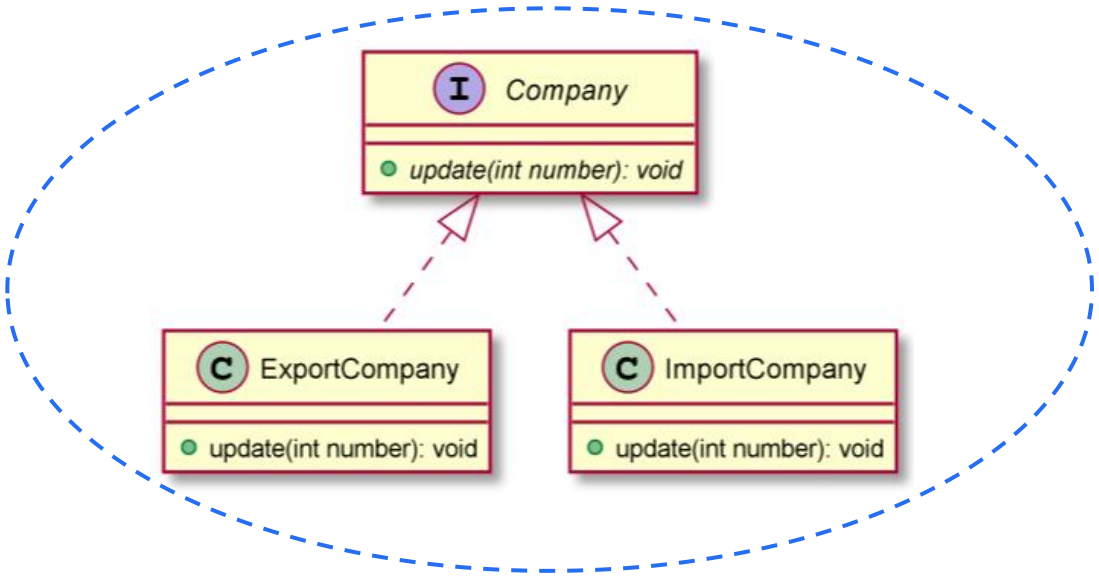




观察者 创建观察者接口和实现该接口的两个具体观察者类。

实验步骤：人民币汇率举例（2）

代 码 实 现



1 创建观察者接口，充当观察者角色；

```
public interface Company {

    // 对人民币汇率的响应
    public abstract void update(int number);
}
```

2 创建两个观察者实体类，充当具体观察者角色；

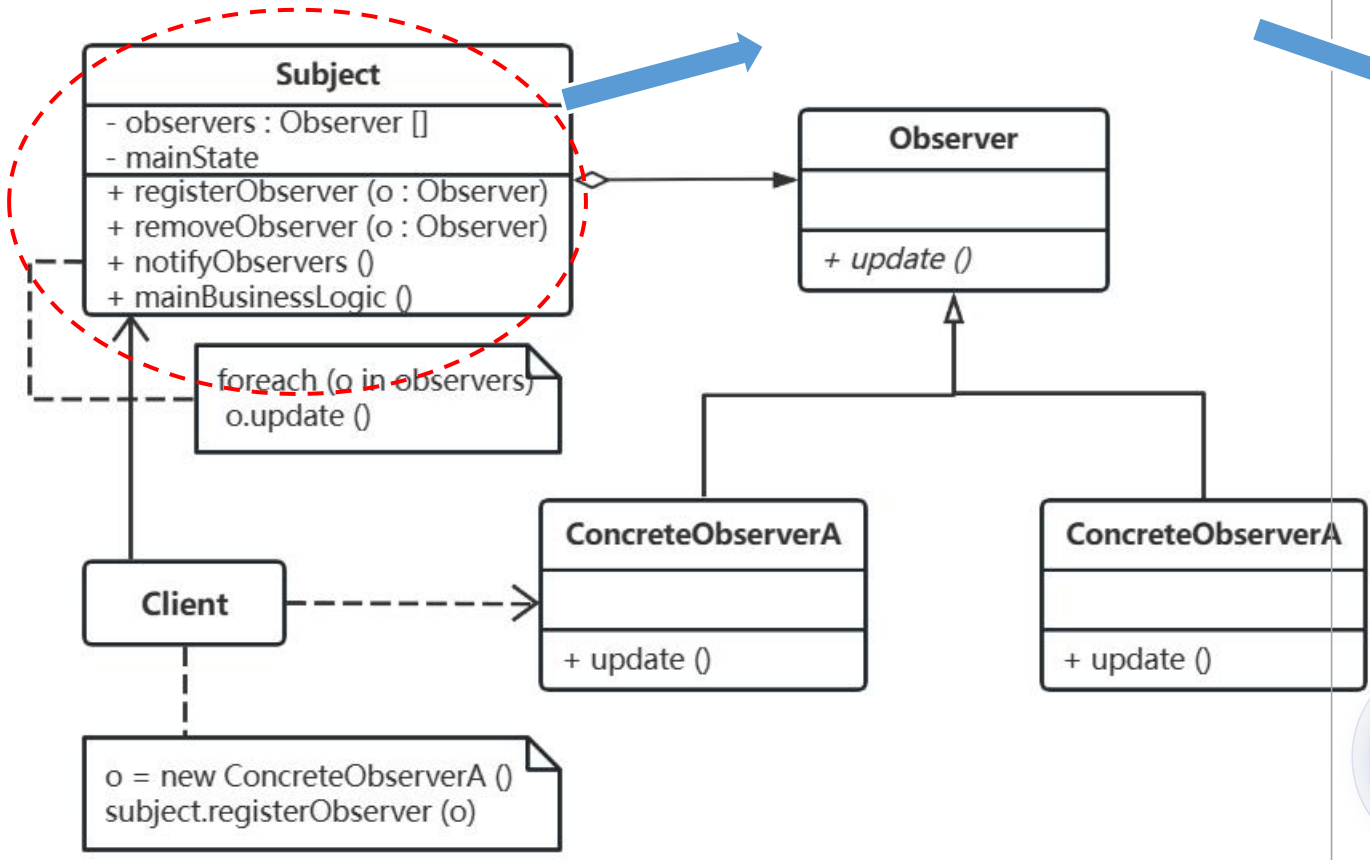
```
public class ImportCompany implements Company {

    @Override
    public void update(int number)
    {
    }
}
```

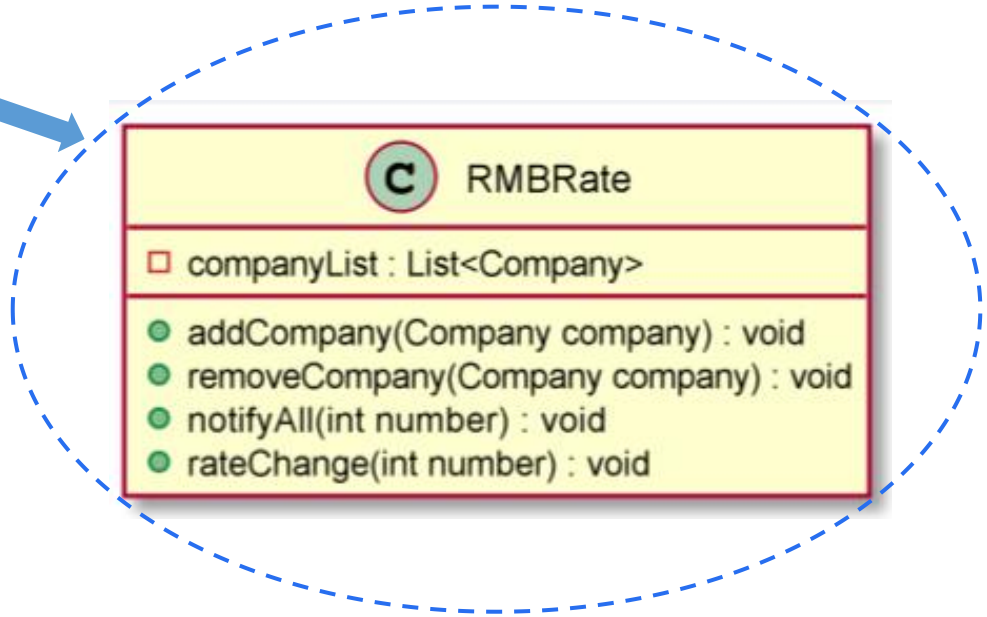
```
public class ExportCompany implements Company {

    @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，公司销售利润提升。");
        }
    }
}
```

出口公司对人民币汇率的响应

实验步骤：人民币汇率举例（3）

UML 类 图 设 计



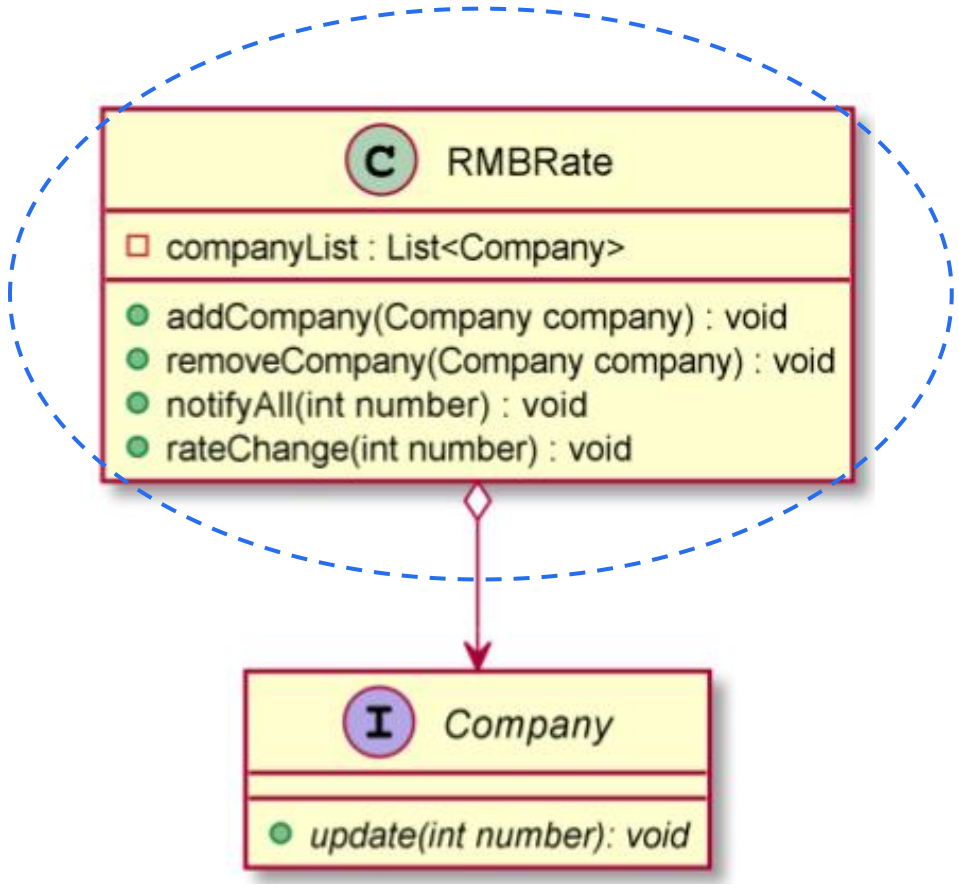


观察目标

创建观察目标RMBRate，当其状态改变时，通知所有观察者们。

实验步骤：人民币汇率举例（4）

代 码 实 现

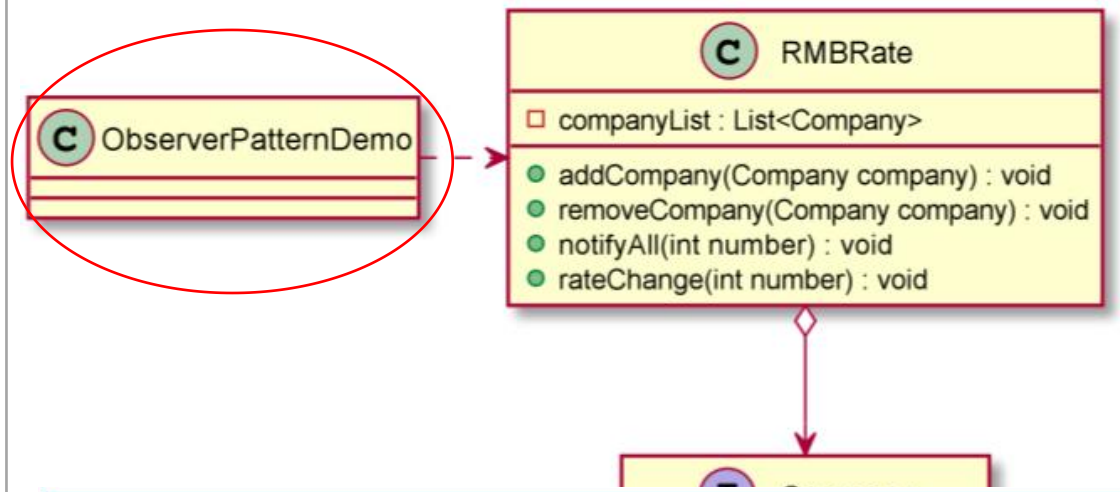


3 创建RMBRate类，充当观察目标角色；

```
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);
    }

}
```

实验步骤：人民币汇率举例（5）

UML 类 图 设 计



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，公司销售利润提升。

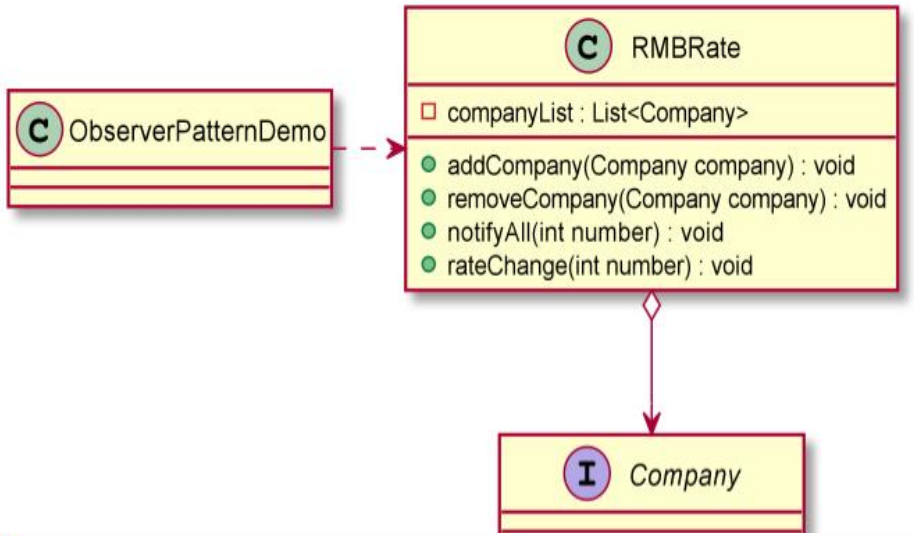
人民币汇率改变：
出口公司收到消息：人民币汇率升值8个基点，出口产品收入降低，公司销售利润降低。

4 客户端注册观察者，并触发观察目标的状态改变。

```
public class ObserverPatternDemo {

    public static void main(String[] args) {
        //初始化
        RMBRate rate = new RMBRate();
        Company company1 = new ImportCompany();
        Company company2 = new ExportCompany();

        //增加观察者
        rate.addCompany(company1);
        rate.addCompany(company2);
        //观察目标状态改变
        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);
        //删除观察者
        rate.removeCompany(company1);
        //观察目标状态改变
        System.out.println("人民币汇率改变：");
        rate.rateChange(8);
```

请思考：如何添加一个进出口公司?

UML 类 图 设 计




```
public class ObserverPatternDemo {

    public static void main(String[] args) {
        //初始化
        RMBRate rate = new RMBRate();
        Company company1 = new ImportCompany();
        Company company2 = new ExportCompany();

        Company company3 = new ImportExportCompany();

        //增加观察者
        rate.addCompany(company1);
        rate.addCompany(company2);

        rate.addCompany(company3);

状态改变
println("人民币汇率
ange(10);
println("人民币汇率
ange(-5);
```

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。
进出口公司收到消息：人民币汇率升值10个基点，进口产品成本降低，出口产品收入降低，公司销售利润持平。

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，公司销售利润提升。
进出口公司收到消息：人民币汇率贬值5个基点，进口产品成本提高，出口产品收入提高，公司销售利润持平。

开闭原则

实验原理：场景分析（2）

难度选择
场景分析

用户进入游戏界面后，可选择某种游戏难度：简单 / 普通 / 困难。用户选择后，出现该难度对应的地图，且游戏难度会相应调整。

实验原理：场景分析（2）

难度选择
场景分析

★ 基本要求	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（至少设置5个）：

- 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敌机产生的得分阈值
- Boss敌机每次出现的血量
- …

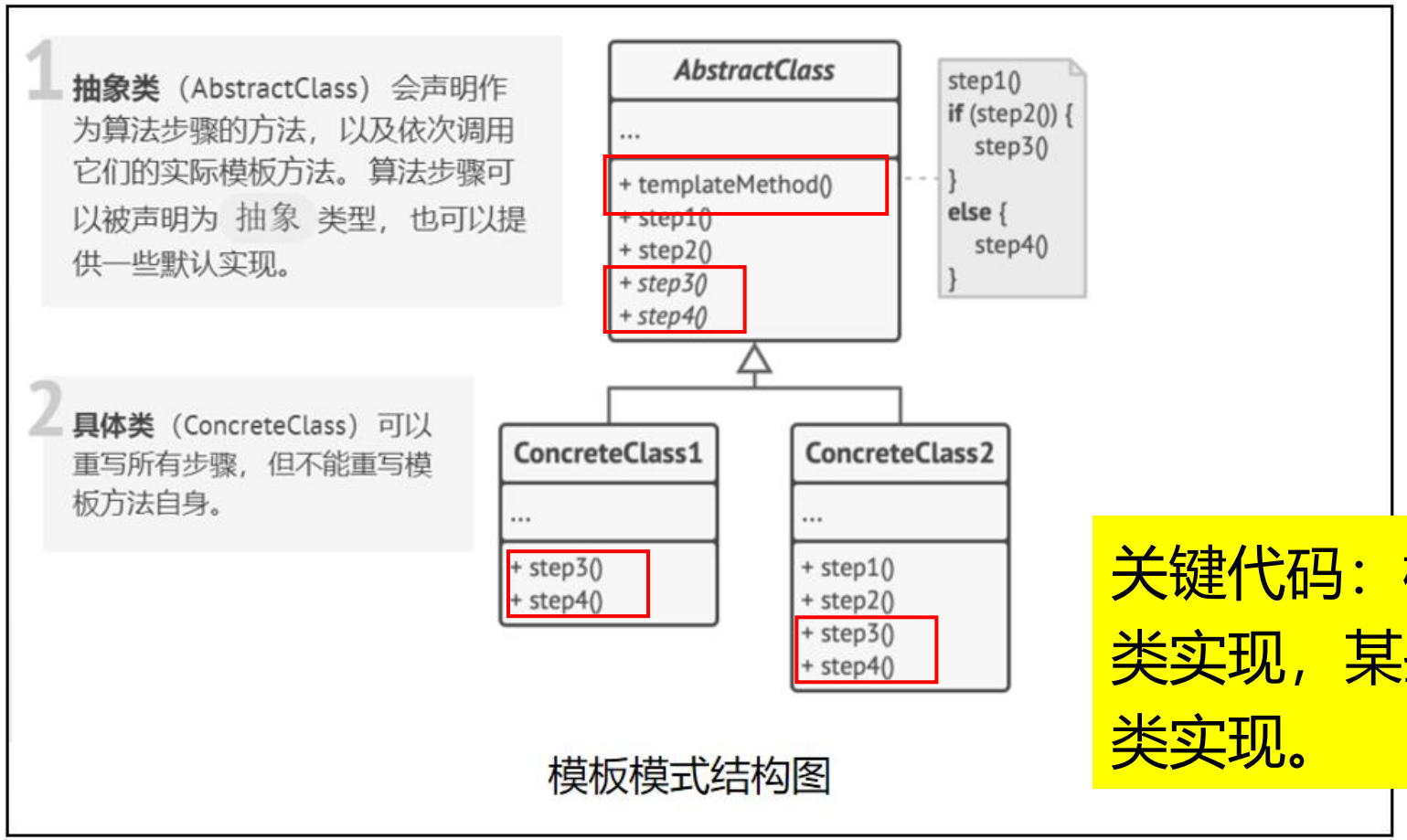
实验原理：场景分析（2）

请思考：

1. 三种游戏难度有哪些共性的地方？哪些不同的地方？
2. 若要增加一种新的游戏难度，需要改动哪些地方？
3. 如何实现代码复用？

实验原理：模板模式结构图

模板模式（Template Pattern）是一种行为型设计模式， 它在抽象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

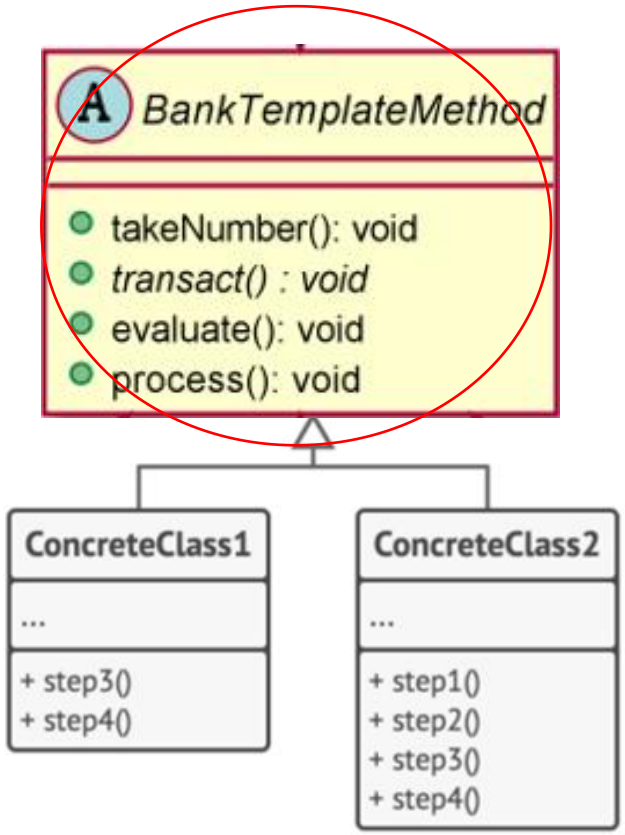
关键代码：模板方法在抽象类实现，某些特定步骤在子类实现。

实验步骤：银行业务办理举例（模板模式）

假如我们要去银行办理业务，要经过取号排队、办理业务、对银行工作人员进行评分三个步骤。我们该如何绘制UML类图？

实验步骤：银行业务办理举例（1）

UML 类 图 设 计



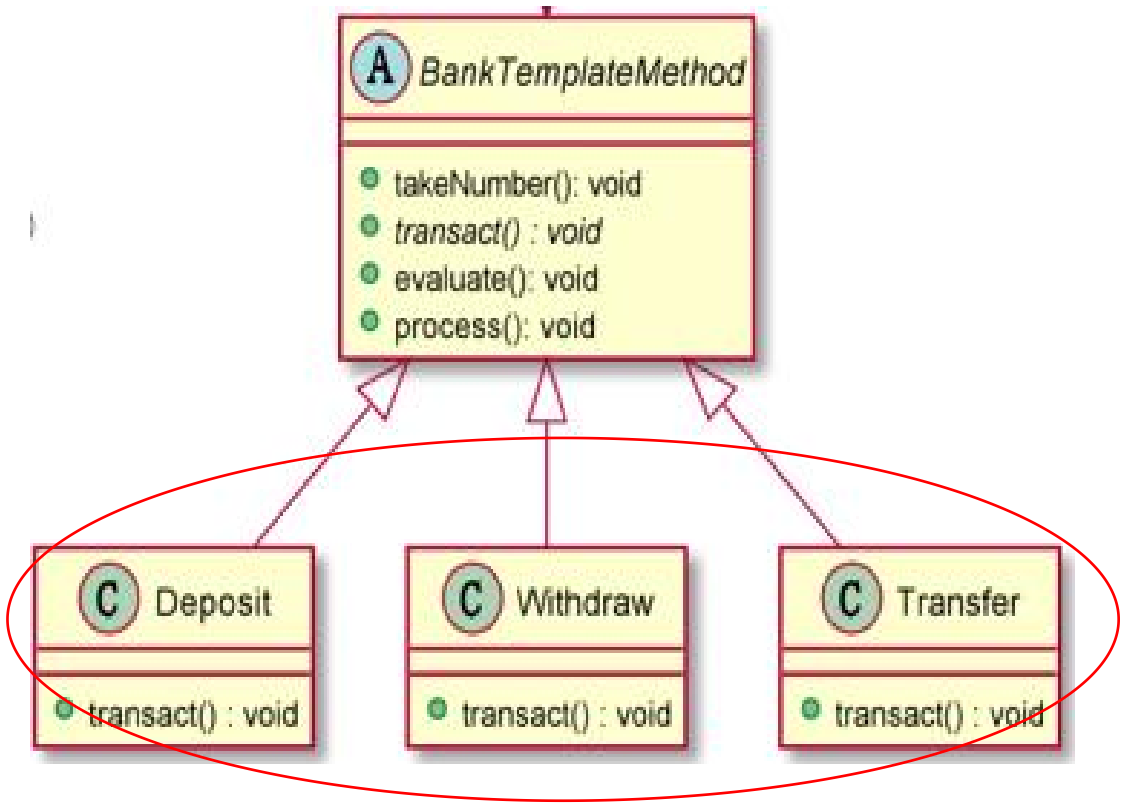
① 创建一个定义操作的BankTemplateMethod 抽象类及模板方法process()。

```
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");
    }

    public final void process()
    {
        this.takeNumber();
        this.transact();
        this.evaluate();
    }
}
```

模板方法

实验步骤：银行业务办理举例（2）

UML 类 图 设 计

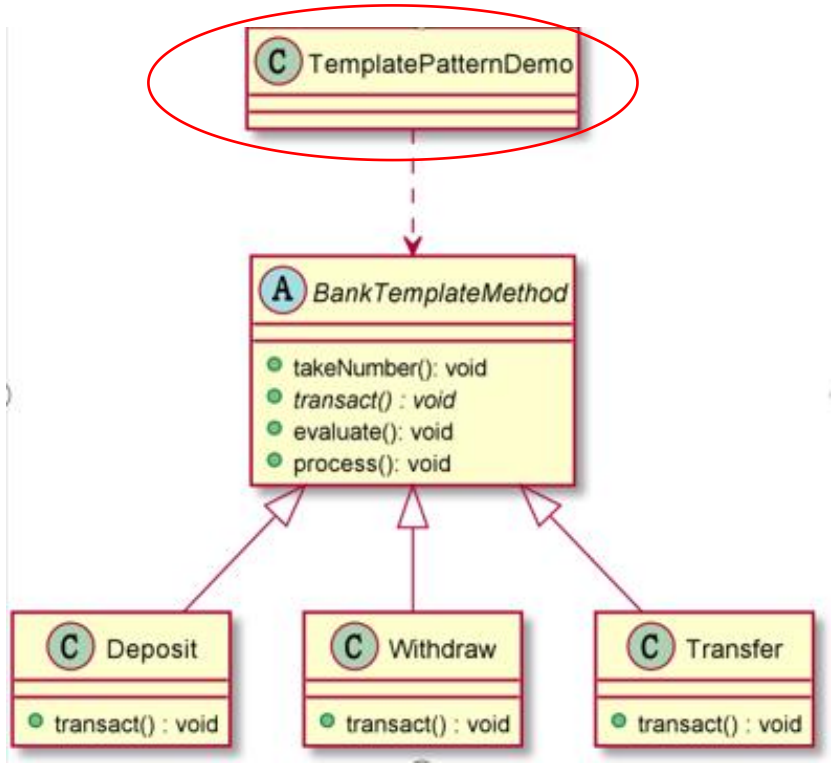


② 扩展该抽象类的实体类 Deposit、Withdraw 和 Transfer，它们重写了抽象类的某些方法。

```
public class Deposit extends BankTemplateMethod {

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

public class Transfer extends BankTemplateMethod {

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

public class Withdraw extends BankTemplateMethod {

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

实验步骤：银行业务办理举例（3）

UML 类 图 设 计



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

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

```
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();

    }
}
```

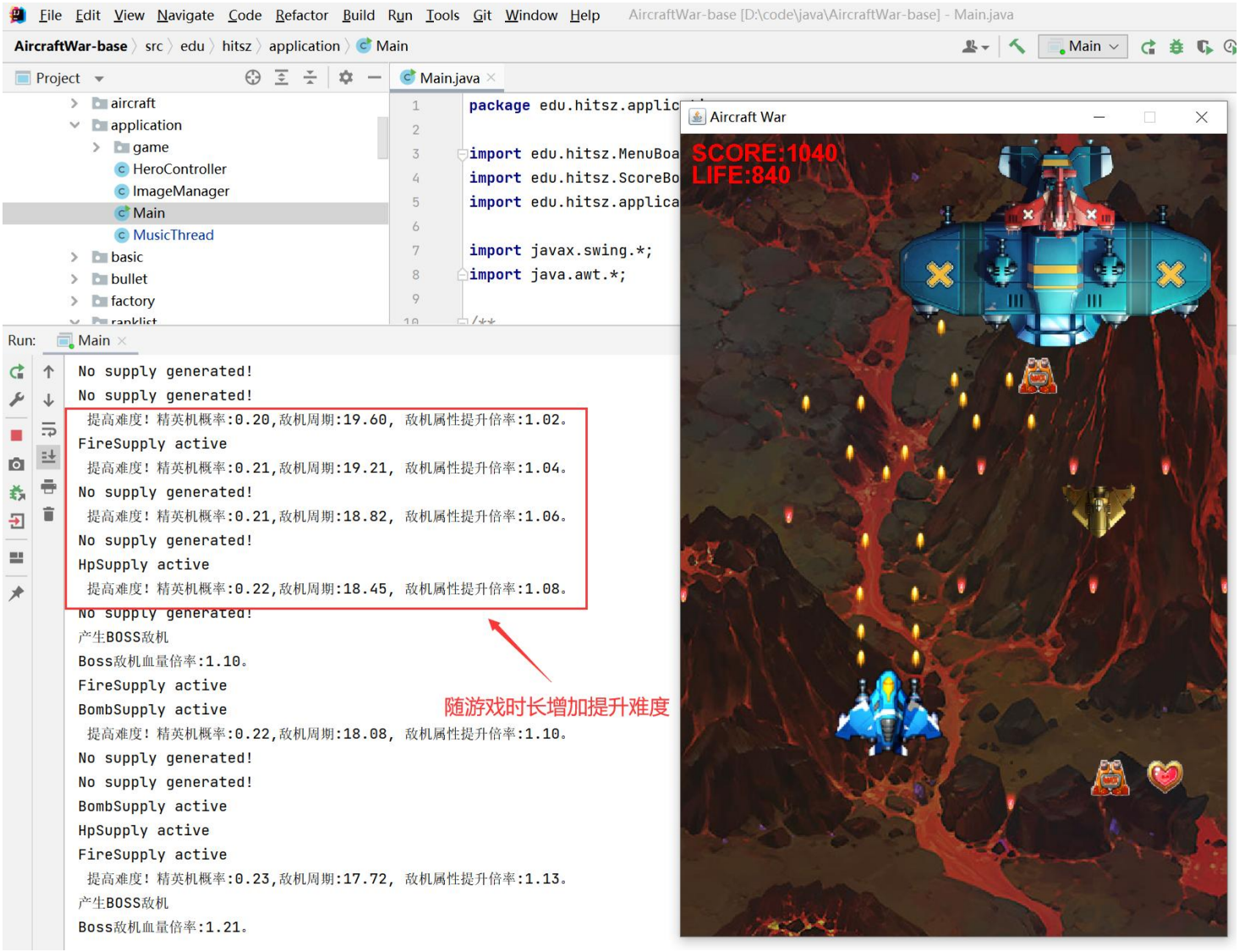
本次迭代开发的目标（1）

- ✓ 使用观察者模式实现炸弹道具；
- ✓炸弹生效时，它可清除界面上的所有的普通、精英敌机和敌机子弹，超级精英敌机血量减少，Boss敌机不受影响。
- ✓英雄机可获得坠毁的敌机分数。

本次迭代开发的目标（2）

✓ 使用模板模式实现三种游戏难度。

✓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机。



作业提交

- 提交内容

① 整个项目压缩包（整个项目压缩成zip包提交，包含代码、uml图等）

② 录制一段游戏的视频（小于2min），展示你的游戏的所有功能点和亮点；

③ 实验报告（按照实验报告模板）。

- 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