实验四报告

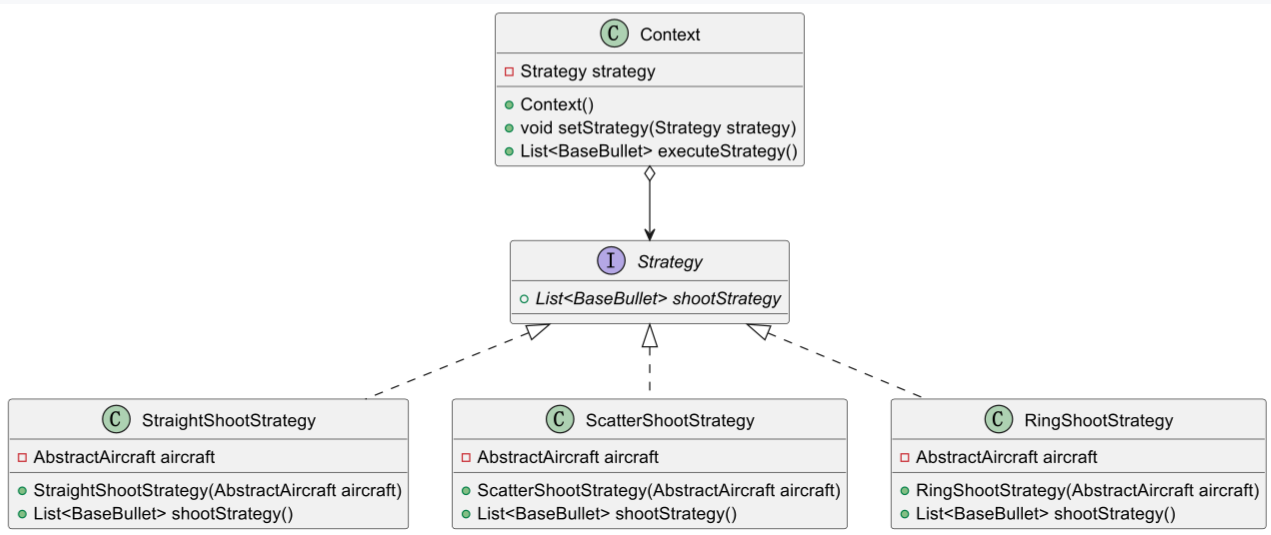
# 策略模式

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. 解决方案



Strategy 接口：shootStrategy 方法返回当前策略

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

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

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

Context 类：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。

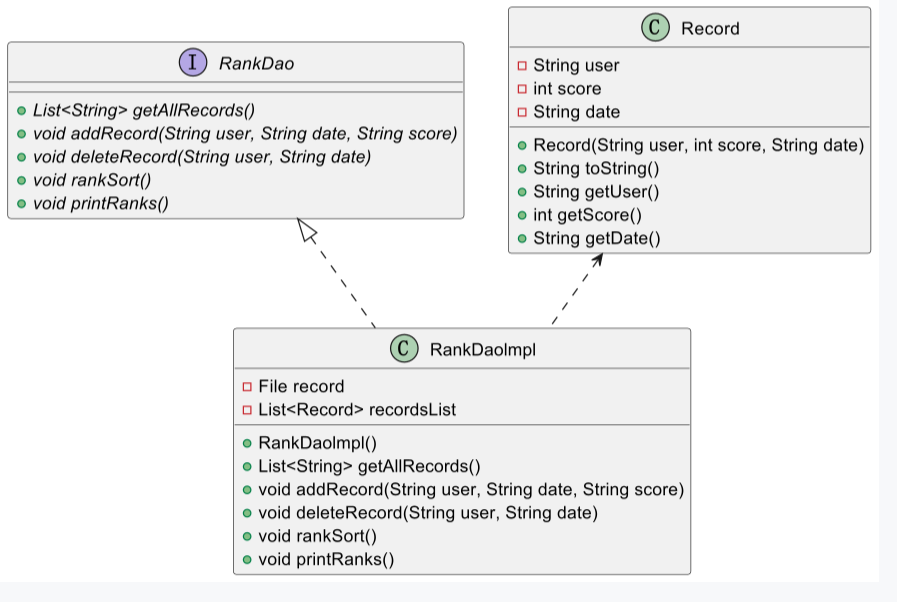
# 数据访问对象模式

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. 解决方案



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 rankSort 方法：根据得分对记录排序；

printRanks 方法：打印当前得分排行榜。

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。