

（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： 22级计科5班

学生学号： 220110512

学生姓名： 宋夏冠男

评阅教师：

报告成绩：

实验与创新实践教育中心制

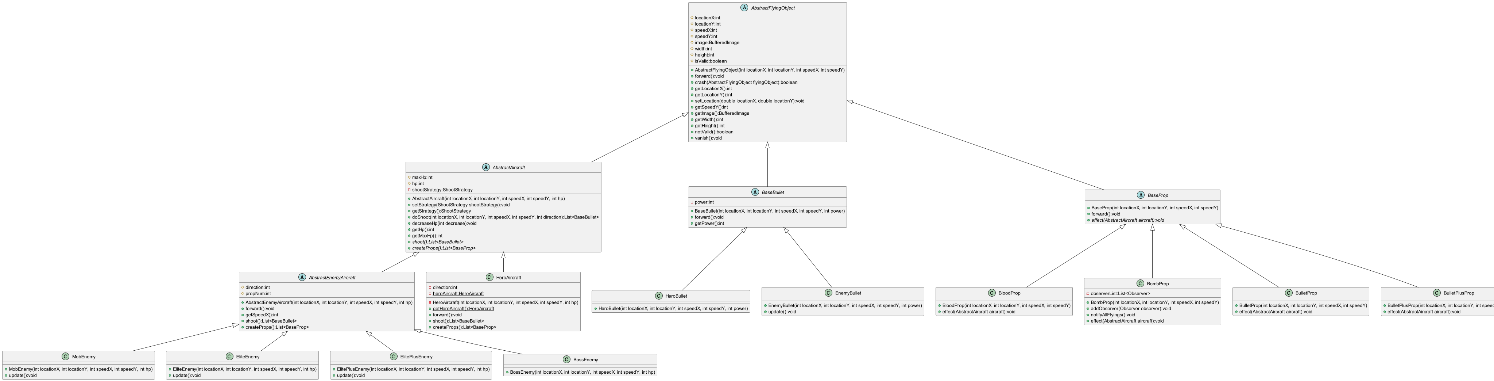
2024年5月

# 实验环境

1. Windows
2. IntelliJ IDEA 2023.3.6
3. OpenJDK 22

# 实验过程

## 2.1类的继承关系

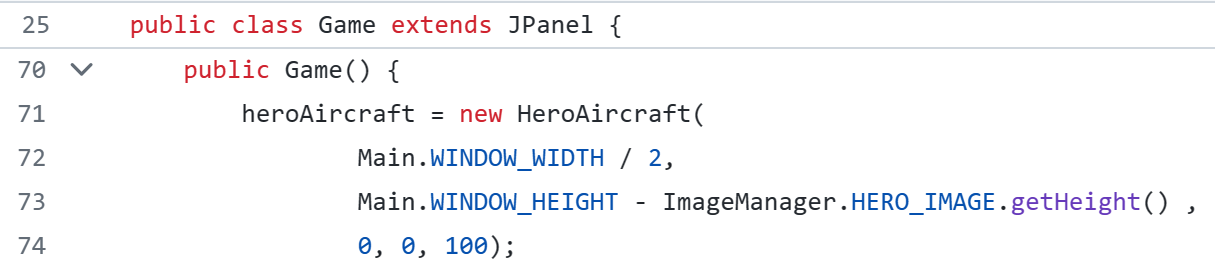


## 2.2设计模式应用

### 2.2.1单例模式

1. 应用场景分析

**应用场景**：英雄机（HeroAircraft），在飞机大战游戏中只有一架英雄机，由玩家通过鼠标控制移动，这符合单例设计模式的特点。对此，我们通过私有构造方法及在本类中创建静态私有对象来保证英雄机类（HeroAircraft）仅有一个实例，并提供一个访问它的全局访问点。同时为了保证线程安全，选择使用双重检查锁。

修改之前代码实现中存在的问题：

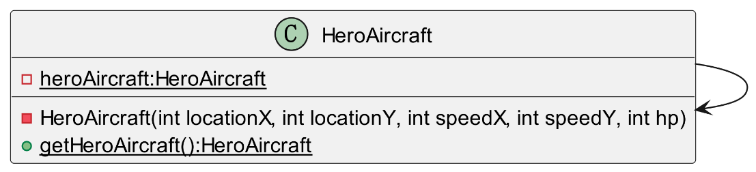
目前的代码，在Game中，创建对象和使用对象没有分离，耦合度较高，违反了单一职责原则；外部程序可以随意用new方法创建一个实例，无法保证英雄机的唯一性。

使用双重检查锁定的单例模式的优势：

使用单例模式，可以保证一个类只有一个实例，并且保证对唯一实例的受控访问；可以减少内存的开销，避免频繁地创造销毁对象，提高性能；可以避免对共享资源的多重占用；单例模式允许可变数目的实例，并且可以全局访问。

使用双重检查锁定，既可以保证线程安全，又可以避免浪费空间，并且可以一定程度上减少由于同步锁带来的效率低的问题。

1. 设计模式结构图



单例模式中只存在一个类，即英雄机（HeroAircraft），没有接口。

1. 关键属性：

- {static} heroAircraft: HeroAircraft：英雄机类（HeroAircraft）中创建英雄机对象（heroAircraft），用static修饰符，使得一个英雄机类只有唯一一个英雄机对象，保证了英雄机的全局唯一性，并且用private使得属性只能被私有访问。

1. 关键方法：

- HeroAircraft(int locationX, int locationY, int speedX, int speedY, int hp)：英雄机类的构造方法，用private修饰，保证外部无法通过构造方法实例出其他对象。

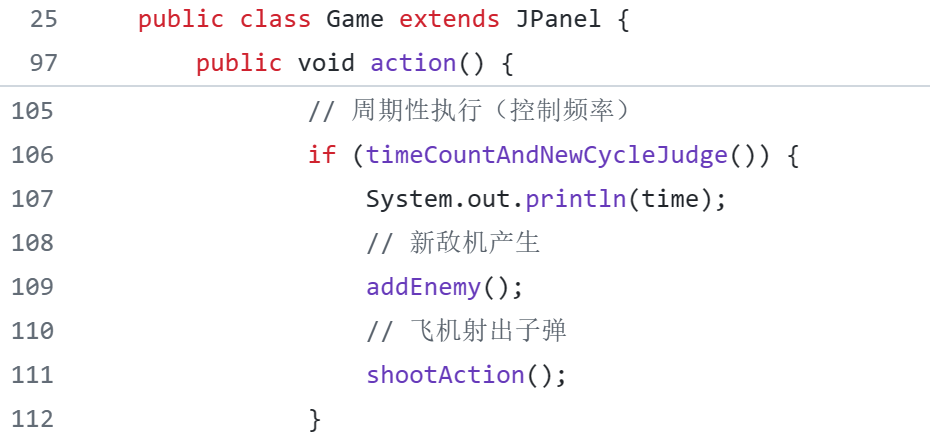
+ {static} getHeroAircraft(): HeroAircraft：用public修饰，提供一个访问私有属性的公共接口，便于全局访问对象，内部使用了双重检查锁和volatile关键字。

### 2.2.2工厂模式

1. 应用场景分析

（1）应用场景1：敌机

目前代码中存在的问题：





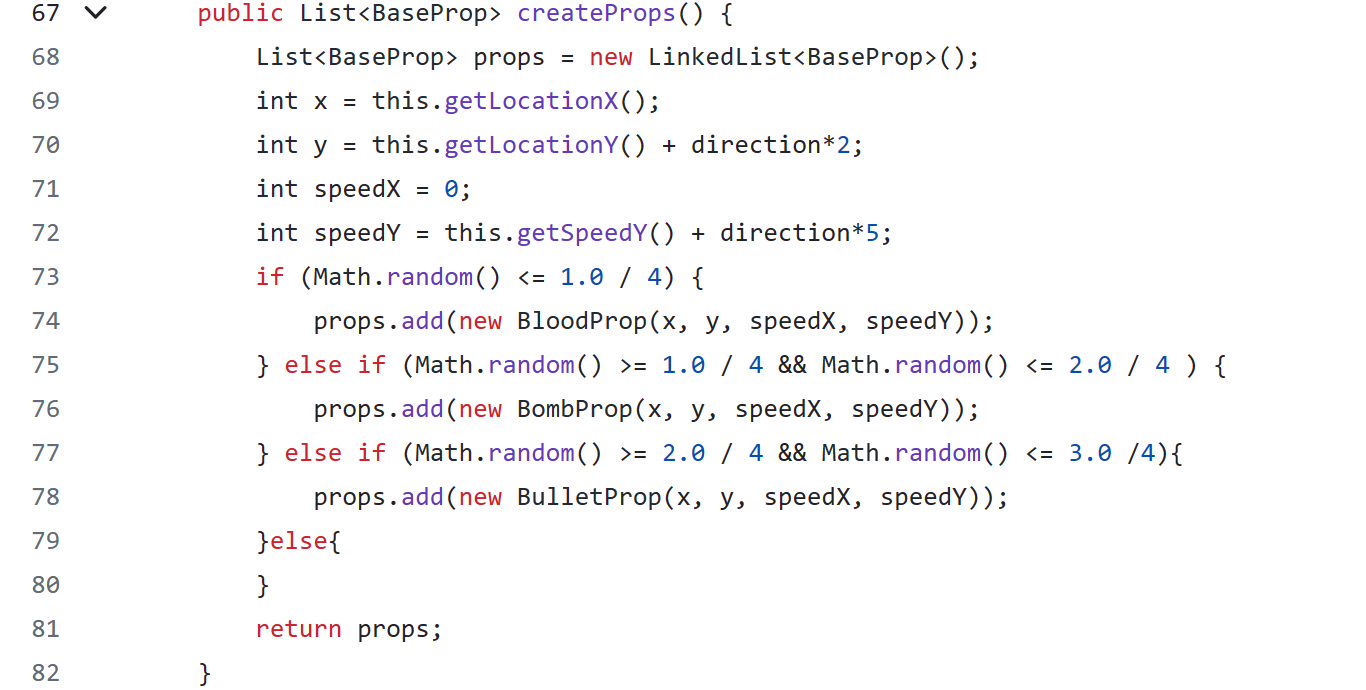
当前飞机大战中一共有四种敌机：MobEnemy, EliteEnemy, ElitePlusEnemy和BossEnemy。四种敌机的创建和使用没有分离，都在Game.java中，耦合度较高，违反了单一职责原则；如果要增加新的敌机，则需要修改代码，而非添加，违反了开闭原则；针对实现编程，而不是针对接口编程，违反了依赖倒转原则。

**使用工厂模式的优势：**使用工厂模式，定义一个用于创建对象的接口，让子类决定实例化哪个产品类对象。将创建对象和使用对象分离在不同的类中，工厂只负责创建对象，符合单一职责原则；可以通过添加工厂来新增具体的类，符合开闭原则；横向扩展方便。

（2）应用场景2：道具

目前代码中存在的问题：

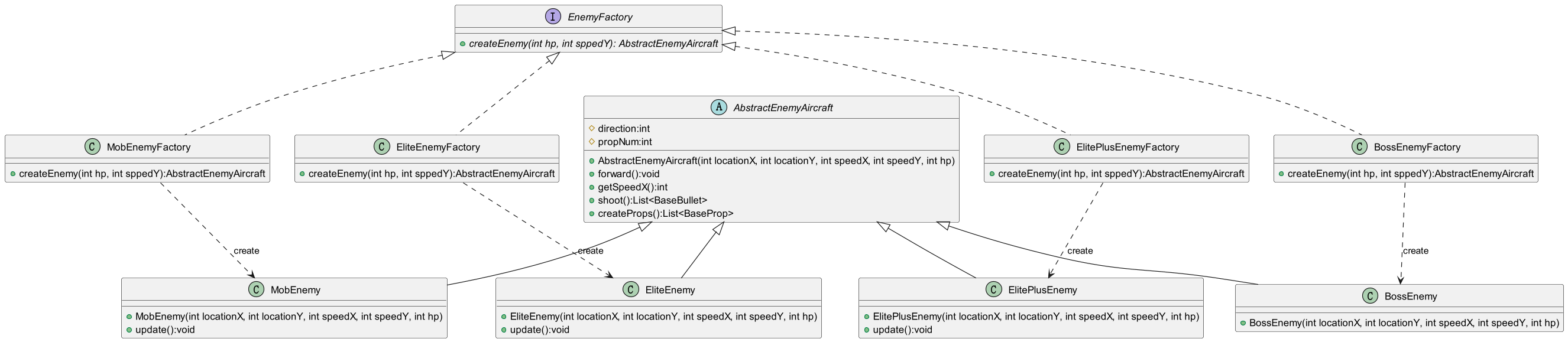




当前飞机大战中一共有四种道具：BloodProp, BombProp, BulletProp和BulletPlusProp。道具的创建和使用没有分离，违背了单一职责原则；如果要增加新的道具，需要修改代码，而非增加代码，违背了开闭原则；针对实现编程，而不是针对接口编程，违反了依赖倒转原则。

**使用工厂模式的优势：**使用工厂模式，定义一个用于创建对象的接口，让子类决定实例化哪个产品类对象。将创建对象和使用对象分离在不同的类中，工厂只负责创建对象，符合单一职责原则；可以通过添加工厂来新增具体的类，符合开闭原则；横向扩展道具方便。

1. 设计模式结构图
2. 敌机：



* 工厂系：

EnemyFactory：生产敌机的工厂接口

+ {abstract}createEnemy(int hp, int sppedY): AbstractEnemyAircraft：创建敌机对象的抽象方法，等待具体工厂类进行实现

MobEnemyFactory：生产普通敌机的具体工厂类

+ createEnemy(int hp, int sppedY):AbstractEnemyAircraft：创建普通敌机的具体方法

EliteEnemyFactory：生产精英敌机的具体工厂类

+ createEnemy(int hp, int sppedY):AbstractEnemyAircraft：创建精英敌机的具体方法

ElitePlusEnemyFactory：生产超级精英敌机的具体工厂类  
 + createEnemy(int hp, int sppedY):AbstractEnemyAircraft：创建超级精英敌机的具体方法

BossEnemyFactory：生产Boss敌机的具体工厂类  
 + createEnemy(int hp, int sppedY):AbstractEnemyAircraft：创建Boss敌机的具体方法

* 产品系：

AbstractEnemyAircraft：敌机抽象类  
 # direction:int：子弹飞行方向，敌机子弹向下飞行，为1  
 # propNum:int：敌机坠毁产生道具的最大数量  
 + AbstractEnemyAircraft(int locationX, int locationY, int speedX, int speedY, int hp)：抽象敌机的构造方法  
 + forward():void：敌机移动方法  
 + getSpeedX():int：获取敌机水平速度  
 + shoot():List<BaseBullet>：敌机发射子弹  
 + createProps():List<BaseProp>：敌机产生道具

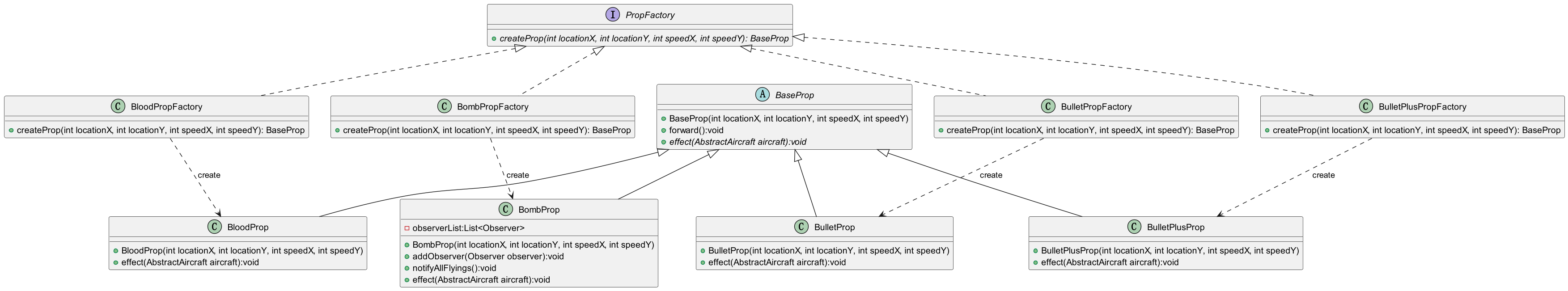
MobEnemy：普通敌机类  
 + MobEnemy(int locationX, int locationY, int speedX, int speedY, int hp)：构造方法  
 + update():void：对炸弹道具的通知做出反应，坠毁

EliteEnemy：精英敌机类  
 + EliteEnemy(int locationX, int locationY, int speedX, int speedY, int hp)：构造方法  
 + update():void：对炸弹道具的通知做出反应，坠毁

ElitePlusEnemy：超级精英敌机类  
 + ElitePlusEnemy(int locationX, int locationY, int speedX, int speedY, int hp)：构造方法  
 + update():void：对炸弹道具的通知做出反应，减少20血量

BossEnemy：Boss敌机类  
 + BossEnemy(int locationX, int locationY, int speedX, int speedY, int hp)：构造方法

1. 道具



* 工厂系：

PropFactory：生产道具的工厂接口

+ {abstract} createProp(int locationX, int locationY, int speedX, int speedY):BaseProp：创建道具对象的抽象方法  
BloodPropFactory：生产加血道具的具体工厂类  
 + createProp(int locationX, int locationY, int speedX, int speedY): BaseProp：创建加血道具的具体方法

BombPropFactory：生产炸弹道具的具体工厂类  
 + createProp(int locationX, int locationY, int speedX, int speedY): BaseProp：创建炸弹道具的具体方法

BulletPropFactory：生产火力道具的具体工厂类  
 + createProp(int locationX, int locationY, int speedX, int speedY): BaseProp：创建火力道具的具体方法

BulletPlusPropFactory：生产超级火力道具的具体工厂类  
 + createProp(int locationX, int locationY, int speedX, int speedY): BaseProp：创建超级火力道具的具体方法

* 产品系：

BaseProp：道具抽象类  
 + BaseProp(int locationX, int locationY, int speedX, int speedY)：构造方法  
 + forward():void：道具移动  
 + {abstract} effect(AbstractAircraft aircraft):void：道具生效的抽象方法

BloodProp：加血道具类  
 + BloodProp(int locationX, int locationY, int speedX, int speedY)：构造方法  
 + effect(AbstractAircraft aircraft):void：加血道具生效的具体方法，得到道具后在不超过最大血量的前提下加30血量  
BombProp：炸弹道具类

- observerList:List<Observer>：观察者列表（普通敌机、精英敌机、超级精英敌机、敌机子弹）  
 + BombProp(int locationX, int locationY, int speedX, int speedY)：构造方法  
 + addObserver(Observer observer):void：增加观察者  
 + notifyAllFlyings():void：通知所有观察者  
 + effect(AbstractAircraft aircraft):void：炸弹道具生效的具体方法（通知所有观察者）  
BulletProp：火力道具类  
 + BulletProp(int locationX, int locationY, int speedX, int speedY)：构造方法  
 + effect(AbstractAircraft aircraft):void：火力道具生效的具体方法，英雄机弹道切换为散射

BulletPlusProp：超级火力道具类  
 + BulletPlusProp(int locationX, int locationY, int speedX, int speedY)：构造方法  
 + effect(AbstractAircraft aircraft):void：超级火力道具生效的具体方法，英雄机弹道切换为环射

### 2.2.3策略模式

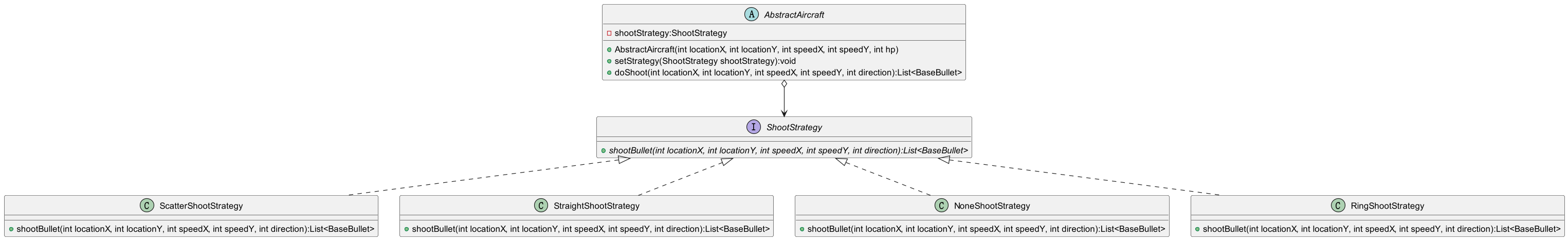
1. 应用场景分析

1）**应用场景**：处理英雄机和飞机的射击方式时会应用到策略模式，以实现直射、散射和环射三种子弹发射弹道。

2）**存在的问题**：如果不使用策略模式的话，每种英雄机的射击方式以及火力道具生效的代码之间有大量的重复，代码复用性差，算法无法独立“Client”变化，代码耦合度高。

3）**策略模式的优势**：各具体策略类都实现同一个接口，因此可以自由切换：飞机在射击时无需关心射击方式的具体实现，只需要选择合适的射击策略即可。同时，策略模式易于扩展：增加一个新的策略只需要添加一个具体的策略类即可，基本不需要改变原有的代码，符合“开闭原则”，充分体现了面向对象设计思想。此外，策略模式可以避免多重条件判断，使代码逻辑更明晰，代码更简洁。

1. 设计模式结构图



AbstractAircraft：抽象飞机类（环境/上下文类）

private ShootStrategy shootStrategy：私有属性的射击策略（成员变量）

public void setStrategy(ShootStrategy shootStrategy)：对外提供的修改策略的方法，以实现对射击方式的自由切换

public List<BaseBullet> doShoot(int locationX, int locationY, int speedX, int speedY, int direction)：射击策略的实现，即调用策略的shootBullet方法

ShootStrategy: 抽象射击策略（接口），其中包含一个抽象方法shootBullet (int locationX, int locationY, int speedX, int speedY, int direction): List<BaseBullet>，并有如下四个实现类，分别代表空射、直射、散射和环射：

NoneShootStrategy：空射击策略（具体策略），返回一个空子弹列表；

StraightShootStrategy：直射策略（具体策略），仅射出1颗子弹，且子弹仅具有y轴速度，子弹射击方向、类型及火力值由飞机类型（敌机/英雄机）决定；

ScatterShootStrategy：散射策略（具体策略），射出3颗子弹，子弹既具有y轴速度也具有x轴速度，排列呈扇形。子弹射击方向、类型及火力值由飞机类型（敌机/英雄机）决定。

RingShootStrategy：环射策略（具体策略），射出20颗子弹，子弹既具有y轴速度也具有x轴速度，排列呈环形。子弹射击方向、类型及火力值由飞机类型（敌机/英雄机）决定。

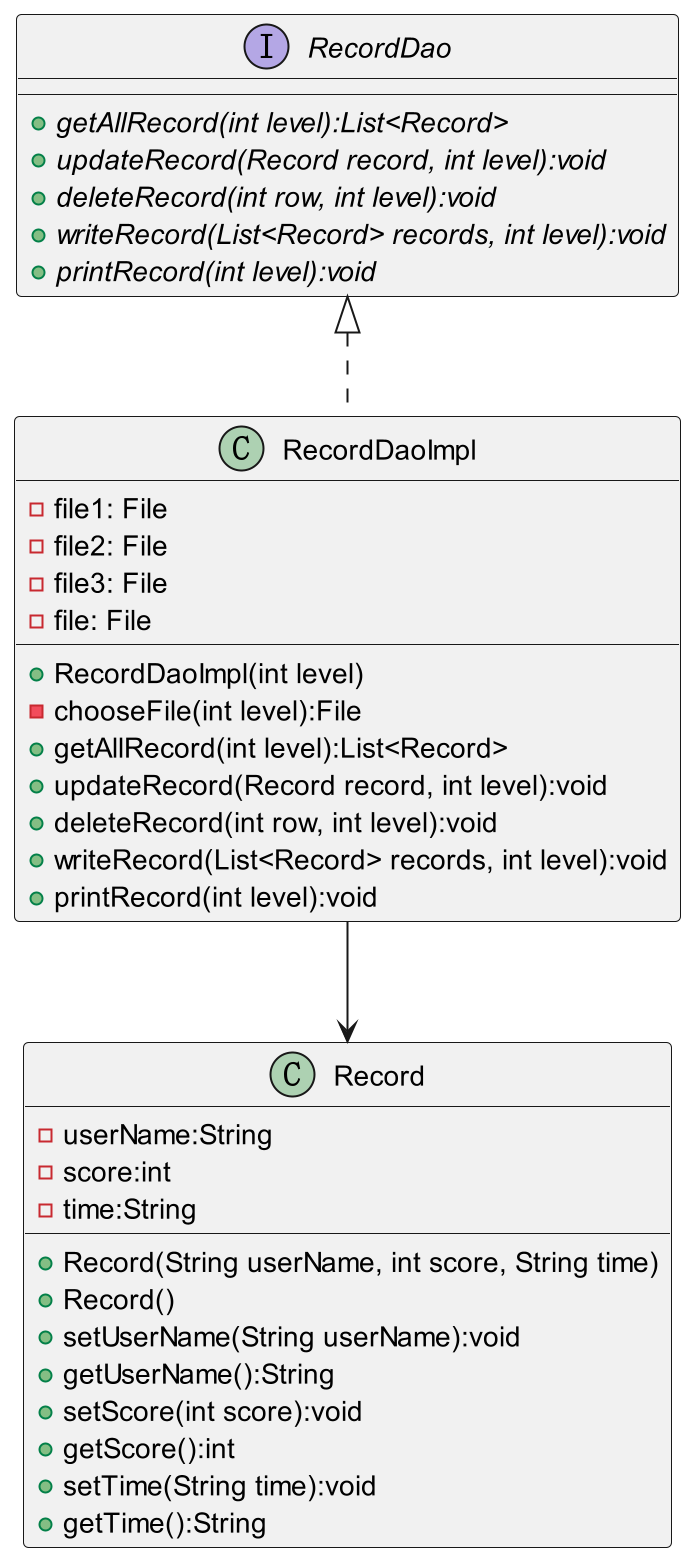
### 2.2.4数据访问对象模式

1. 应用场景分析

1）**应用场景**：每局游戏结束后，记录用户名、分数、当前时间等信息时需要用到数据访问对象模式，并通过IO流保存至对应的游戏难度文件中。

2）**数据访问对象模式的优势**：数据访问对象模式可以将对数据的基础操作实现封装在DAO类，只提供高级的数据操作方法，实现高级操作时不需要知道具体的实现方法，封装了变化；将读写数据的功能解耦出来，符合单一职责原则。同时，由于新增了DAO层，隔离了数据层，又不会影响到服务或者实体对象与“数据库”交互。

1. 设计模式结构图



**RecordDao**：数据访问对象接口，定义了在Record类型对象上要执行的标准操作：读取文件中全部记录、增加（更新）记录、删除记录、将记录写入文件、打印文件中全部记录，方法如下：

+ {abstract} getAllRecord(int level):List<Record>：获取文件中的所有记录对象  
+ {abstract} updateRecord(Record record, int level):void：将新的记录加入榜单，并对榜单内记录按照分数从高到低排序   
+ {abstract} deleteRecord(int row, int level):void：删除榜单中某个用户的记录  
+ {abstract} writeRecord(List<Record> records, int level):void：将记录写入文件内  
+ {abstract} printRecord(int level):void：将文件内的榜单记录打印到控制台

**RecordDaoImpl**：实现了RecordDao接口，实现了读取文件中全部记录、增加（更新）记录、删除记录、将记录写入文件、打印文件中全部记录的具体操作，关键属性为：

- file1: File：储存简单模式下的排行榜数据  
- file2: File：储存普通模式下的排行榜数据  
- file3: File：储存困难模式下的排行榜数据

- file: File：对象序列化写入的文件

其他方法：

- chooseFile(int level):File：根据游戏难度，选择对象序列化写入的文件以及读取的文件

**Record**：记录对象，主要包含属性（即用户信息：用户名、分数、时间）以及对应的getter和setter方法

用户名：- userName:String

+ setUserName(String userName):void  
+ getUserName():String

分数： - score:int

+ setScore(int score):void  
+ getScore():int

时间： - time:String

+ setTime(String time):void  
+ getTime():String

### 2.2.5观察者模式

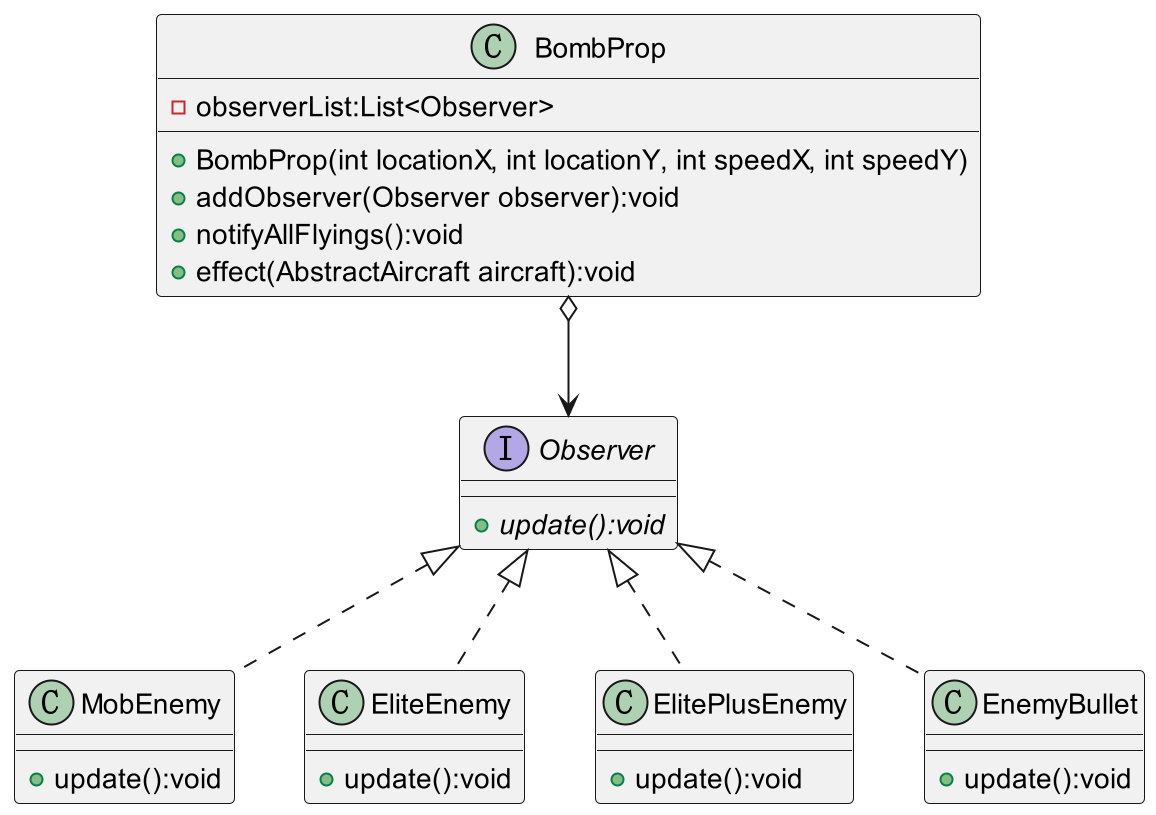
1. 应用场景分析

应用场景：炸弹道具。炸弹道具可清除界面上的所有普通敌机、精英敌机和敌机子弹，并让超级精英敌机减少一定的血量。英雄机可获得坠毁的敌机分数。

在应用观察者模式时，炸弹道具为观察目标，普通敌机、精英敌机、超级精英敌机及敌机子弹为观察者，符合一对多的依赖关系。一旦英雄机碰撞到炸弹道具，炸弹道具立即生效，通知各个观察者，并使所有普通敌机、精英敌机和敌机子弹消失，并让超级精英敌机减少一定的血量，但坠毁的敌机不会掉落道具。

观察者模式的优势：1）可以实现表示层和数据逻辑层的分离；2）在观察目标和观察者之间建立一个抽象的耦合；3）支持广播通信，简化了一对多系统设计的难度；4）符合开闭原则，增加新的具体观察者无须修改原有系统代码，在具体观察者与观察目标之间不存在关联关系的情况下，增加新的观察目标也很方便。

1. 设计模式结构图

****

BombProp：具体观察目标，炸弹道具类，关键属性及方法如下：

- observerList:List<Observer>：观察者列表（普通敌机、精英敌机、超级精英敌机、敌机子弹）  
 + BombProp(int locationX, int locationY, int speedX, int speedY)：构造方法  
 + addObserver(Observer observer):void：增加观察者  
 + notifyAllFlyings():void：通知所有观察者  
 + effect(AbstractAircraft aircraft):void：炸弹道具生效的具体方法（通知所有观察者）  
Observer：观察者接口，关键方法如下：  
 + {abstract} update():void：更新方法，在得到观察目标通知时做出相应反应  
四种具体观察者：具体实现观察者接口内的抽象观察方法，对观察目标的通知做出相应的反应：

MobEnemy：+ update():void：对炸弹道具的通知做出反应，坠毁

EliteEnemy： + update():void：对炸弹道具的通知做出反应，坠毁

ElitePlusEnemy：+ update():void：对炸弹道具的通知做出反应，减少20血量

EnemyBullet：+ update():void：对炸弹道具的通知做出反应，消失

### 2.2.6模板模式

1. 应用场景分析

游戏难度设计：

|  |  |  |  |
| --- | --- | --- | --- |
| **因素** | **简单模式** | **普通模式** | **困难模式** |
| Boss敌机 | 无 | 有 | 有 |
| 难度是否随时间增加 | 否 | 是 | 是 |
| 敌机最大数量 | 4 | 初始为6，每经过2s增加1 | 初始为8，每经过2s增加1 |
| 敌机射击周期、敌机产生周期 | 600 | 初始为500，每经过2s减少10，直到减少为400 | 初始为460，每经过2s减少20，直到减少为300 |
| 英雄机射击周期 | 600 | 初始为450，每经过2s减少10，直到减少为350 | 初始为460，每经过2s减少10，直到减少为340 |
| Boss敌机产生的分数阈值间隔 | - | 初始为300，每经过2s减少10，直到为150 | 初始为300，每经过2s减少20，直到减少为100 |
| 每次召唤Boss敌机血量是否增加 | - | Boss敌机血量不变，始终为150 | 初始为350，每次召唤Boss敌机血量增加200 |
| 精英机概率 | 0.3 | 初始为0.3，每经过2s增加0.04，直到为0.45 | 初始为0.3，每经过2s增加0.05，直到增加为0.55 |
| 敌机血量 | 普通敌机为30，精英敌机和超级精英敌机为60 | 普通敌机初始为30，精英敌机和超级精英敌机初始为60，每经过2s血量增加1，增加了15血量后不再增加 | 普通敌机初始为30，精英敌机和超级精英敌机初始为60，每经过2s血量增加2，增加了20血量后不再增加 |

应用场景：三种游戏难度

用户进入游戏界面后，可选择某种游戏难度：简单/普通/困难。由于三种游戏难度存在很多共性，程序流程大体一致，仅飞机属性、难度设置等部分存在区别，因此使用模板模式进行三种游戏难度的选择。我们将不变的代码、控制游戏流程的模板方法放在Game(父类)里，将改变的代码由三种不同的难度的具体子类实现。

模板模式的优势：1）提高了代码复用性，将相同部分的代码放在抽象的父类中，而将不同的代码放入不同的子类中；2）实现了反向控制：通过一个父类调用其子类的操作，通过对子类的具体实现扩展不同的行为，实现了反向控制，并符合“开闭原则”；3）减少耦合，封装变化。

1. 设计模式结构图



Game：抽象游戏类，提供框架，关键属性及方法为：  
 # enemyMaxNumber:int：屏幕中出现的敌机最大数量  
 # cycleDuration:int：敌机子弹反射周期、敌机产生周期  
 # heroCycleDuration:int：英雄机子弹发射周期  
 - cycleTime:int：用于判断是否进入新的敌机子弹发射周期、敌机产生周期  
 - heroCycleTime:int：用于判断是否进入新的英雄机子弹发射周期  
 # threshold:int：开始出现Boss敌机的分数阈值  
 # increaseThreshold:int：出现Boss敌机的分数间隔阈值  
 # difficulty:int：游戏难度  
 # eliteProbability:double：产生精英敌机的概率  
 + Game()：构造方法  
 + {abstract} increaseDifficulty():void：提升游戏难度的抽象方法，等待子类具体实现  
 + action(): void：游戏启动入口，执行游戏逻辑，模板方法，按顺序调用其中的方法  
 - timeCountAndNewCycleJudge():boolean：判断是否进入新的敌机子弹发射周期、敌机产生周期  
 - heroTimeCountAndNewCycleJudge():boolean：判断是否进入新的英雄机子弹发射周期  
 + {abstract} addEnemy():void：产生敌机的抽象方法，等待子类具体实现  
三种具体子类，实现了抽象类的抽象方法：

EasyGame：简单模式游戏类

+ increaseDifficulty():void：简单模式游戏难度不变，函数体为空

+ addEnemy():void：随机生成普通敌机、精英敌机、超级精英敌机

MediumGame：简单模式游戏类

- increasehp:int：敌机血量增加量

+ increaseDifficulty():void：提高游戏难度（具体参见上问表格）

+ addEnemy():void：随机生成普通敌机、精英敌机、超级精英敌机，分数超过阈值，生成Boss敌机

HardGame：简单模式游戏类

- increasehp:int：敌机血量增加量

+ increaseDifficulty():void：提高游戏难度（具体参见上问表格）

+ addEnemy():void：随机生成普通敌机、精英敌机、超级精英敌机，分数超过阈值，生成Boss敌机，且每次召唤，Boss敌机血量增加

# 收获和反思

飞机大战的实验终于结束，一路走来，有深夜鏖战实验室的苦楚，有百思不得其解的纳罕，但更多的是将理论知识转为实践成果的满足。

从实验一中老师提供的只能进行“英雄机发射子弹击落普通敌机”的模板程序，到一步步实践设计模式，重构代码、添加功能，最终在实验六结束之时，完成一份基本完整的飞机大战程序，没有什么比亲手使这万丈高楼平地起更让人欣喜。

本次实验难度适中，对逻辑性的要求并不算很高，更多地带领我们了解Java语法、掌握面向对象的思想、体味常见的几种设计模式的优势。对于我而言，实验一和实验五难度最大，实验一的难点在于，从来没有接触过这么大的项目代码，最初看着这么多文件夹只觉得压力山大，于是一遍遍地阅读代码、理解代码，将每个文件之间的关系串联起来，渐渐的，我已经对项目结构十分了解；实验五的难点在于JavaSwing和I/O，需要对整个排行榜的运行逻辑有着准确的把握，并且对细节的要求很高，一旦文件读写细节出现纰漏，便会导致程序报错，实验五也是我耗时最长的一次实验，一次次的试验、失败，试验、失败……终于把文件读写搞定啦！！除此之外，在完成实验三时，我因为在产生敌机的过程中，将null加入List中，导致遍历敌机列表时出错，程序卡死，而我当时由于对IDEA的debug不太熟悉，只好一个节点一个节点print大法，最终终于在实验房老师的帮助下找到了这个隐蔽性较强的问题。

特别感谢实验汪老师、房老师，理论课马老师在课上课下对我提出的问题耐心、详细的回答，心里暖暖的，嘿嘿（花痴脸）

坦白说，在没有开始之前，我对这个闻名已久的飞机大战实验有一点害怕和抗拒，但当我如今真正回头看这几个月，我觉得我在这个实验中收获了许许多多的快乐，让我对知识的掌握更加深入，也引导我对就业方向产生更多的思考。

再次感谢这门课程在我枯燥的学习生活中增添了一抹亮色，虽然过程有时坎坷，但结局还算圆满。不过，也有一点小建议，如果老师能将同学私聊提出的共性问题在群里做统一解答，有助于同学们自检以及提前防范可能的错误。

最后，衷心祝愿课程越办越好！