哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

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 计科 5 班

学生学号：220110504

学生姓名：李乐怡

评阅教师：

报告成绩：

实验与创新实践教育中心制

2024 年 5 月

1 实验环境

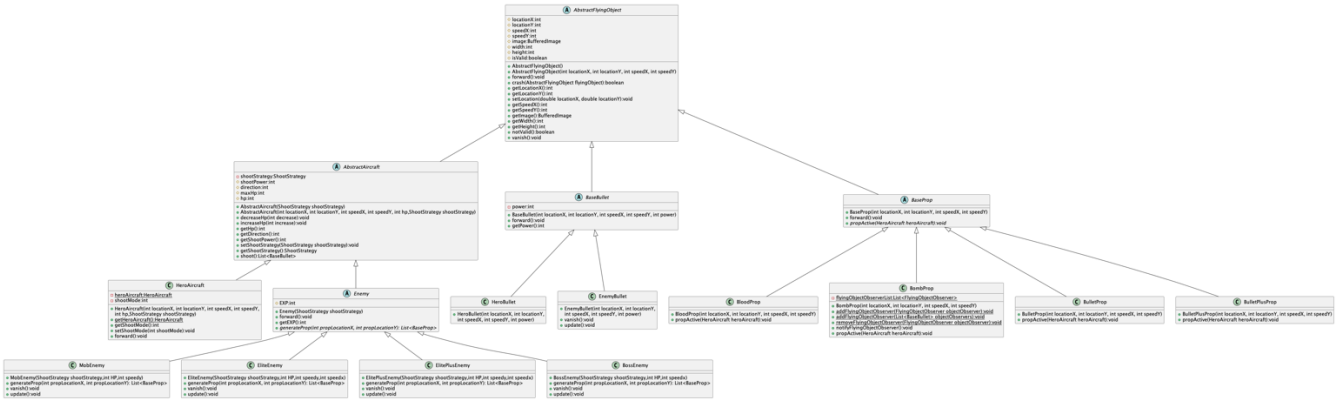
1. macOS Sonoma 14.4.1
2. IntelliJ IDEA 2024.1
3. Openjdk-22

2 实验过程

以下 UML 结构图请更新为最终提交版本。

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

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。

（1）场景分析：

飞机大战游戏中，只创建一种英雄机，只有一种英雄机，且每局游戏只有一架英雄机，由玩家通过鼠标控制其移动。

（2）设计中出现的问题：

①目前代码在 Game 类用 new 方法创建英雄机，创建和使用在同一类中，违反单一职责，不符合面向对象设计原则。

②目前代码可以在外部程序随意用 new 方法创建一个实例，不能保证英雄机的唯一性。

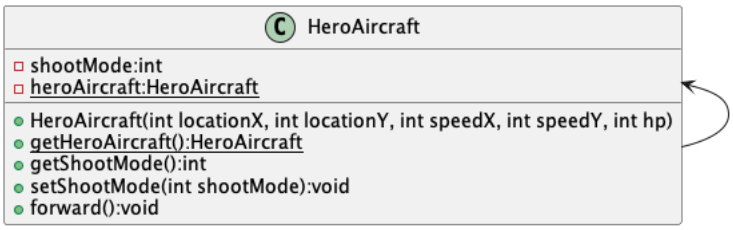
（3）单例模式优势：

①保证一个类仅有一个实例，保证当前游戏只有一个英雄机。

②提供一个访问该唯一实例的全局访问点，使其易于访问。

2. 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。



①属性 heroAircraft 是私有的，也是静态的，用来记录 HeroAircraft 类的唯一实例。

②构造函数 HeroAircraft(int locationX, int locationY, int speedX, int speedY, int hp)是私有的，使外界无法利用 new 创建此类实例，保证其只在此类中创造一个实例。

③方法 getHeroAircraft()是 public 的，也是静态的，允许外界访问它的唯一实例。

2.2.2 工厂模式

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。

场景一：敌机

（1）场景分析：

飞机大战游戏中，需创建 4 种类型的敌机：普通敌机、精英敌机、超级精英敌机、Boss 敌机，并进行相应的操作。

（2）设计中出现的问题：

①目前代码在 Game 类用 new 方法创建敌机，创建和使用在同一类中，违反单一职责，不符合面向对象设计原则。

②目前代码若要增加新型的敌机，需在 Game 类进行修改，没有对修改封闭，违反开闭原则，不符合面向对象设计原则。

③目前代码是针对实现编程，不是针对抽象编程，违反依赖倒转原则，不符合面向对象设计原则。

（3）工厂模式优势：

①创建敌机过程封装在在工厂子类，由子类决定实例化对象的敌机类型，将客户程序中关于超类的代码和子类对象创建的代码解耦，满足单一职责和开闭原则。

②工厂模式的代码是针对抽象编程，符合依赖倒转原则。

场景二：道具

（1）场景分析：

飞机大战游戏中，需创建 4 种类型的道具：火力道具、超级火力道具、炸弹道具、加血道具，并进行相应的操作。

（2）设计中出现的问题：

①目前代码在 Game 类用 new 方法创建道具，创建和使用在同一类中，违反单一职责，不符合面向对象设计原则。

②目前代码若要增加新型的道具，需在 Game 类进行修改，没有对修改封闭，违反开闭原则，不符合面向对象设计原则。

③目前代码是针对实现编程，不是针对抽象编程，违反依赖倒转原则，不符合面向对象设计原则。

（3）工厂模式优势：

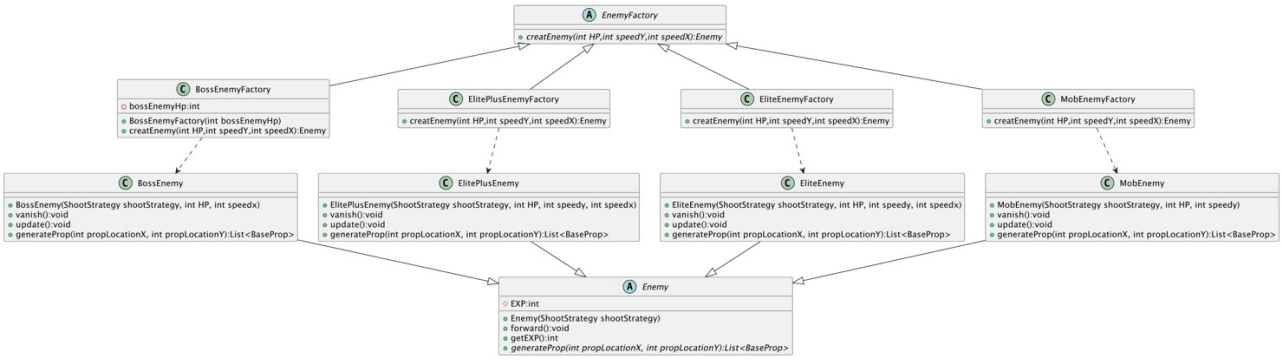
①创建道具过程封装在在工厂子类，由子类决定实例化对象的道具类型，将客户程序中关于超类的代码和子类对象创建的代码解耦，满足单一职责和开闭原则。

②工厂模式的代码是针对抽象编程，符合依赖倒转原则。

2. 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。

工厂一：敌机工厂



（1） **产品系**：创建 Enemy 抽象类和**继承**该抽象类的四个敌机实体类 MobEnemy、EliteEnemy、ElitePlusEnemy 和 BossEnemy。

① Enemy：

-方法：方法 generateProp(int propLocationX, int propLocationY)是抽象方法。

-作用：充当产品角色——敌机，声明敌机应有的功能。

② MobEnemy：

-方法：构造方法 MobEnemy(ShootStrategy shootStrategy, int HP, int speedy)初始化了普通敌机的各项数据，方法 generateProp(int propLocationX, int

propLocationY)由普通敌机子类进行重写。

-作用：充当具体产品角色——普通敌机，有普通敌机的具体功能。

③ EliteEnemy：

-方法：构造方法 EliteEnemy(ShootStrategy shootStrategy, int HP, int speedy,int speedx)初始化了精英敌机的各项数据，方法 generateProp(int propLocationX, int propLocationY)由精英敌机子类进行重写。

-作用：充当具体产品角色——精英敌机，有精英敌机的具体功能。

④ ElitePlusEnemy：

-方法：构造方法 ElitePlusEnemy(ShootStrategy shootStrategy, int HP,int speedy, int speedx)初始化了超级精英敌机的各项数据，方法 generateProp(int propLocationX, int propLocationY)由超级精英敌机子类进行重写。

-作用：充当具体产品角色——超级精英敌机，有超级精英敌机的具体功能。

⑤ BossEnemy：

-方法：构造方法 BossEnemy(ShootStrategy shootStrategy, int HP, int speedx)初始化了 Boss 敌机的各项数据，方法 generateProp(int propLocationX, int propLocationY)由 Boss 敌机子类进行重写。

-作用：充当具体产品角色——Boss 敌机，有 Boss 敌机的具体功能。

(2) **工厂系**：创建工厂抽象类 EnemyFactory 和**继承**该抽象类的四个具体敌机工厂实体类 MobEnemyFactory 、 EliteEnemyFactory 、 ElitePlusEnemyFactory 和 BossEnemyFactory。

① EnemyFactory：

-方法：方法 createEnemy(int HP,int speedY,int speedX)是抽象方法，返回对象类型即为产品角色 Enemy。

-作用：充当创建者角色——敌机工厂。

② MobEnemyFactory：

-方法：方法 createEnemy(int HP,int speedY,int speedX)由普通敌机工厂子类进行重写，使其返回普通敌机产品。

-作用：充当具体创建者角色——普通敌机工厂。

③ EliteEnemyFactory：

-方法：方法 createEnemy(int HP,int speedY,int speedX)由精英敌机工厂子类进行重写，使其返回精英敌机产品。

-作用：充当具体创建者角色——精英敌机工厂。

④ ElitePlusEnemyFactory：

-方法：方法 createEnemy(int HP,int speedY,int speedX)由超级精英敌机工厂子类进行重写，使其返回超级精英敌机产品。

-作用：充当具体创建者角色——超级精英敌机工厂。

⑤ BossEnemyFactory：

-属性：私有属性 bossEnemyHp，传入此次生成 boss 敌机的血量，与其他固定血量的敌机不同

-方法：方法 createEnemy(int HP,int speedY,int speedX)由 Boss 敌机工厂子类进行重写，使其返回 Boss 敌机产品。

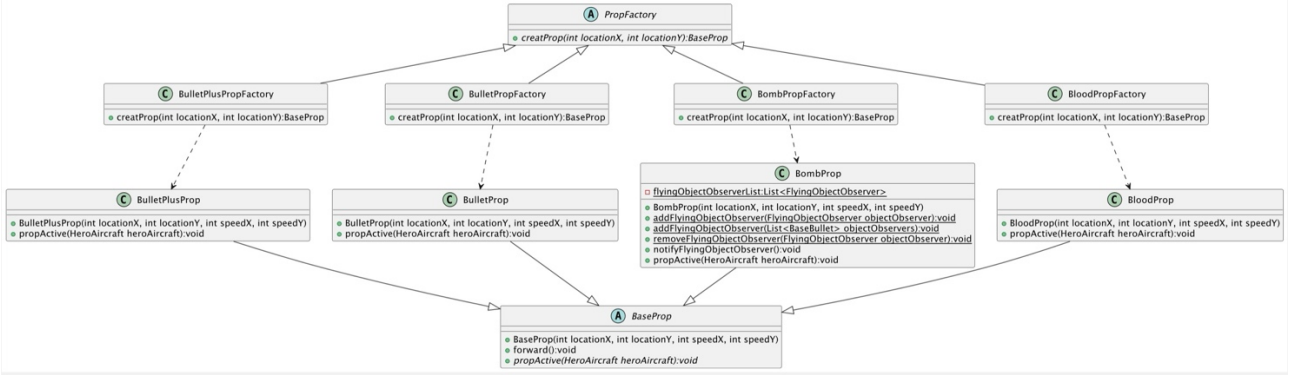
-作用：充当具体创建者角色——Boss 敌机工厂。

(3) 类 MobEnemyFactory **依赖**类 MobEnemy，

类 EliteEnemyFactory **依赖**类 EliteEnemy

类 ElitePlusEnemyFactory **依赖**类 ElitePlusEnemy
类 BossEnemyFactory **依赖**类 BossEnemy

工厂二：道具工厂



(1) **产品系**：创建 BaseProp 抽象类和**继承**该抽象类的四个道具实体类 BloodProp、BombProp、BulletProp 和 BulletPlusProp。

① BaseProp：

-方法：方法 propActive(HeroAircraft heroAircraft)是抽象方法。

-作用：充当产品角色——道具，声明道具应有的功能。

② BloodProp：

-方法：构造方法 BloodProp(int locationX, int locationY, int speedX, int speedY)初始化了加血道具的各项数据，方法 propActive(HeroAircraft heroAircraft)由加血道具子类进行重写。

-作用：充当具体产品角色——加血道具，有加血道具的具体功能。

③ BombProp：

-方法：构造方法 BombProp(int locationX, int locationY, int speedX, int speedY)初始化了炸弹道具的各项数据，方法 propActive(HeroAircraft heroAircraft)由炸弹道具子类进行重写。

-作用：充当具体产品角色——炸弹道具，有炸弹道具的具体功能。

④ BulletProp：

-方法：构造方法 BulletProp(int locationX, int locationY, int speedX, int speedY)初始化了火力道具的各项数据，方法 propActive(HeroAircraft heroAircraft)由火力道具子类进行重写。

-作用：充当具体产品角色——火力道具，有火力道具的具体功能。

⑤ BulletPlusProp：

-方法：构造方法 BulletPlusProp(int locationX, int locationY, int speedX, int speedY)初始化了超级火力道具的各项数据，方法 propActive(HeroAircraft heroAircraft)由超级火力道具子类进行重写。

-作用：充当具体产品角色——超级火力道具，有超级火力道具的具体功能

(2) **工厂系**：创建工厂抽象类 PropFactory 和**继承**该抽象类的四个具体工厂实体类 BloodPropFactory、BombPropFactory、BulletPropFactory 和 BulletPlusPropFactory。

① PropFactory：

-方法：方法 createProp(int locationX, int locationY)是抽象方法，返回对象类型

即为产品角色 BaseProp。
-作用：充当创建者角色——道具工厂。
② BloodPropFactory：
-方法：方法 createProp(int locationX, int locationY)由加血道具工厂子类进行重写，使其返回加血道具产品。
-作用：充当具体创建者角色——加血道具工厂。
③ BombPropFactory：
-方法：方法 createProp(int locationX, int locationY)由炸弹道具工厂子类进行重写，使其返回炸弹道具产品。
-作用：充当具体创建者角色——炸弹道具工厂。
④ BulletPropFactory：
-方法：方法 createProp(int locationX, int locationY)由火力道具工厂子类进行重写，使其返回火力道具产品。
-作用：充当具体创建者角色——火力道具工厂。
⑤ BulletPlusPropFactory：
-方法：方法 createProp(int locationX, int locationY)由超级火力道具工厂子类进行重写，使其返回超级火力道具产品。
-作用：充当具体创建者角色——超级火力道具工厂。

（3） 类 BloodPropFactory **依赖**类 BloodProp，
类 BombPropFactory **依赖**类 BombProp，
类 BulletPropFactory **依赖**类 BulletProp，
类 BulletPlusPropFactory **依赖**类 BulletPlusProp

2.2.3 策略模式

1. 应用场景分析

（1）场景分析：

飞机大战游戏中，不同类型飞机的发射弹道各不相同，且火力道具生效时英雄机切换相应的弹道。

（2）设计中出现的问题：

①目前代码在抽象飞机类定义了抽象方法 shoot，在不同飞机产品类用具体的 shoot 方法实现不同飞机的不同子弹发射，这些 shoot 方法算法相似。

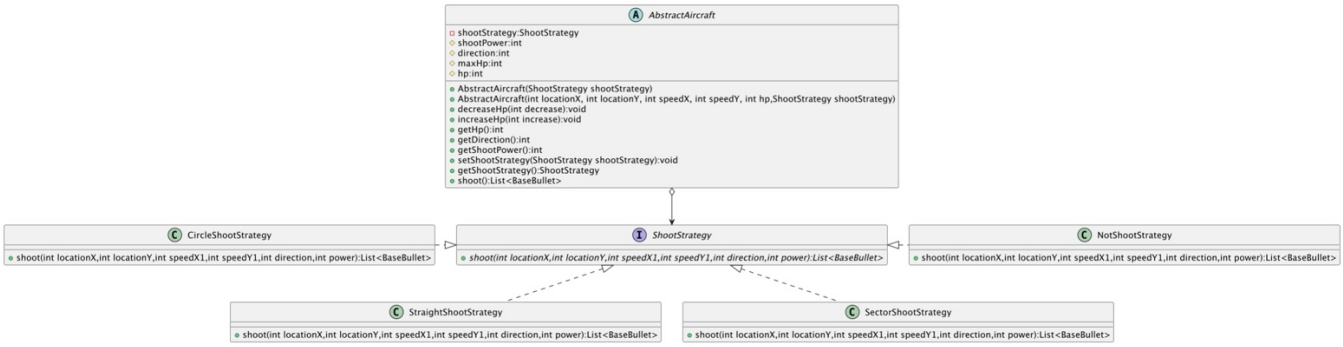
②目前代码在实现英雄机获取火力道具后弹道的改变时，需要从敌机类将散射和环射的代码复制到英雄机类，重复代码多、代码难维护、易引入 bug，会违反开闭原则。

（3）策略模式优势：

①面向更高的逻辑抽象层，可减少耦合、封装变化，尽量避免错误蔓延。
②算法策略可以自由切换。
③避免使用多重条件判断。
④方便拓展和增加新的算法，满足开闭原则。

2. 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。



（1）抽象策略接口：

ShootStrategy 接口：

-方法：抽象方法 shoot(int locationX,int locationY,int speedX1,int speedY1,int direction, int power)，返回子弹列表 List<BaseBullet>

-作用：充当抽象策略角色，给出具体策略类所需的接口。

（2）4 个具体策略类：均继承于 Strategy

① NotShootStrategy 类：

-方法：具体方法 shoot。

-作用：充当具体策略角色——实现不射击策略，并将其封装起来。

② StraightShootStrategy 类：

-方法：具体方法 shoot。

-作用：充当具体策略角色——实现直线射击策略，并将其封装起来。

③ SectorShootStrategy 类：

-方法：具体方法 shoot。

-作用：充当具体策略角色——实现扇形射击策略，并将其封装起来。

④ CircleShootStrategy 类：

-方法：具体方法 shoot。

-作用：充当具体策略角色——实现环形射击策略，并将其封装起来。

（3）上下文类：

AbstractAircraft 类：

-属性：私有属性 ShootStrategy 类的 shootStrategy

-方法：

a. 构造方法，传入一个具体的策略对象，初始化 shootStrategy

b. 方法 setShootStrategy(ShootStrategy shootStrategy)，设置一个具体的策略对象。

c. 方法 getShootStrategy()，返回一个策略对象

d. 方法 shoot()，调用当前具体策略 shootStrategy 的 shoot 方法，返回子弹列表 List<BaseBullet>。

-作用：充当上下文角色，屏蔽高层模块对策略算法的直接访问，封装可能存在的变化。

4 个具体策略类均继承于抽象策略类 ShootStrategy

ShootStrategy 类与 AbstractAircraft 类是聚合关系，前者为可独立部分类，后者为整体类

2.2.4 数据访问对象模式

1. 应用场景分析

（1）场景分析：

飞机大战游戏中，每局游戏记录英雄机得分，游戏结束后，显示该难度的玩家得分排行榜，按成绩降序排列。

（2）设计中出现的问题：

①目前代码将数据存储在文件中，想进行玩家得分等数据的增删改查等操作时，需要在 Game 类中进行更改，并且对存储数据的文件进行直接操作。较为繁琐，并且易引入 bug。

②若需要更改数据源，目前代码需要更改 Game 类，违反开闭原则。

（3）数据访问对象模式优势：

①面向更高的逻辑抽象层，可减少耦合、封装变化，尽量避免错误蔓延。

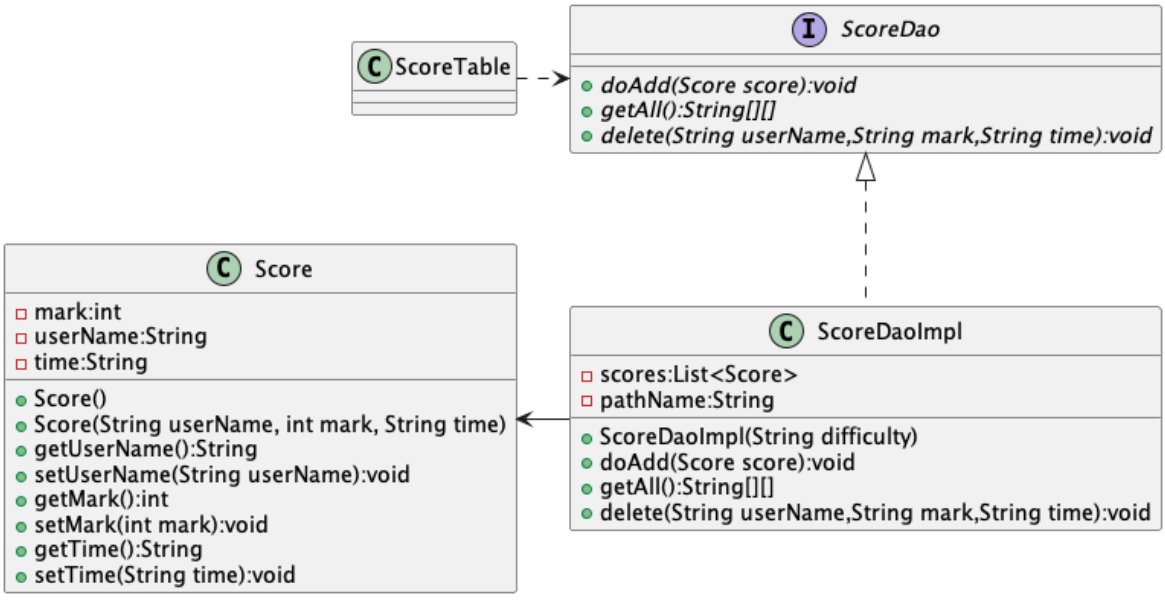
②将低级的数据访问操作从高级的业务服务中分离出来。

③进行低级的数据访问操作时，Dao 模式封装了数据库的操作，提供了简单而统一的操作接口，不需要直接操作数据库，方便对前端存储的管理。

④若要更换数据库，只会更改 Dao 层而不会影响到服务层或者实体对象，减少了服务层与数据存储设备层之间的耦合度。

2. 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。。



（1）作为模型对象/数值对象的实体类：

Score 类：

-属性：私有属性 int 型 mark、String 类型 userName 和 time

-方法：构造方法，各属性对应的 get、set 方法

-作用：包含了 get/set 方法来存储或读取通过使用 Dao 类检索到的数据。

（2）数据访问对象接口：

ScoreDao 接口：

-方法：抽象方法 doAdd，实现数据的添加；

抽象方法 getAll，获取所有数据并存入 String[][]中；

抽象方法 delete，将对应的 Score 数据删除。

-作用：定义了在一个模型对象上要执行的添加数据、获取排行榜、删除一行数据的标准操作。

（3）数据访问对象实现类：

ScoreDaoImpl 类：

-属性：私有属性 List<Score>类的 scores

私有属性 String 类型的 pathName

-方法：

a. 构造方法 ScoreDaoImpl(String difficulty)，初始化 scores，pathName 存所选难度所对应的存储数据的文件名

b. 方法 doAdd(Score score)，添加一组数据到对应的文件内。

c. 方法 getAll()，从对应文件中获取全部数据，存储到 scores 中，将 scores 中的数据排序进行降序输出，并存入一个 String[][]类型的 scoreTable 中，scoreTable 为返回值。

d. 方法 delete(String userName,String mark,String time)，从对应文件中删除某特定一行数据

-作用：该类实现了上述的接口，负责从数据源获取数据、添加数据以及删除数据。

（4）使用数据访问对象模式的类

ScoreTable 类：

-作用：调用 ScoreDaoImpl 类的方法进行数据的访问。

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。

（1）场景分析：

飞机大战游戏中，敌机坠毁时可能掉落炸弹道具，其功能是清除界面上的所有的普通、精英敌机和敌机子弹，减少超级精英敌机血量，Boss 敌机不受影响。英雄机可

获得坠毁的敌机分数。

（2）设计中出现的问题：

①目前代码在炸弹道具类实现了炸弹爆炸时的作用生效，有普通敌机、精英敌机、超级精英敌机、boss 敌机、敌机子弹需要对炸弹做出响应，对多个不同的待响应物进行处理，违反了单一职责原则。

②如果需要增加一个对炸弹有响应的新角色，需要在炸弹道具类中增添代码，违反开闭原则。

③如果需要增加一个减速道具，需要重新将类似的代码复制过去，针对现实编程，违反依赖倒转原则。

（3）观察者模式优势：

①可以实现表示层和数据逻辑层的分离。

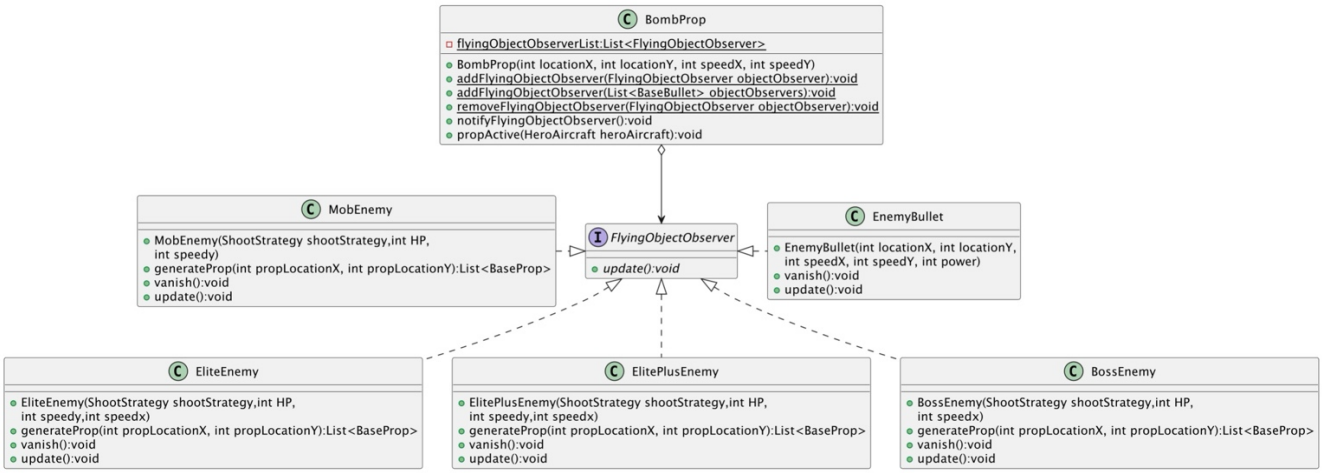
②支持广播通信，简化了一对多系统设计的难度。

③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较为方便。

④在观察目标和观察者之间建立一个抽象的耦合。

2. 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。



（1）观察者接口：

FlyingObjectObserver 接口：

-方法：抽象方法 update()

-作用：充当观察者角色，为所有具体的观察者定义一个接口，在得到观察目标的通知时更新自己。

（2）5 个具体观察者：均实现了 FlyingObjectObserver

① MobEnemy 类：

-方法：具体方法 update()，实现敌机坠毁。

-作用：充当具体观察者角色——实现做出坠毁响应的更新接口。

② EliteEnemy 类：

-方法：具体方法 update()，实现敌机坠毁。

-作用：充当具体观察者角色——实现做出坠毁响应的更新接口。

③ ElitePlusEnemy 类：

-方法：具体方法 update()，实现超级精英敌机血量减少。

-作用：充当具体策略角色——实现超级精英敌机血量减少响应的更新接口

④ BossEnemy 类：

-方法：具体方法 update()，不做任何改变。

-作用：充当具体观察者角色——实现不做出改变响应的更新接口。

④ EnemyBullet 类：

-方法：具体方法 update()，实现敌机坠毁。

-作用：充当具体观察者角色——实现做出子弹销毁响应的更新接口。

（3）观察目标类：

BombProp 类：

-属性：静态属性 List<FlyingObjectObserver>类的 flyingObjectObserverList

-方法：

a.静态方法 addFlyingObjectObserver(FlyingObjectObserver objectObserver)，向观察者集合中添加一个观察者

b. 静态方法 addFlyingObjectObserver(List<BaseBullet> objectObservers)，向观察者集合中添加一个 List<BaseBullet>类型的观察者列表

c. 静态方法 removeFlyingObjectObserver(FlyingObjectObserver objectObserver)，向观察者集合中删除一个观察者

d. 方法 notifyFlyingObjectObserver()，通知所有的观察者们

e. 方法 propActive(HeroAircraft heroAircraft)，炸弹道具生效，调用方法 notifyFlyingObjectObserver()，通知所有观察者们

-作用：作为观察目标，把所有对观察者对象的引用保存在一个集合中，提供了可以增加和删除观察者角色的方法。在其内部状态改变时，给所有观察者集合内的观察者发出通知。

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。

（1）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，具体设计如下：

基础设置	简单	普通	困难
Boss敌机	无	有，每次召唤不改变Boss敌机的血量	有，每次召唤增加Boss敌机的血量：60
初始屏幕中出现的敌机最大数量	5	5	5
敌机纵向速度	6	7	9
敌机横向速度	1	1	1
精英敌机产生概率	0.2	0.3	0.35
超级精英敌机产生概率	0.08	0.1	0.15
Boss敌机首次出现时的血量	/	120	180
产生敌机的周期	700	600	600
英雄机射击的周期	500	600	600
敌机射击的周期	700	600	550
难度设置			
难度是否随时间增加而提升	否	是	是
难度提升时间间隔	/	7000	5000
增加界面中出现的敌机数量的最大值	/	每次+0.1，取整数部分	每次+0.15，取整数部分
增加敌机纵向速度	/	每次+0.1，取整数部分	每次+0.15，取整数部分
增加敌机横向速度	/	每次+0.2，取整数部分	每次+0.3，取整数部分
增加精英敌机产生概率	/	每次*1.03	每次*1.1
增加超级精英敌机产生概率	/	每次*1.03	每次*1.1
缩小产生敌机的周期	/	每次/1.03	每次/1.1
缩小敌机射击的周期	/	每次/1.01	每次/1.04

（2）设计中出现的问题：

①三种游戏难度的逻辑骨架一致，部分方法一致，比如碰撞检测方法crashCheckAction()，若分成三个难度的模块分开写会导致大量代码重复。

②如果需要增加一个新的游戏难度，仅仅只是要改变某个特定的步骤，需要在已有的类中进行修改，违反开闭原则。

（3）模版模式优势：

①在抽象类中定义了一个算法的框架和固定不变的方法步骤。

②允许子类在不修改结构的情况下重写算法的特定步骤。

③提高了代码的复用。

④减少耦合，封装变化。

⑤把不变的行为搬到超类，去除子类中的重复的代码。

2. 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。

（1）抽象类：

Game 类：

-方法：构造方法 Game()

抽象方法 changeDifficulty()规定了子类需要重写的改变难度方法

方法 action()以及其他具体的方法，其中方法 action()定义了一个顶层逻辑骨架，即一个模版方法。

-作用：声明作为算法步骤的方法，以及依次调用它们的实际模版方法，其中可变的“改变难度”部分被声明为抽象类型。

（2）3 个具体类：均继承于 Game

① EasyGame 类：

-方法：构造方法 EasyGame()，初始化简单游戏模式的不同基本信息

具体方法 changeDifficulty ()，实现不改变难度。

-作用：重写改变难度方法，作为一个有不同难度设置的简单游戏模式。

② MediumGame 类：

-方法：构造方法 MediumGame()，初始化普通游戏模式的不同基本信息
具体方法 changeDifficulty ()，实现难度随时间增加而提升。
-作用：重写改变难度方法，作为一个有不同难度设置的普通游戏模式。

③ HardGame 类：
-方法：构造方法 HardGame()，初始化困难游戏模式的不同基本信息
具体方法 changeDifficulty ()，实现难度随时间增加而提升。
-作用：重写改变难度方法，作为一个有不同难度设置的困难游戏模式。

3 收获和反思

请填写本次实验的收获，记录实验过程中出现的值得反思的问题及你的思考。欢迎为本课程实验提出宝贵意见！

1. 收获：
   巩固了导论课上学习到的知识点，把课上一知半解的知识点在这里弄清楚了。
   真正完成了一个项目，体会了做项目的过程。
2. 出现的问题以及思考：
   （1） **问题**：通过多线程来实现音乐播放时，调用到音乐停止的方法后，音乐还会再继续播放 1-2 秒才停止。
   思考：因为在 MusicThread 中每次循环从音频流读取指定的最大数量的数据字节并将其放入缓冲区中，因此需要每隔一小段时间才能判断是否当前的音乐需要暂停。但这个问题无法解决。
   （2） **问题**：火力道具生效时间为 5 秒，当英雄机连续获取了两次超级火力道具，其生效的时间是第一次获取超级火力道具的时间开始的 5 秒。但应该是第二次获取道具重新计时，一直到第二次获取道具开始的 5 秒再结束才更合理
   思考：每次创建新的线程，开始增大火力，并将线程 sleep，英雄机类并不知道当前是哪一线程控制的火力效果。因此在英雄机类中添加 shootMode 属性，每次创建新的线程时，将当前是第几个火力道具线程在生效传入英雄机的 shootMode 属性中，每次 sleep 结束，判断当前英雄机生效的火力道具线程是否与此线程一致，一致则恢复到直射。
   （3） **问题**：排行榜删除一行数据后，要实现删除后显示的排行榜名次仍是连续的，被删除数据之后的数据名次要有正确的改变
   思考：每次删除后，重新调用 scoreDao 的 getAll()方法将新的数据存到 tableData[0]中，然后重新画出排行榜表格。
   （4） **问题**：排行榜删除一行数据后，要实现存储数据源的文件中对应数据的删除，但我无法在读取文件中数据时就进行删除。
   思考：由于可能不同数据的成绩或玩家名或时间相同，因此需要用所有的三个属性来确定一个唯一的数据，因此传入 delete 方法的参数有三个。而在文件中每个属性存一行，因此无法直接在扫描读取文件时进行删除。我选择边

扫描读取文件的数据、边将不用删除的数据存入 List<Score>数据列表，再将数据列表里的数据写入一个新的文件里，最后再删除掉旧的文件，将新的文件重命名为原来的文件名。

3. 意见：

随着难度的提升子弹几乎满屏幕都是，下一届可以增加防护罩功能，屏蔽一段时间的子弹攻击