

（深圳）

实验报告

开课学期： 2024春季

课程名称： 面向对象的软件构造导论

实验名称： 飞机大战游戏系统的设计与实现

实验性质： 设计型

实验学时： 16 地点： T2210

学生班级： 5 班

学生学号： 220110515

学生姓名： 金正达

评阅教师：

报告成绩：

实验与创新实践教育中心制

2024年5月

# 实验环境

操作系统：windows 11

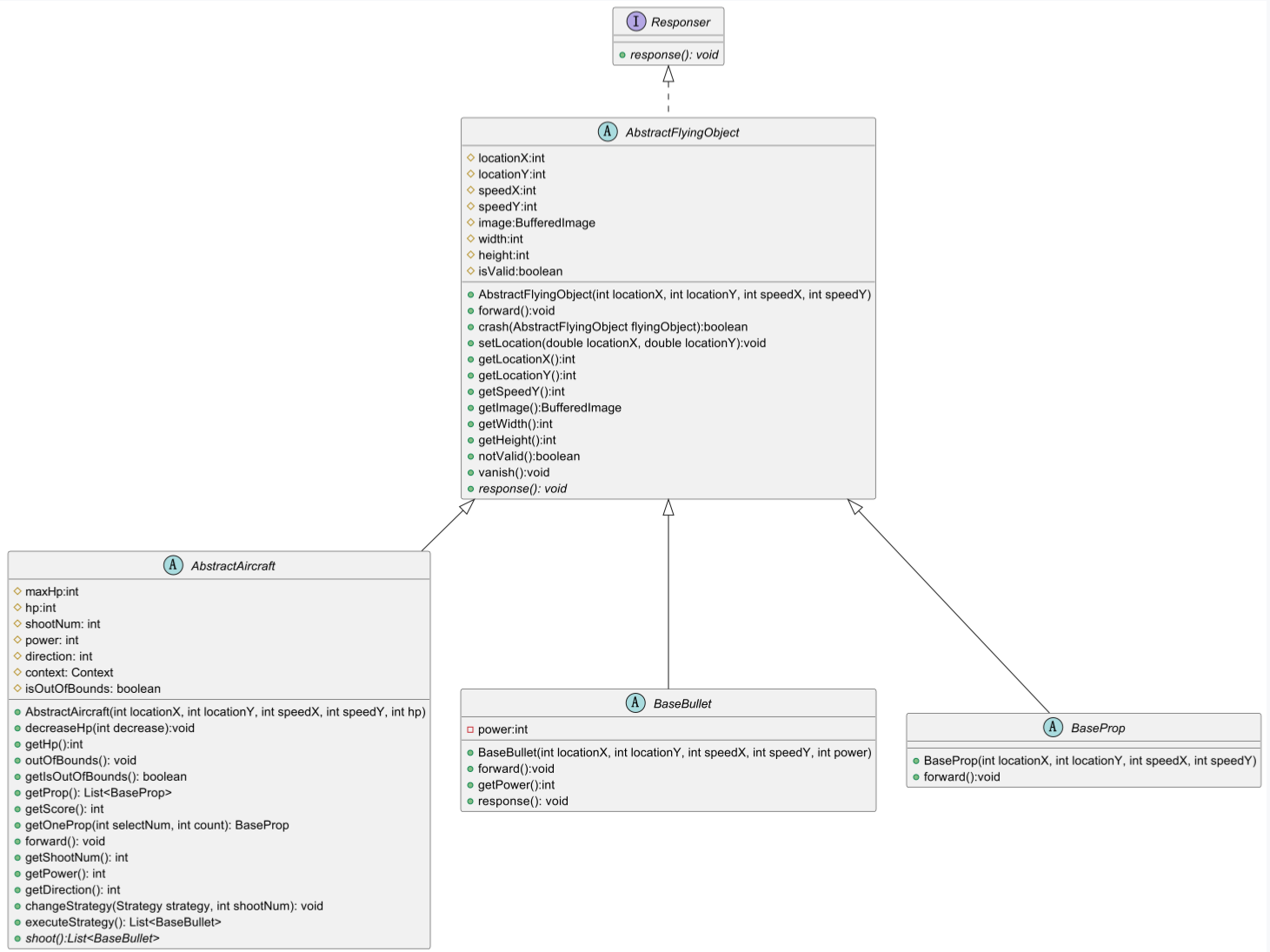
IDE：IntelliJ IDEA 2024.1.1

JDK：OpenJDK 21.0.2

# 实验过程

## 2.1类的继承关系

1. 抽象类的继承关系



AbstractFlyingObject 抽象类实现了 Responser 接口（用于观察者模式），提供了基本的飞行物位置、速度、大小、贴图相关的属性及方法。

AbstractAircraft 抽象类继承了 AbstractFlyingObject 抽象类，进一步增加了体力、射击子弹数、飞行方向、策略、是否出界等相关属性，还拥有增减体力、掉落道具、获得得分、更换和执行策略、射击的方法，重写了 forward 方法。

BaseBullet 抽象类继承了 AbstractFlyingObject 抽象类，进一步增加了子弹威力的属性，重写了 forward 和 response 方法。

BaseProp 抽象类继承了 AbstractFlyingObject 抽象类，重写了 forward 方法。

增加三个细分抽象类有助于分别将飞机、子弹、道具的共同行为属性抽象出来，减少代码的重复度，更利于复用方法。

2. 飞机类的继承关系



HeroAircraft 类继承了 AbstractAircraft 抽象类，增加了适配单例模式的属性和方法，重写了 shoot 方法，增加了 increaseHp 方法用于获得体力道具时恢复体力。

MobEnemy 类继承了 AbstractAircraft 抽象类，重写了 shoot、response、getScore 方法用于响应射击调用、炸弹道具生效和销毁时玩家能获得的得分。

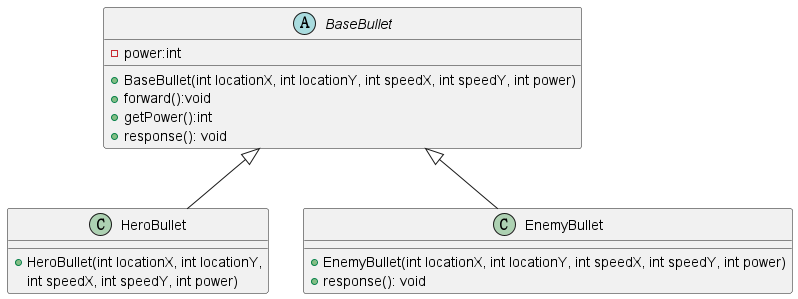
EliteEnemy 类继承了AbstractAircraft 抽象类，重写了 getProp 方法，用于在销毁时掉落道具，其余重写与 MobEnemy 类似。

ElitePlusEnemy 类继承了 AbstractAircraft 抽象类，重写方式与 EliteEnemy 相似。

BossEnemy 类继承了 AbstractAircraft 抽象类，相比上述敌机类没有重写 response 方法，即无需响应炸弹道具。

设计具体的飞机类将各类飞机职能分开进行管理，方便在游戏中具体对某个飞机类进行操作。

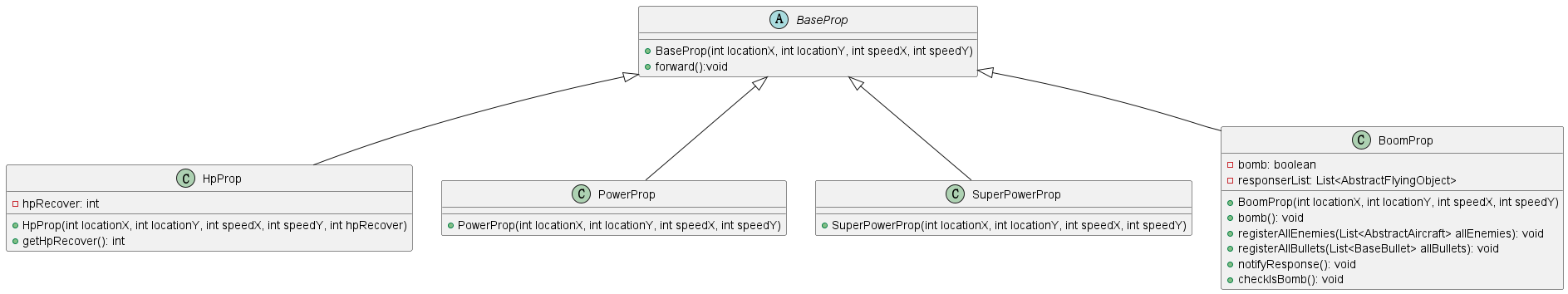
3. 子弹类的继承关系



HeroBullet 类继承了 BaseBullet 抽象类。

EnemyBullet 类继承了 BaseBullet 抽象类，重写了 response 方法用于响应炸弹道具。

4. 道具类的继承关系



HpProp 类继承了 BaseProp 抽象类，增加了 hpRecover 实例字段用于记录道具回复体力量，getHpRecover 方法用于返回 hpRecover 的值。

PowerProp、SuperPowerProp 类继承了 BaseProp 抽象类。

BombProp 类继承了 BaseProp 抽象类，增加了观察者模式相关方法和属性。

## 2.2设计模式应用

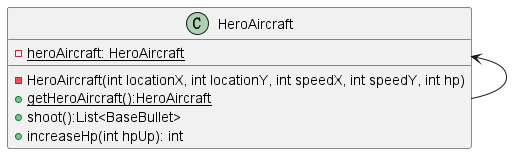
### 2.2.1单例模式

1. 应用场景分析

飞机大战中，一局的英雄机只能有一个，直接使用构造方法创建的英雄机并没有限制，可能存在英雄机被过多创建的可能。

在英雄机的创建过程中使用单例模式能够保证英雄机类只有一个实例对象，即全局游戏中只有一个英雄机。

1. 设计模式结构图



HeroAircraft 类应用单例模式，主要体现在它拥有私有属性 heroAircraft，私有构造函数 HeroAircraft 和 公开的 getHeroAircraft 方法 。

调用者只有通过 getHeroAircraft 方法获得该类的实例对象，调用 getHeroAircraft 方法时，该方法会在 heroAircraft不为 null 时调用 HeroAircraft 的构造函数创建实例对象赋值给 heroAircraft 属性并将其返回给调用者，保证了该类的仅有一个实例对象。

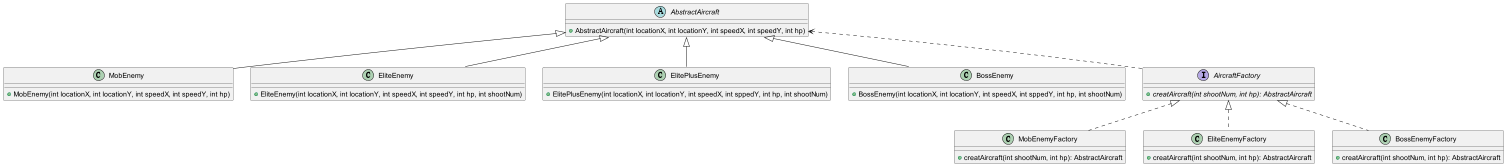
### 2.2.2工厂模式

1. 应用场景分析

飞机大战的敌机和道具创建过程都是在 Game 模块中直接调用各个类的构造函数并传递较多的参数来创建的，破坏了类的封装。

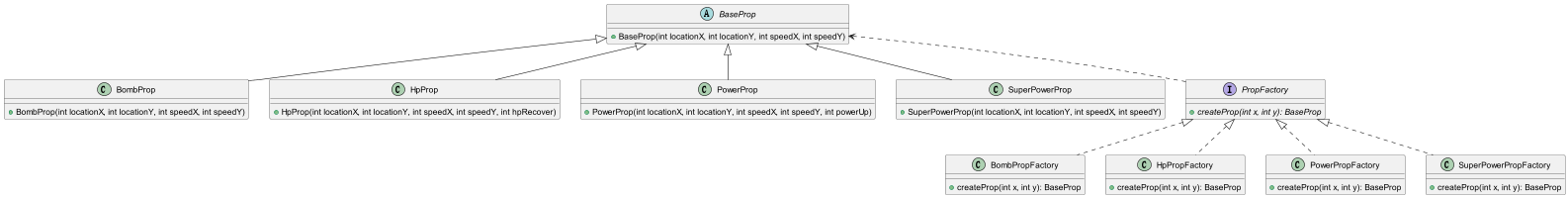
使用工厂模式创建敌机和道具，可以将敌机和道具的创建和使用解耦，提高系统的灵活性和可维护性。

1. 设计模式结构图
2. 飞机工厂模式



AbstractAircraft、MobEnemy、EliteEnemy、ElitePlusEnemy、BossEnemy 为产品类，拥有各自的构造函数，AircraftFactory、 MobEnemyFactory、EliteEnemyFactory、BossEnemyFactory 为工厂类（接口），拥有 creatAircraft 方法，用于创建各种具体的敌机。

1. 道具工厂模式



BaseProp、PowerProp、HpProp、BombProp、SuperPowerProp 为产品类，拥有各自的构造函数，PropFactory、PowerPropFactory、HpPropFactory、BombPropFactory、SuperPowerPropFactory 为工厂类（接口），拥有 creatProp 方法，用于创建各种具体的道具。

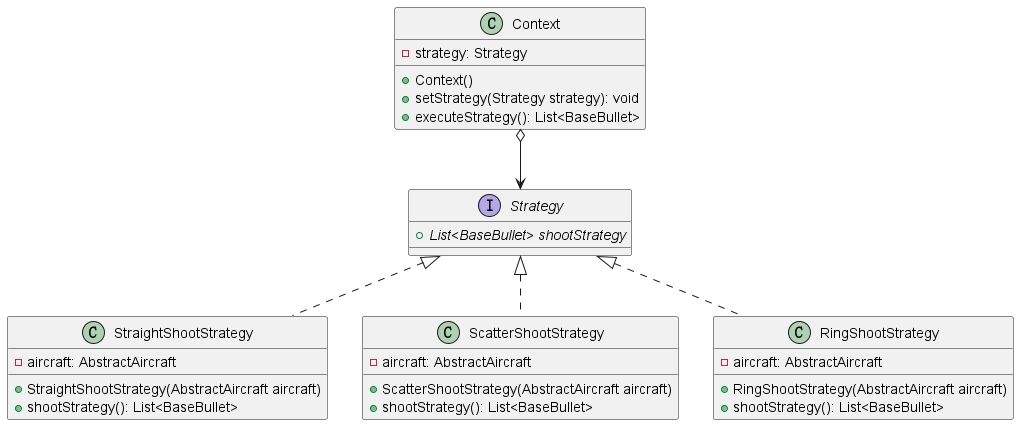
### 2.2.3策略模式

1. 应用场景分析

在飞机大战中，不同的飞机有不同的弹道，且英雄机在获得火力道具时会切换弹道，为每个飞机实现具体的弹道逻辑会使代码重复率高，不宜于管理和维护。

使用策略模式将各个弹道的实现进行封装，使各个弹道算法成为可互换的策略,使得代码更加清晰、灵活和易于维护。

1. 设计模式结构图



Strategy 接口的shootStrategy 方法返回当前策略

StraightShootStategy 类的shootStrategy 方法：返回直射弹道策略。

ScatterShootStrategy 类的shootStrategy 方法：返回散射弹道策略。

RingShootStrategy 类的shootStrategy 方法：返回环射弹道策略。

Context 类的私有strategy 字段：保存当前策略；Context 构造器：实例化 Context类；setStrategy 方法：设置当前策略为指定策略；executeStrategy 方法：执行当前策略。

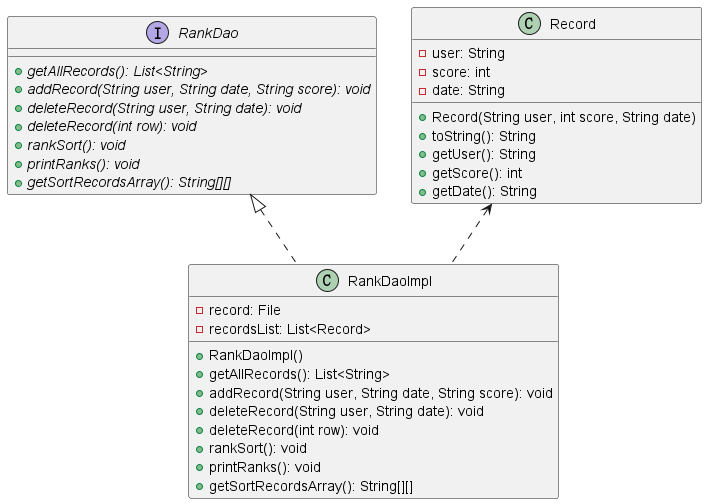
### 2.2.4数据访问对象模式

1. 应用场景分析

在飞机大战中，游戏结束的时候需要给出得分排行榜，因此需要记录每次游戏的相关数据，在主程序代码中直接操作数据会造成主程序代码繁杂冗余，也不利于管理和维护数据。

使用数据对象访问模式可以将对底层数据的操作从主程序中分离出来，分离了数据访问逻辑和主程序逻辑，上层逻辑无需考虑数据访问的细节。

1. 设计模式结构图



RankDao 接口：getAllRecords 方法：获得所有已有的记录；

addRecord 方法：向数据集中添加一条记录；

deleteRecord方法: 根据用户名和时间删除数据集对应的数据；

deleteRecord方法: 根据行数删除数据集对应数据；

rankSort 方法：根据得分对记录排序；

printRanks 方法：打印当前得分排行榜；

getSortRecords 方法；以数组形式返回排序完成的得分排行榜。

RankDaoImpl 类：私有 record 字段：记录当前存储数据所用文件；

私有 recordsList 字段：记录当前的所有记录；

其余方法均实现了 RankDao 接口。

Record 类：私有 user 字段：玩家用户名；

私有 score 字段：本次游戏得分；

私有 date 字段：本次游戏结束时间；

Record 构造器：新建一条记录；

toString 方法：将 Record 类转变为字符串输出；

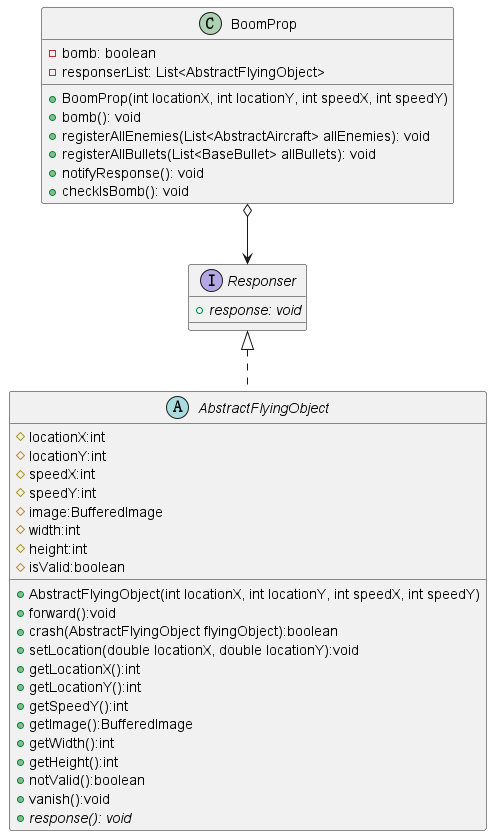
getUser、getScore、getDate 方法：获取对应私有字段。

### 2.2.5观察者模式

1. 应用场景分析

在炸弹道具生效时，部分敌机、敌机子弹需要响应炸弹道具，进行清空操作，使用观察者模式可以使观察者和被观察者之间的耦合度降低，被观察者无需知道具体观察者的实现，只需通知观察者接口；可以动态添加和删除观察者，无需修改被观察者的代码。这使得系统可以灵活地扩展新的功能和模块；系统可以在不修改现有代码的情况下进行扩展，符合软件设计的开闭原则（对扩展开放，对修改封闭）；一个被观察者的状态变化可以通知多个观察者，实现一种广播通信机制，简化多目标通知的复杂性。

1. 设计模式结构图



Responser 接口的 response 方法：在对炸弹道具进行响应时，响应对象应该调用的方法。

AbstractFlyingObject 抽象类实现了 Responser 接口，包含了一个 response 的空实现，其子类根据需要，若要响应炸弹道具，则需要重写该方法。

BombProp 类依赖 Responser 接口实现类实现功能，其拥有 bomb 字段指示炸弹道具是否生效，responserList字段用于集中管理响应者，bomb 方法用于使炸弹道具生效，registerAllEnemies 和 registerAllBullets 方法用于注册需要响应炸弹道具的敌机类和子弹类，notifyResponse 方法用于提醒所有注册的类对炸弹道具进行响应，checkIsBomb 方法用于检测炸弹道具是否生效并决定是否调用 notifyResponse。

### 2.2.6模板模式

1. 应用场景分析

简单模式：无 Boss 敌机，游戏难度不会随着时间增加而增加。

普通模式：有 Boss 敌机，每次生成其 Hp 保持不变，游戏难度随时间增加而增加，具体体现为：每隔一段时间，敌机最大数量增加1，所有敌机的 Hp 提高至原来的 1.1 倍，精英类敌机生成概率小幅度提升，敌机生成周期小幅度减小，敌机射击周期小幅度减小。

困难模式：有 Boss 敌机，每次生成其 Hp 增加，游戏难度随时间增加而增加，具体体现为：每隔一段时间，敌机最大数量增加3，所有敌机的 Hp 提高至原来的 1.5 倍，精英类敌机生成概率较大幅度提升，敌机生成周期较大幅度减小，敌机射击周期较大幅度减小。

在飞机大战中，整个游戏的算法步骤是固定的，但某些具体步骤在不同难度下需要不同实现时。这就需要在几个不同难度的子类中共享相同的代码，且某些操作需要执行一系列步骤，这些步骤的顺序是固定的，但具体实现是变化的。

使用模板模式可以将通用部分抽取到父类中，减少代码重复，提高代码复用性，

子类可以在不改变算法整体结构的情况下，通过覆盖父类的具体步骤方法来扩展新的行为，减少了代码的冗余。

父类提供了一个一致的算法框架，使得算法的步骤顺序对所有子类保持一致，确保算法的整体结构不变。

1. 设计模式结构图



GameTemplate 类是模板模式的模板类，GameTemplate：构造函数；action：模板方法，其中包含一系列游戏运行的逻辑子方法，不会被子类重写；noBossEnemyGenerator 方法：生成除 Boss 敌机之外的敌机类；enemyGnerator、increaseDifficulty 为 action 调用的子方法，设置为抽象以要求子类实现。

EasyGame、NormalGame、HardGame 类均继承自 GameTemplate 类，实现了 enemyGenerator 和 increaseDifficulty 方法，以适应不同难度的需要。

# 收获和反思

收获：通过本次一系列的飞机大战开发实验，我较为系统的了解、学习、实践了使用 java 这门面向对象的语言进行软件开发的流程，深入理解了 java 语言的特性和面向对象编程的思想方法，熟悉了六种面向对象的设计模式，学会了如何在开发过程中重构代码以不断提高代码的复用度、减少代码冗余度，锻炼了代码开发能力，为今后的学习、工作打下基础。

反思：在编写代码过程中，我一般会在代码较为复杂时才去尝试重构代码，这是不好的习惯，应该在编写代码之前进行设计，在写代码的过程中注意降低代码耦合度。在开发过程中，我应该先绘制 UML 类图，明确需求再进行代码编写，这样能在更上一层次对项目结构有总体的认识。