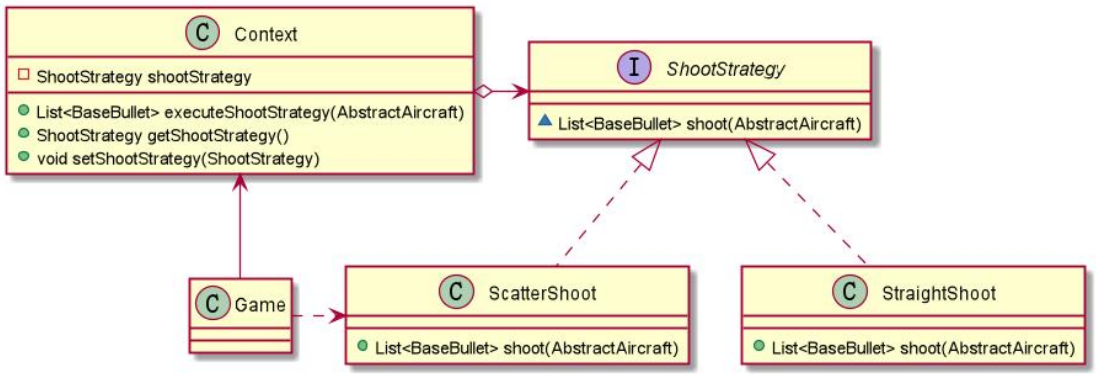
实验四报告

一、 策略模式

1. 应用场景分析

策略模式（Strategy Pattern）是一种行为型设计模式，它能让你定义一系列算法，并将每种算法分别放入独立的类中，以使算法的对象能够相互替换。
本例中，原来的代码直接由飞机调用 shoot 方法，无法使不同的射击方法分开，不利于后续添加弹道以及弹道代码的复用，原代码中（未删除，但本例实验中未调用）的 shoot 方法虽然实现了直射和散射的功能，但较为臃肿，采用策略模式能够使得代码模块化，同时不同策略的代码能够复用到不同类型的飞机上，提高了代码的复用性。

2. 解决方案



使用策略模式完成代码编写，uml 图如上所示：
Game 类：
策略模式外层的调用类，作为 client 存在
Context 类：
关键属性：
- **shootStrategy**：用于记录当前条件下需要执行的策略类型，在本例于 Game 类中实例化了 enemyShootStrategy 和 heroShootStrategy，用于根据条件改变飞行器射击的方式

关键方法：
- **executeShootStrategy**：用于执行对应的飞行器发射子弹的方法，具体而言通过将飞行器引用传入本方法，再根据飞行器类别（hero，enemy）、Context 的属性（即 shootStrategy 属性）来执行相应的射击方法。
- **setShootStrategy**：用于设置 Context 的属性，由 Game 中传入的参数进行设置

ShootStrategy 接口：

关键方法：

- **shoot**：射击方法的接口，对于不同的射击方法拥有相同的参数和返回值

ScatterShoot 方法：

关键方法：

- **shoot**：散射方法的具体实现，implement 了 ShootStrategy 接口，具体的实现而言就是在子弹上添加横向的速度，默认子弹初始位置构成抛物线

StraigtShoot 方法：

关键方法：

- **shoot**：直射方法的具体实现，implement 了 ShootStrategy 接口，具体的实现为固定子弹的初始位置为关于飞机中轴坐标对称的抛物线

二、 数据访问对象模式

1. 应用场景分析

数据访问对象模式（Data Access Object Pattern）用于把低级的数据访问 API 和操作从高级的业务服务中分离出来。本例中因为需要将排行榜每轮游戏结束时进行更新，需要使用此模式将数据的访问部分和游戏上层的结构分离出来，有利于模块化的管理和流程的控制。

本例中使用的 DAO 模式，采用 json 保存 java 对象 Record 类，Record 类的读取由 DAO 管理，使得不同的功能相分离，提高了代码的复用性和封装性，有助于对代码上层结构的修改而不改变底层的读写逻辑。

2. 解决方案

使用 DAO 模式构建底层读写逻辑，puml 类图如下所示：

Game 类：

操作 DAO 模式的上层逻辑，游戏结束后将得分记录写入 record.json 保存，并在控制台打印当前的 ranking list

Record 类：

用于记录游戏得分情况的类，提供了展示数据，写入数据的方法。

关键属性：

- **rank**：本无必要保存 rank 值，但为了方便 DAO 完成数据的读取，添加了这一属性，在增删数据时重新排序以获得准确 rank 值
- **username**：输入的用户名
- **level**：选择的难度，当前全为“MEDIUM”
- **score**：游戏结束时的得分
- **datetime**：当前的系统时间

关键方法：

- **空参构造函数**：构造一个随机指定的记录，通常没有作用只是用于测试时填充

数据

- **含参构造函数**：根据用户名和得分以及当前游戏难度，获取系统时间得到当前记录的属性（不含 rank）
- **静态方法**
  - **getDateTime**：获取当前系统时间的格式化字符串表示，注意与实例的方法重名，当然不影响使用。
  - **generateUsername**：随机数和当前系统时间组成的 String 对象 SHA256 值的前 8 位作为随机生成的用户名，仅用于测试
  - **prettyPrintHeader**：打印记录的 header，控制了精确的间隔以呈现较好的格式
- **prettyPrintRecord**：打印记录，因前提是构建了实例，故为非静态方法，控制了精确的间隔以呈现较好的格式。Java 中没有运算符重载
- **getter、setter**：实例属性的 getter 和 setter，分别用于读取和设置实例的属性值。

RecordDAO 接口（详细描述见 **RecordDAOImpl**）：

关键方法：

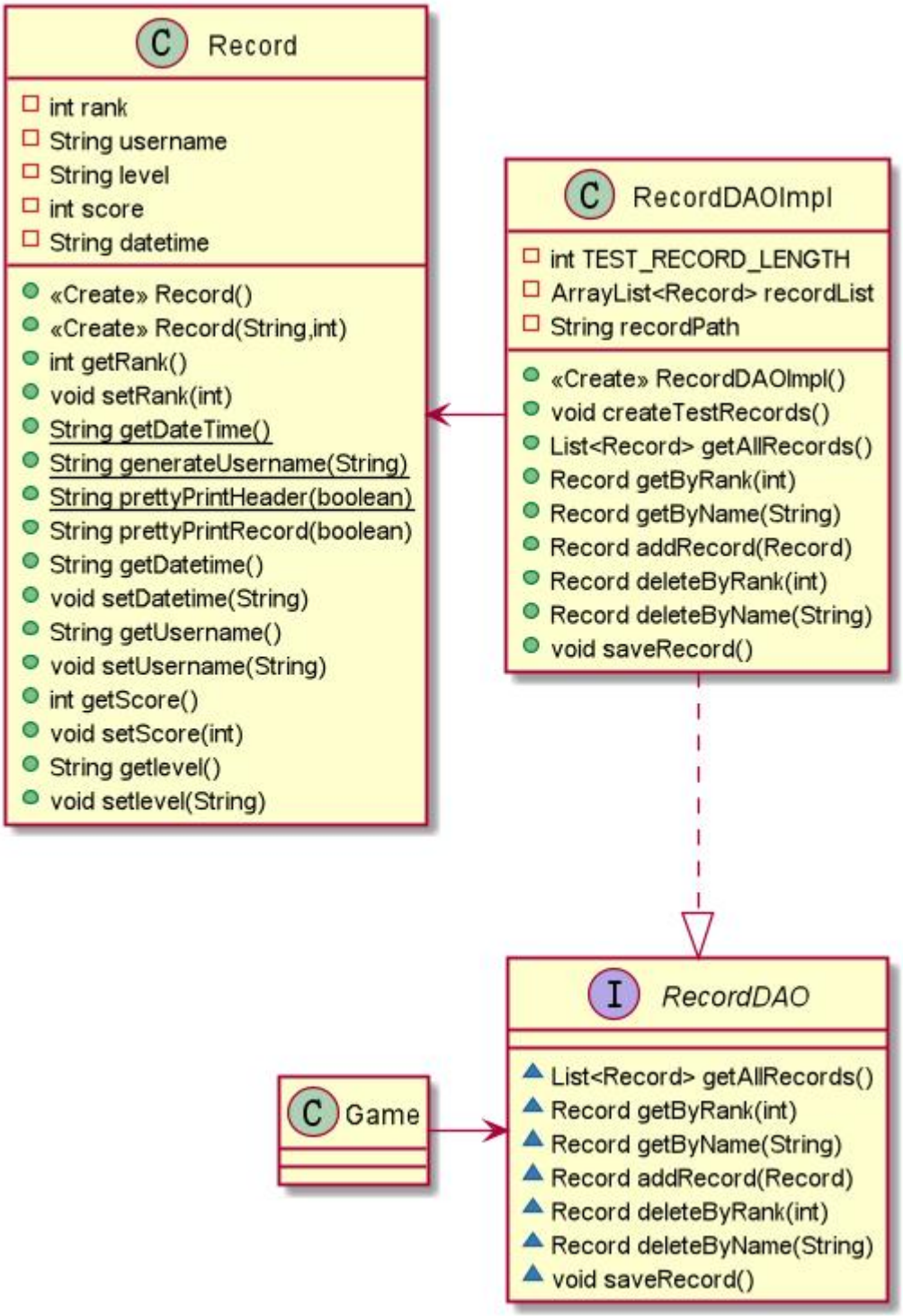
- `List<Record> getAllRecords();`
- `Record getByRank(int rank);`
- `Record getByName(String name);`
- `Record addRecord(Record record);`
- `Record deleteByRank(int rank);`
- `Record deleteByName(String name);`
- `void saveRecord();`

RecordDAOImpl 方法：

实现了 RecordDAO 接口，用于完成数据的访问（读写、排序、删改操作等）

关键方法：

- **List<Record> getAllRecords()**
  返回记录的 ArrayList 引用，默认使用 Record 实例的 prettyPrintRecord 方法打印每一条记录
- **Record getByRank(int rank)**
  返回按排名查询得到的记录，需要排序获得正确的得分顺序（效率更高的做法应当是构建一个排序标记，只要无序状态时读取才进行排序）
- **Record getByName(String name)**
  返回按用户名查询所得记录。（此处未进行排序，非正确做法。）
- **Record addRecord(Record record)**
  添加一条记录，因需要写入 rank 属性，故需要进行排序，但非有效率做法。
- **Record deleteByRank(int rank)**
  根据排名删除一条记录，未进行排序。
- **Record deleteByName(String name);**
  根据用户名删除一条记录，未进行排序
- **void saveRecord();**
  将 Java 对象写以 json 格式写入文件，此处使用了 gson 包来完成 json 的读写。


Record
int rank
String username
String level
int score
String datetime
«Create» Record()
«Create» Record(String,int)
int getRank()
void setRank(int)
String getDateTime()
String generateUsername(String)
String prettyPrintHeader(boolean)
String prettyPrintRecord(boolean)
String getDatetime()
void setDatetime(String)
String getUsername()
void setUsername(String)
int getScore()
void setScore(int)
String getlevel()
void setlevel(String)
RecordDAOImpl
int TEST_RECORD_LENGTH
ArrayList<Record> recordList
String recordPath
«Create» RecordDAOImpl()
void createTestRecords()
List<Record> getAllRecords()
Record getByRank(int)
Record getByName(String)
Record addRecord(Record)
Record deleteByRank(int)
Record deleteByName(String)
void saveRecord()
RecordDAO
List<Record> getAllRecords()
Record getByRank(int)
Record getByName(String)
Record addRecord(Record)
Record deleteByRank(int)
Record deleteByName(String)
void saveRecord()
Game