

（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： T2608

学生班级： 计科一班

学生学号： 220110112

学生姓名： 姜懿城

评阅教师：

报告成绩：

实验与创新实践教育中心制

2024年3月

# 实验环境

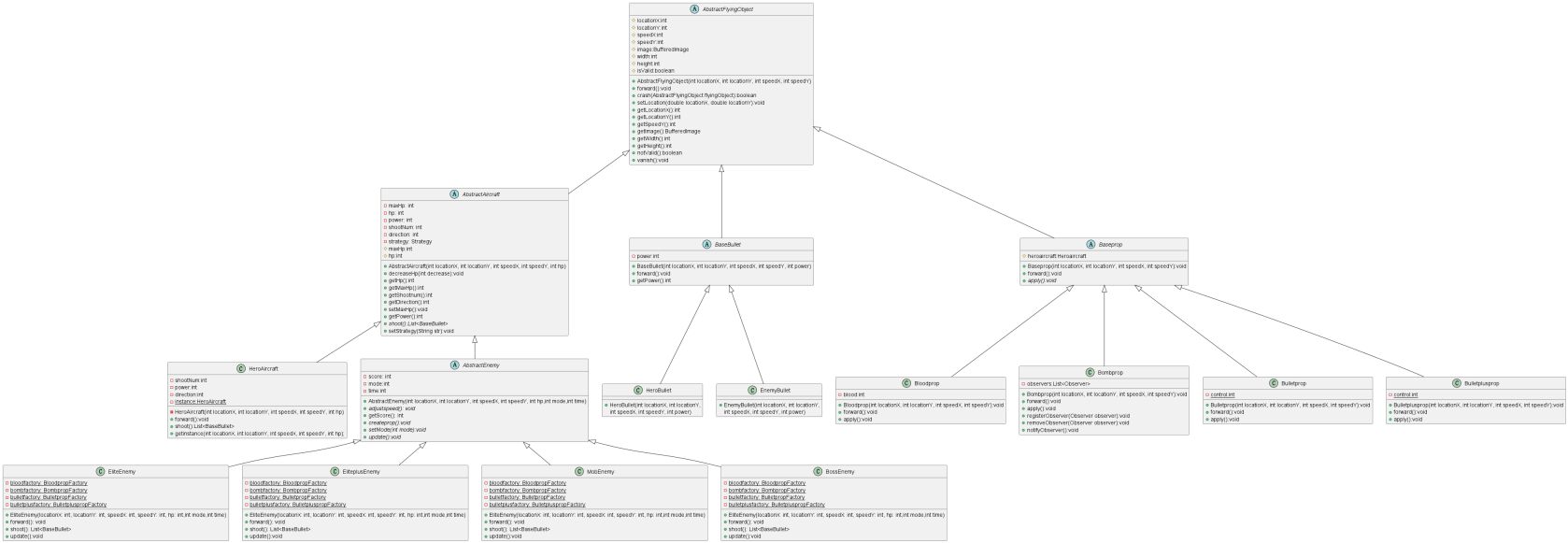
Windows系统，使用IntelliJ IDEA Community Edition 2023.3.3作为IDE

# 实验过程

以下UML结构图请更新为整个项目最终提交版本。

## 2.1类的继承关系

*请根据面向对象设计原则，分析和设计游戏中的所有飞机类、道具类和子弹类，并使用 PlantUML 插件绘制相应的 UML 类图及继承关系，类图中需包括英雄机、所有敌机、道具、子弹及它们所继承的父类。*



## 2.2设计模式应用

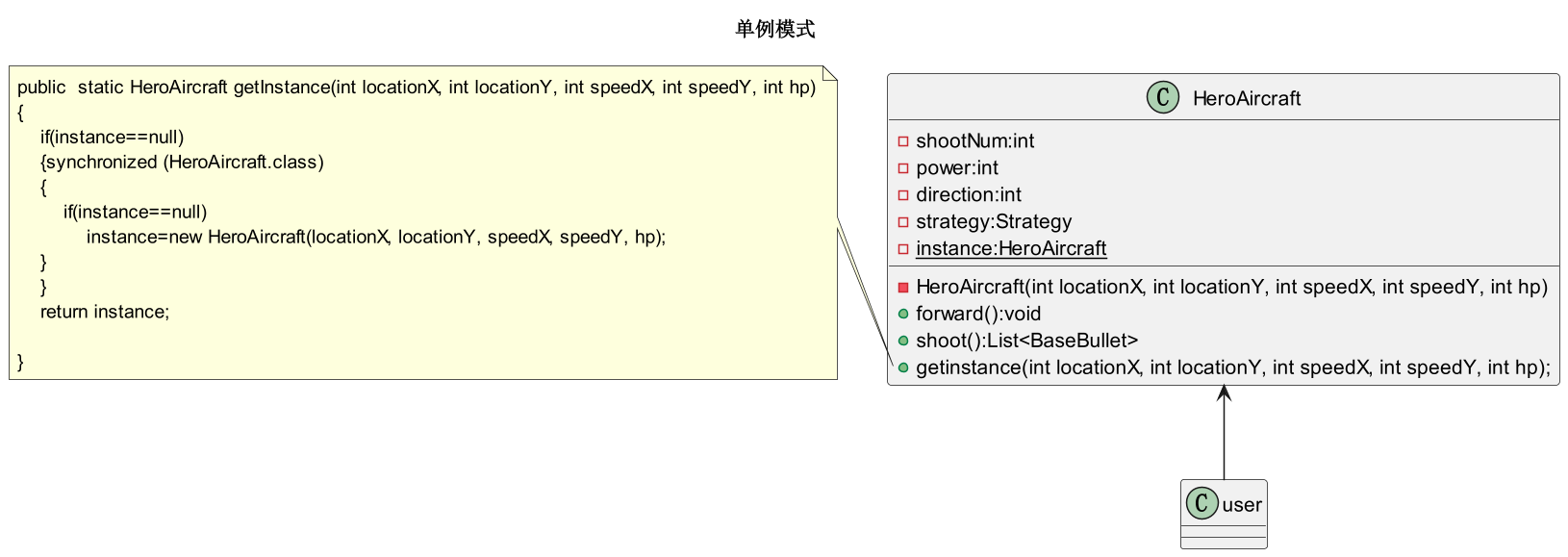
### 2.2.1单例模式

1. 应用场景分析

**应用场景：创建英雄机**

**目前问题：英雄机应是唯一的对象，而从外部可以创建不限数量个英雄机，违反了设计需求。使用该模式可以避免在外部创建英雄机对象时，创建多个对象，导致其不唯一产生混淆。**

1. 设计模式结构图



**解题思路：使用双重检查锁定（DCL）方法；**

**首先将类的构造函数声明为私有的，防止其被外部调用，声明一个静态实例对象，并定义一个公有的getinstance方法代替构造函数：**

**当instance未初始化时，调用一个加了线程锁的块：为避免多线程竞争，故需判断instance是否初始化，当未初始化时，调用自身的构造函数创建一个新示例。**

**并在最后返回已经初始化的instance对象。**

**关键属性：instance**

**创建一个静态实例对象，并在每次创建英雄机时返回这个实例即可实现单一实例对象。**

**关键方法：**

**HeroAircraft:构造函数，但设定为私有，防止外部调用产生多个实例**

**Getinstance:每次调用时，返回instance对象看，并当其未初始化时将其初始化。**

**当instance未初始化时，调用一个加了线程锁的块：为避免多线程竞争，故需判断instance是否初始化，当未初始化时，调用自身的构造函数创建一个新示例。**

**并在最后返回已经初始化的instance对象。**

### 2.2.2工厂模式

1. 应用场景分析

**当创建不同种类敌机对象，以及创建不同道具对象时可以使用该模式。**

**目前代码中存在创建敌机和道具操作封装不够严密，需要主程序中对其各参数进行初始化，且并无创建对象的同一逻辑和接口。**

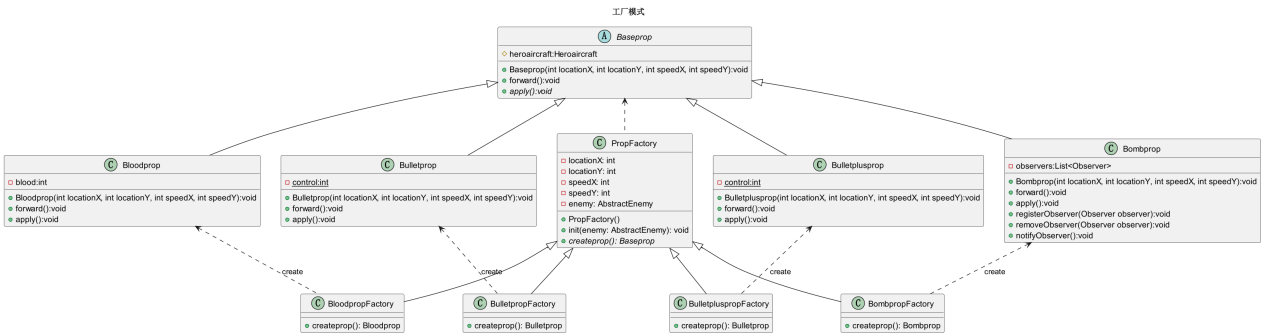
**优势：**

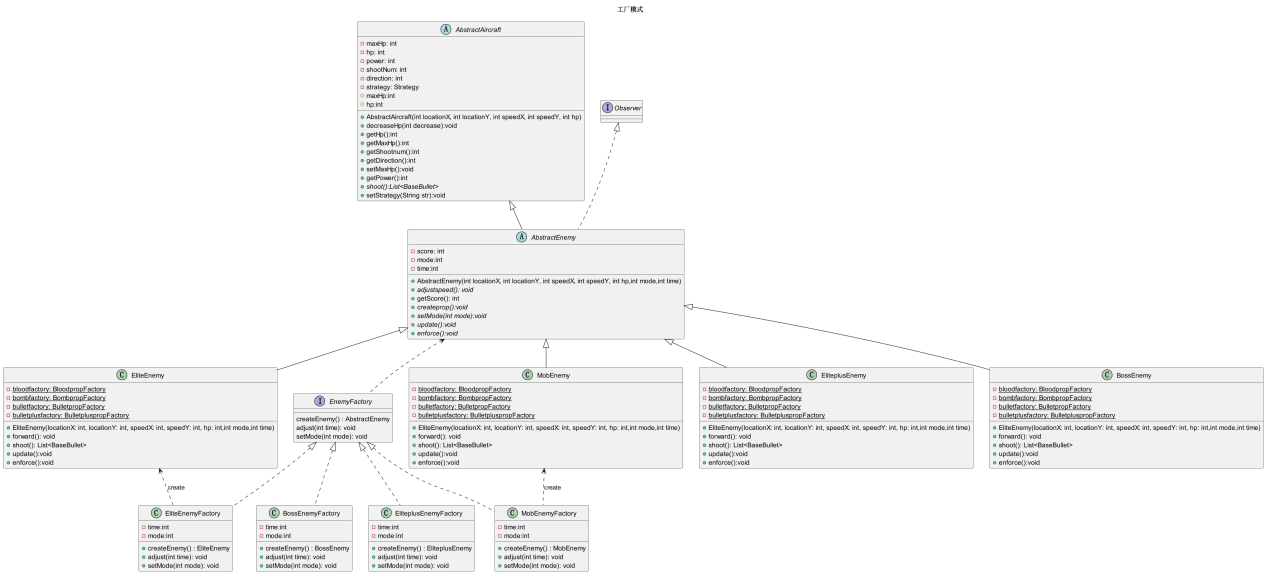
**1.使得对象的创建过程与使用过程解耦，使得客户端代码与具体的对象创建实现相分离。**

**2.生成统一的对象创建接口，并支持扩展和变化，对后续改进功能有帮助。**

**3.封装复杂的创建逻辑，使得主程序对其调用更加简洁易于理解**

1. 设计模式结构图





### 2.2.3策略模式

1. 应用场景分析

**应用场景：飞机发射不同类型的子弹，以及切换发射模式时。**

**目前问题：**

**飞机子弹的发射代码存在冗余，且对发射子弹的方式封装的不够好，增添新发射模式时不够灵活。**

**优势：**

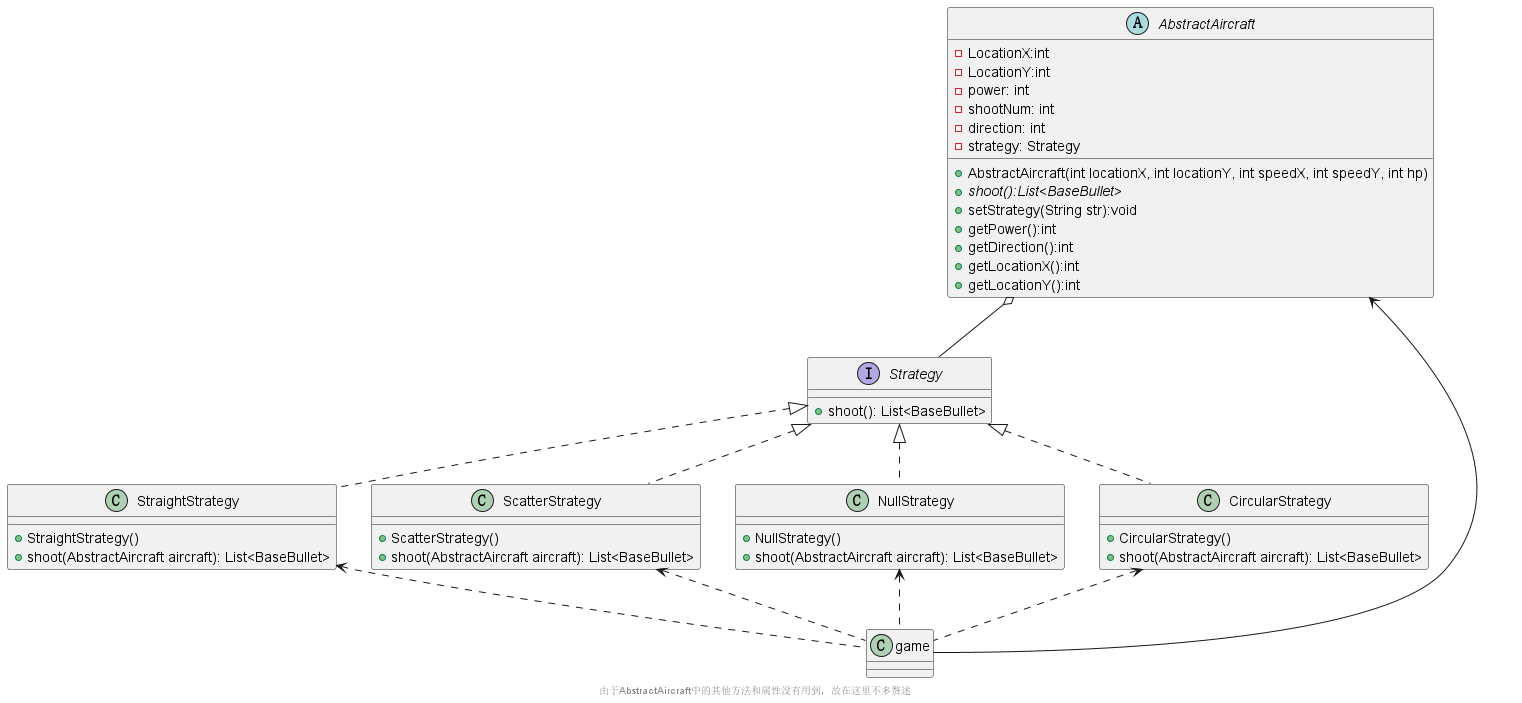
**1.提高了代码的重用率，对于客户端避免了重写代码。**

**2.封装性提高，避免重复使用if-else语句，使得代码结构更清晰，可读性更高。**

**3.增加可扩展性，当实现新的发射逻辑时，创建新的策略类即可。**

1. 设计模式结构图

*结合飞机大战实例，绘制该场景下具体的解决方案（UML类图）。描述你设计的UML类图结构中每个角色的作用，并指出它的关键属性和方法。*



**接口：Strategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)，即要求实现它的类必须实现发射子弹的函数**

**作用：作为每个策略的接口，对发射子弹的策略参数及函数名称进行规范**

**类：NullStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现不射子弹的逻辑，返回空的子弹列表。**

**类：StraightStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现直射子弹的逻辑，根据传入飞机的参数，对子弹列表进行初始化并返回**

**类：ScatterStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现散射子弹的逻辑，根据传入飞机的参数，对子弹列表进行初始化并返回**

**类：CircularStrategy**

**方法：List<BaseBullet>shoot(AbstractAircraft aircraft**

**)**

**作用：实现环射子弹的逻辑，根据传入飞机的参数，对子弹列表进行初始化并返回**

**抽象类：AbstractAircraft**

**属性：**

**LocationX:X坐标**

**LocationY:Y坐标**

**Power:子弹伤害值**

**shootNum:子弹数量**

**Direction：相对发射位置**

**Strategy:发射策略**

**方法：**

**AbstractAircraft(int locationX, int locationY, int speedX, int speedY, int hp)构造函数**

**一系列对私有属性的getter和setter方法**

**+ setStrategy(String str):void**

**根据传入的字符串，对策略进行选择。**

**+getPower():int**

**+getDirection():int**

**+getLocationX():int**

**+getLocationY():int**

**发射子弹：**

**shoot():List<BaseBullet>**

**类：game**

**方法和属性不多做说明。**

**作用：作为主方法，实现对各个飞机策略的初始化，以及英雄机各个策略模式之间的切换。**

**game中通过各个飞机对各个具体策略进行使用，与各种具体策略为关联关系。**

### 2.2.4数据访问对象模式

1. 应用场景分析

*描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。*

**应用场景：在对游戏的排行榜进行存储以及读取**

**优势：**

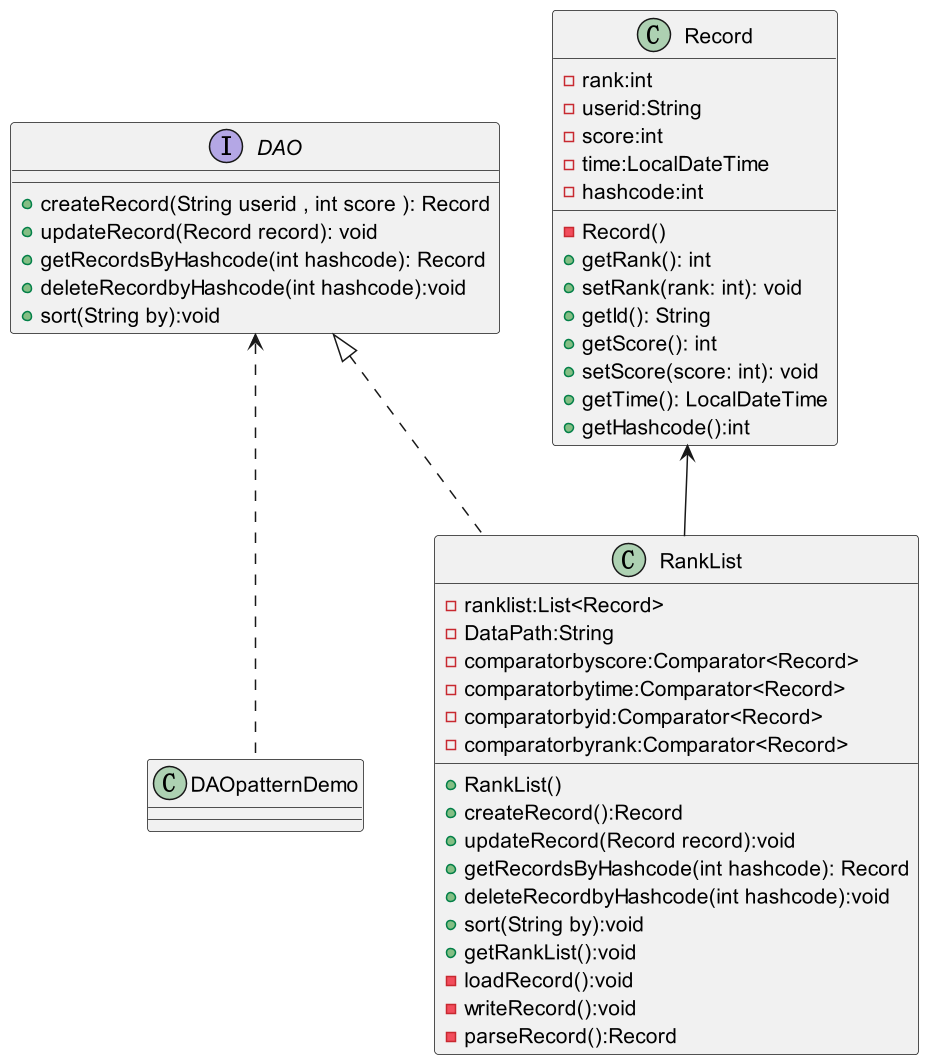
**1. 分离了数据访问逻辑和业务逻辑，将数据访问逻辑封装在DAO对象中，使得业务逻辑可以独立于具体的数据访问细节。这样可以降低代码的耦合性，提高代码的可维护性和可测试性。**

**2. 提供了统一的数据访问接口，通过这些接口，可以统一管理数据访问操作，这些一致的数据访问方式降低了代码的复杂性。**

**3. 隐藏了底层数据操作细节，DAO模式将底层的数据访问细节封装在DAO对象中，对上层代码隐藏了数据库的具体实现细节，有助于提高代码的安全性。**

1. 设计模式结构图

*结合飞机大战实例，绘制该场景下具体的解决方案（UML类图）。描述你设计的UML类图结构中每个角色的作用，并指出它的关键属性和方法。。*



**接口：DAO**

**声明了以下5个方法**

**Record createRecord(String userid, int score);**

**List<Record> getRecordsByint Hashcode(int Hashcode);**

**void updateRecord(Record record);**

**void deleteRecordByHashcode(int Hashcode);**

**void sort(String by);**

**作用：为对数据访问和操作提供一组规范的接口和方式。**

**注意：查找和删除都是根据hashcode进行，而hashcode具体由userid和time计算出。**

**类：Record**

**共有五个属性：rank,userid,score,time,hashcode分别表示：排名，用户名称，分数，记录时间，哈希值。**

**方法：**

**构造函数:public Record(int rank,String userid,int score,LocalDateTime time)**

**作用：对自身进行初始化。**

**一系列setter和getter方法：**

**getRank()**

**setRank()**

**getId()**

**getScore()**

**setScore()**

**getTime()**

**getHashcode()**

**对上述私有变量分别进行赋值和取值**

**作用：作为一项记录的结构体，对记录的数据进行封装。**

**类:RankList**

**属性：List<Record> ranklist**

**作用：用于暂时存储数据库中全部记录便于增删查改**

**属性：Comparator<Record> comparatorbyscore**

**作用：传入sort函数中的比较器，用于对各项记录进行排序操作。**

**属性： String DataPath**

**作用：记录数据库的存储位置**

**方法：**

**构造函数：**

**RankList()：给数据库路径初始化**

**void loadRecord()：读取csv格式数据文件，并将其解析后存入ranklist变量中**

**void writeRecord()：打开csv格式数据文件，并将当前的ranklist以csv格式写入文件**

**Record parseRecord(String str)：将从csv中读取出的每行字符串格式的记录解析成Record类型的，并返回。**

**Record createRecord(String userid,int score)：调用Record的构造函数，返回Record**

**Record getRecordsByHashcode(int Hashcode) ：首先调用loadRecord读取数据文件，存入ranklist，遍历ranklist，当Hashcode相同时则将其添加到暂存数组，并将其全部返回。**

**void updateRecord(Record record)：首先调用loadRecord读取数据文件，存入ranklist；**

**接着将遍历ranklist中每一项record1，当record1.score>record.score时，将record.rank++，即排名+1;当record1.score<record.score时，将record1的rank++。**

**最终使用writeRecord()将其存回文件。**

**void getRankList(Record record)：返回该ranklist。**

**void deleteRecordbyHashcode(int Hashcode)：先调用loadRecord()将数据读入ranklist，再调用getRecordsByHashcode得到给定Hashcode的记录。接着遍历ranklist将该记录删除，并调整其他记录的名次，最后调用writeRecord()将其写入文件。**

**void printRecord() ：调用loadRecord()将数据读入ranklist，再按一定格式将其打印至控制台。**

**void sort(String by)： 按照给定的“by”方法选择不同的比较器，使用collection内置比较函数对ranklist进行排序。**

**类：DAOpatternDemo**

**作用：对DAO的一系列操作进行测试。**

### 2.2.5观察者模式

1. 应用场景分析

*描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。*

**应用场景：炸弹道具对各个飞机/子弹造成伤害**

**实际问题：不同的对象对道具的反应不同，然而行为模式一致，当新增关联对象时或者新增道具时增添代码不易修改。**

**优势：**

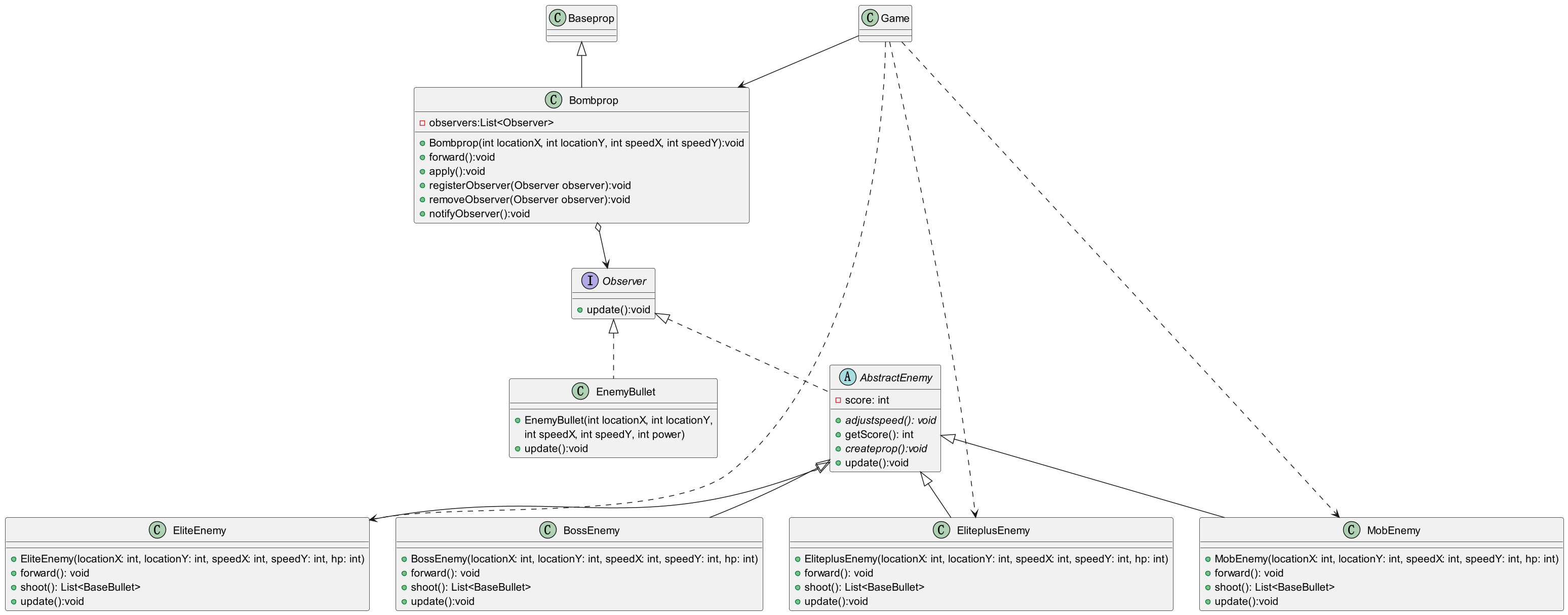
**1.观察者模式可以将观察者和被观察者之间的耦合度降低，可以方便地增加、删除或修改观察者**

**2.可以轻松添加新的观察者，而无需修改现有代码，使得系统更容易扩展**

**3.被观察者和观察者可以被单独复用，提高了代码的可重用性和灵活性**

1. 设计模式结构图

*结合飞机大战实例，绘制该场景下具体的解决方案（UML类图）。描述你设计的UML类图结构中每个角色的作用，并指出它的关键属性和方法。*



**Observer接口：声明了update方法，提供每一个Observer对象的行为接口规范**

**Bombprop：作为话题发布者，保存了一个Observer列表，用于保存所登记的所有观察者对象。**

**其中观察者分别为四种敌机和敌机子弹。**

**registerObserver():将给定的观察者对象注册到当前发布者对象的观察者列表中**

**removeObserver():将给定的观察者对象从当前发布者对象的观察者列表中移除**

**notifyObserver():当发布者发布消息时，向所有观察者对象发布信息**

**Apply方法实现道具发挥作用，即令所有Observer对象进行update()操作。**

**对于各个敌机：**

**Update()：实现对发布者的响应，具体表现为非超级精英敌机坠毁，超级精英敌机掉血（我设定为最大血量的一半），boss敌机无反应，敌机子弹全部消失**

### 2.2.6模板模式

1. 应用场景分析

*请简单描述你对三种游戏难度是如何设计的，影响游戏难度的因素有哪些。描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。*

**应用场景：游戏难度选择**

**实际问题：在Game类中实现不同难度时逻辑复杂，当新增难度时需要改变逻辑，不符合开闭原则，且三种模式存在共同逻辑，可以使用模板模式实现代码复用。**

**1.模板模式通过将算法的框架结构放在抽象类中，可以使多个子类共享相同的算法结构和代码逻辑。这样可以避免重复编写相似的代码，提高代码复用性。**

**2.通过将具体步骤的实现延迟到子类中，模板模式使得算法的具体实现可以独立进行扩展和修改。可以通过添加新的子类来扩展算法的功能，而无需修改抽象类和其他已有的子类。**

**3.模板模式中的抽象类可以定义钩子方法，用于在算法的不同步骤中提供额外的扩展点。**

**4.模板模式使得算法的框架结构和具体步骤分离，降低了代码的耦合性，使代码更易于理解、维护和修改。**

**模式设计：**

**1.简单模式：不改变游戏难度，维持初始难度**

**2.普通模式：随时间增长，增强所有敌机：速度和血量，并增加最大敌机数量、精英敌机产生概率和英雄、敌机射击频率**

**具体数据如下：**

**敌机速度增长：min(time/4000+speedY,8)**

**敌机血量增长：min(time/4000+this.hp,100)，对于三种敌机，血量最大值依次为100，300，400.**

**敌机数量增长：min(enemyMaxNumber+time/8000,7)**

**精英敌机概率增长：min(eliteprob+(double)time/200000.0,0.7)**

**射击频率增长（周期减小）：max(cycleDuration-time/100,400)**

**Boss敌机产生的得分阈值为1500**

**3.困难模式：在普通模式的基础上，随击杀次数增加boss血量。**

**Boss机血量增长：min(bosscount\*500+this.hp,4000)  
另外修改参数：**

**敌机数量增长：min(enemyMaxNumber+time/8000,9)**

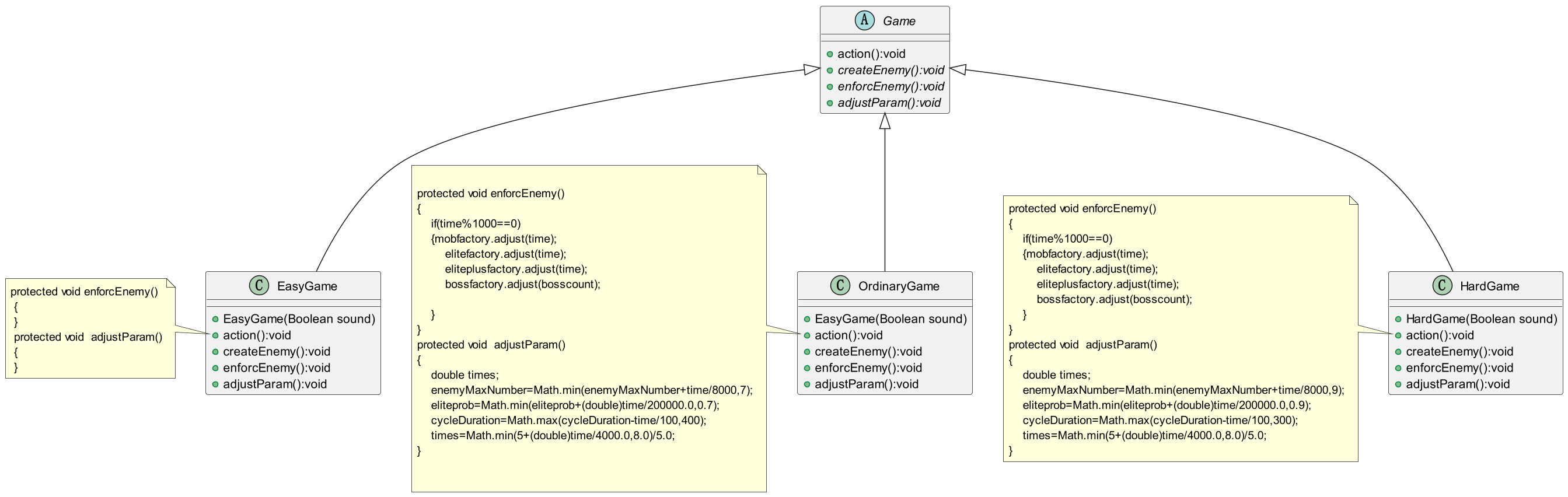
**精英敌机概率增长：min(eliteprob+(double)time/200000.0,0.9)**

**射击频率增长（周期减小）：max(cycleDuration-time/100,300)**

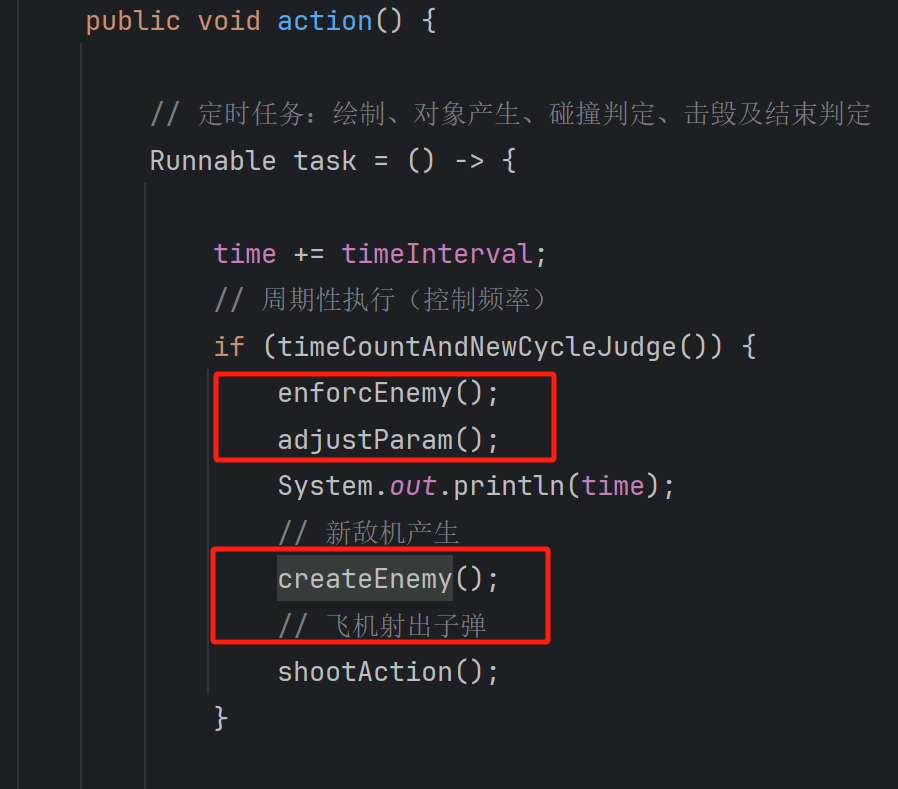
**Boss敌机产生的得分阈值为2000**

1. 设计模式结构图

*结合飞机大战实例，绘制该场景下具体的解决方案（UML类图）。描述你设计的UML类图结构中每个角色的作用，并指出它的关键属性和方法。*



Game抽象类：作为各个游戏模式的超类，提供了方法模板action，其中共包含createEnemy(),enforcEnemy()和adjustParam()三个抽象方法待实现。



EasyGame中enforcEnemy()和adjustParam()不做任何事,createEnemy()中只有产生非boss敌机的逻辑。

OrdinaryGame中createEnemy()在EasyGame的基础上增加了产生boss敌机的逻辑，enforcEnemy()和adjustParam()具体代码如上图所示，前者负责使得敌机工厂参数随时间改变，进而改变敌机的初始属性；后者负责Game中最大敌机数量、精英敌机概率，射击周期等参数随时间发生改变。

HardGame中createEnemy()实现逻辑和OrdinaryGame一样，enforcEnemy()和adjustParam()和OrdinaryGame实现功能基本一致，区别在于参数不同，最大敌机数量、精英敌机概率，射击周期等参数的最大值均更大。

# 收获和反思

**收获：掌握了各种设计模式，熟悉了java语言尤其是多线程编程和swing GUI设计。**

**同时对于代码架构和面对对象编程思想有了更深入的理解。**

**建议：希望实验报告能少写一点。**