哈尔滨工业大学（深圳）

面向对象的软件构造导论
实验指导书

实验四 设计模式实验（2）

—— 策略模式和数据访问对象模式

2023 春

目录

1. 实验目的

1. 理解策略模式和数据访问对象模式的模式动机和意图，掌握模式结构；
2. 结合实例，熟练绘制策略和数据访问对象两种模式的 UML 结构图；
3. 重构代码，熟练使用代码实现策略和数据访问对象两种模式。

2. 实验环境

1. Windows 10
2. IntelliJ IDEA 2022.3.2
3. Java 11

3. 实验内容（2 学时）

（1）结合实例，绘制策略模式的 UML 结构图；
（2）根据类图，重构代码，采用策略模式实现不同机型的火力弹道及火力道具的加成效果。
（3）结合实例，绘制数据访问对象模式的 UML 结构图。
（4）根据类图，重构代码，采用数据访问对象模式实现玩家的得分排行榜。

4. 实验步骤

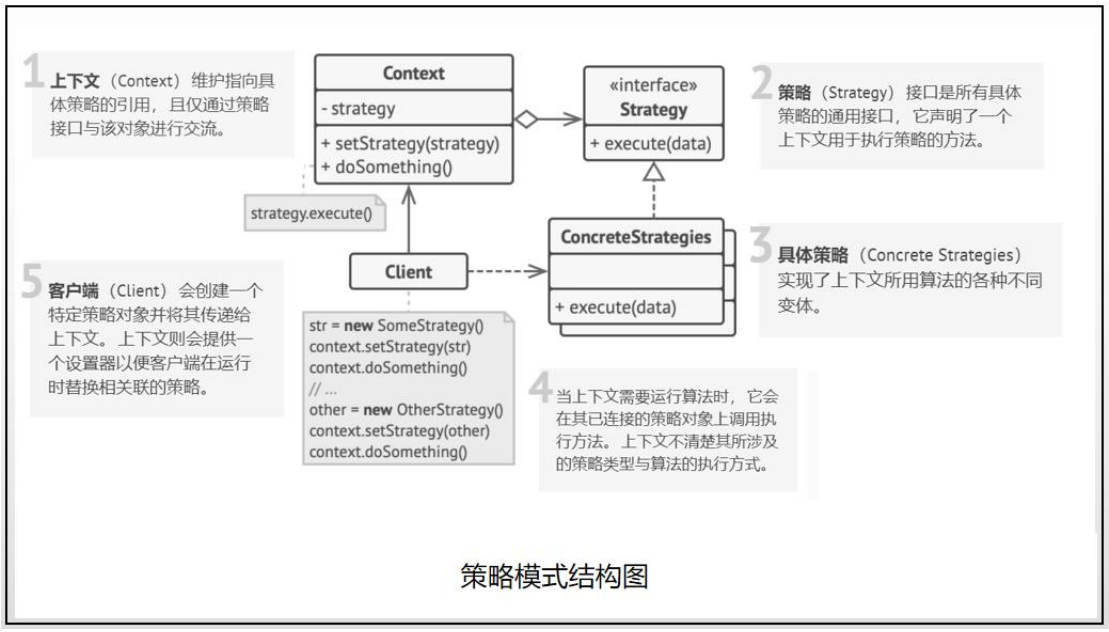
4.1 结合飞机大战实例，绘制策略模式的 UML 结构图

在飞机大战游戏中，英雄机和各种类型的敌机所发射子弹的图片、子弹数量、火力值和弹道都不相同。英雄机自动发射子弹，如有火力道具加成则可以改变弹道、火力值、子弹数量等，如由直射改为散射。普通敌机不发射子弹，精英敌机和 Boss 敌机按各自方式发射子弹，如 Boss 敌机可散射。

请结合该实例场景，为不同子弹发射绘制策略模式的 UML 结构图，要求给出设计模式的名称，类名、方法名和属性名可自行定义。

策略模式

策略模式（Strategy Pattern）是一种行为设计模式， 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。



策略模式结构图

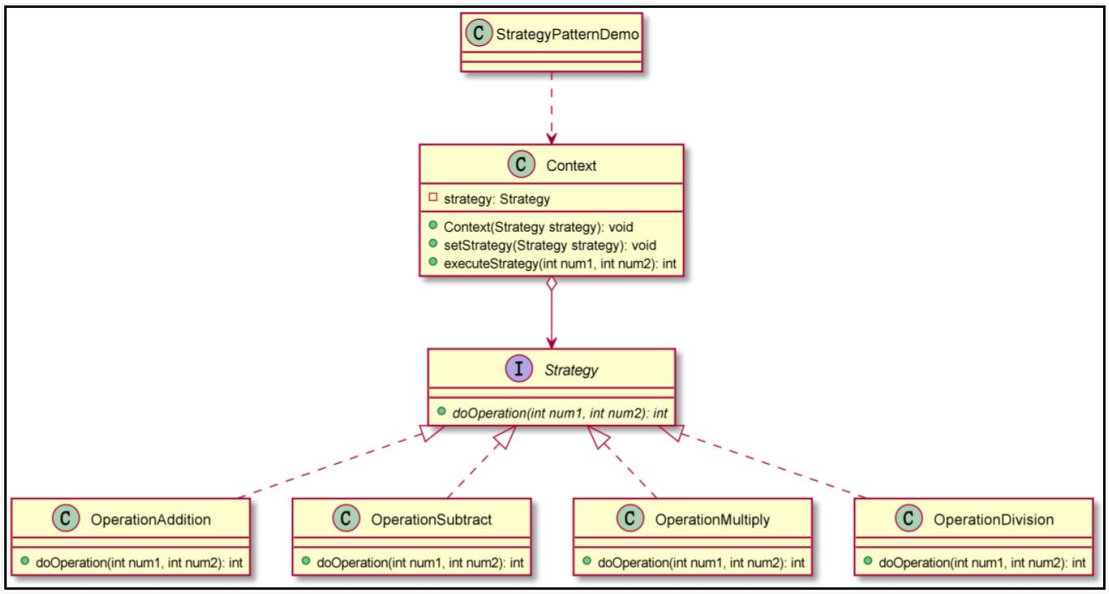
请参考以上 UML 结构图，绘制飞机大战中的策略模式。

4.2 根据设计的类图，重构代码，实现策略模式

根据 4.1 中你所设计的 UML 类图，重构代码，采用策略模式实现不同机型的火力发射和火力道具加成效果。

策略模式代码示例：

我们将创建一个定义活动的 Strategy 接口和实现了 Strategy 接口的实体策略类。Context 是一个使用了某种策略的类。StrategyPatternDemo，我们的演示类使用 Context 和策略对象来演示 Context 在它所配置或使用的策略改变时的行为变化。



步骤 1：创建一个接口，充当抽象策略角色。

Strategy.java

```
public interface Strategy {
    int doOperation(int num1, int num2);
}
```

步骤 2：创建实现接口的实体类，充当具体策略角色。

OperationAdd.java

```
public class OperationAdd implements Strategy{
    @Override
    public int doOperation(int num1, int num2) {
        return num1 + num2;
    }
}
```

OperationSubtract.java

```
public class OperationSubtract implements Strategy{
    @Override
    public int doOperation(int num1, int num2) {
        return num1 - num2;
    }
}
```

OperationMultiply.java

```
public class OperationMultiply implements Strategy{
    @Override
    public int doOperation(int num1, int num2) {
        return num1 * num2;
    }
}
```

OperationDivision.java

```
public class OperationDivision implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
         try {
               int num3 = num1 / num2;
         } catch (Exception e) {
               e.printStackTrace();
         }
         return num1 / num2;
   }
}
```

步骤 3：创建 Context 类。

Context.java

```
public class Context {
   private Strategy strategy;

   public Context(Strategy strategy){
      this.strategy = strategy;
   }

    public void setStrategy(Strategy strategy) {
      this.strategy = strategy;
   }

   public int executeStrategy(int num1, int num2){
      return strategy.doOperation(num1, num2);
   }
}
```

步骤 4：使用 Context 来查看当它改变策略 Strategy 时的行为变化。

StrategyPatternDemo.java

```
public class StrategyPatternDemo {
   public static void main(String[] args) {

      Context context = new Context(new OperationAddition());
      System.out.println("10 + 5 = " + context.executeStrategy(10, 5));

      context.setStrategy(new OperationSubtract());
      System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

      context.setStrategy(new OperationMultiply());
      System.out.println("10 * 5 = " + context.executeStrategy(10, 5));

      context.setStrategy(new OperationDivision());
      System.out.println("10 / 5 = " + context.executeStrategy(10, 5));
   }
}
```

步骤 5：执行程序，输出结果：

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2
```

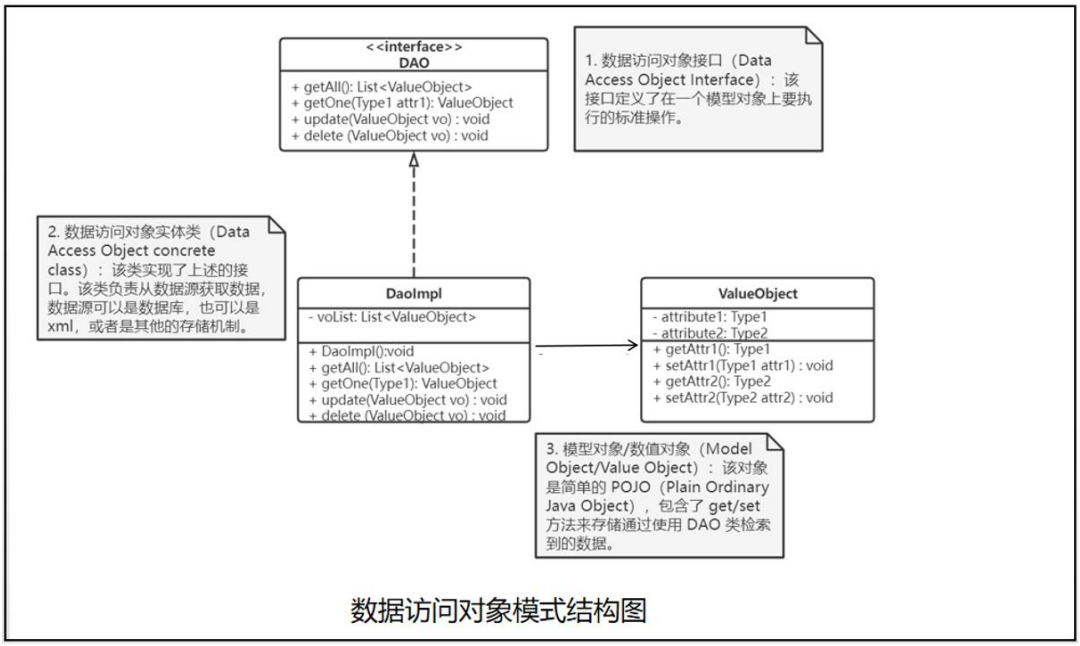
4.3 结合飞机大战实例，绘制数据访问对象模式 UML 结构图

在飞机大战游戏中，每局游戏过程中记录英雄机得分，在界面左上角显示。英雄机击落敌机后可增加相应的分数。每局游戏结束后，显示该难度的总分排行榜，内容包括“名次”、“玩家名”、“得分”和“记录时间”。每局游戏结束后询问玩家是否存储记录，如存储则插入新的游戏记录。

请结合该实例场景，为创建排行榜绘制数据访问对象模式的 UML 结构图，要求给出设计模式的名称，类名、方法名和属性名可自行定义。

数据访问对象模式

数据访问对象模式（Data Access Object Pattern）或 DAO 模式用于把低级的数据访问 API 或操作从高级的业务服务中分离出来。



数据访问对象模式结构图

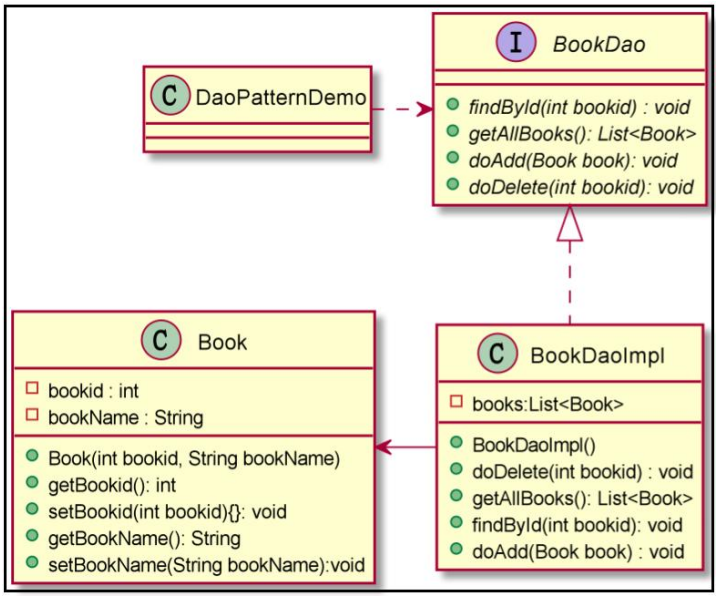
请参考以上 UML 结构图，绘制飞机大战中的数据访问对象模式。

4.4 根据设计的类图，重构代码，实现数据访问对象模式

根据 4.3 中你所设计的 UML 类图，重构代码，采用数据访问对象模式实现得分排行榜。本次实验只需要在游戏结束后记录该局分数，并在控制台打印输出得分排行榜即可，无需实现界面和玩家交互。得分数据可存储在文件中。

数据访问对象模式代码示例：

我们将创建一个作为模型对象或数值对象的 Book 对象。BookDao 是数据访问对象接口。BookDaoImpl 是实现了数据访问对象接口的实体类。DaoPatternDemo 是我们的演示类，使用 BookDao 来演示数据访问对象模式的用法。



步骤 1：创建数值对象 VO 实体类。

Book.java

```
public class Book {
    private int bookid;
    private String bookName;

    Book(int bookid, String bookName) {
        this.bookid = bookid;
        this.bookName = bookName;
    }

    public int getBookid() {
        return bookid;
    }

    public void setBookid(int bookid) {
        this.bookid = bookid;
    }
```

```
    public String getBookName() {
        return bookName;
    }

    public void setBookName(String bookName) {
        this.bookName = bookName;
    }

}
```

步骤 2：创建数 DAO 接口。

BookDao.java

```
public interface BookDao {
    void findById(int bookid);

    List<Book> getAllBooks();

    void doAdd(Book book);

    void doDelete(int bookid);
}
```

步骤 3：创建实现了上述接口的 DAO 实现类。

BookDaoImpl.java

```
public class BookDaoImpl implements BookDao {

    //模拟数据库数据
    private List<Book> books;

    public BookDaoImpl() {
        books = new ArrayList<Book>();
        books.add(new Book(1001, "Clean Code"));
        books.add(new Book(1002, "Design Patterns"));
        books.add(new Book(1003, "Effective Java"));
    }

    //删除图书
    @Override
    public void doDelete(int bookid) {
        for (Book item : books) {
            if (item.getBookid() == bookid) {
                books.remove(item);
                System.out.println("Delete Book: ID [" + bookid + "]");
                return;
            }
        }
        System.out.println("Can not find this book!");
    }
```

```
    //获取所有图书
    @Override
    public List<Book> getAllBooks() {
        return books;
    }

    //查找图书
    @Override
    public void findById(int bookid) {
        for (Book item : books) {
            if (item.getBookid() == bookid) {
                System.out.println("Find Book: ID [" + bookid + "],
                                        Book Name [" + item.getBookName() + "]");
                return;
            }
        }
        System.out.println("Can not find this book!");
    }

    //新增图书
    @Override
    public void doAdd(Book book) {
        books.add(book);
        System.out.println("Add new Book: ID [" + book.getBookid() + "],
                                Book Name [" + book.getBookName() + "]");
    }
}
```

步骤 4：使用 DaoPatternDemo 来演示数据访问对象模式的用法。

DaoPatternDemo.java

```
public class DaoPatternDemo {
    public static void main(String[] args) {

        BookDao bookDao = new BookDaoImpl();

        //输出所有图书
        for (Book book : bookDao.getAllBooks()) {
            System.out.println("Book ID [" + book.getBookid() + "],
                                    Book Name : [" + book.getBookName() + "]");
        }
        System.out.println("-----------------------------------------------");

        //查找图书
        bookDao.findById(1002);
        System.out.println("-----------------------------------------------");

        //删除图书
        bookDao.doDelete(1002);
        System.out.println("-----------------------------------------------");

        //新增图书
        Book newBook = new Book(1004, "Thinking In java");
```

```
            bookDao.doAdd(newBook);
            System.out.println("--------------------------------------------");

            //输出所有图书
            for (Book book : bookDao.getAllBooks()) {
                System.out.println("Book ID [" + book.getBookid() + "],
                                    Book Name : [" + book.getBookName() + "]");
            }
        }
}
```

步骤 5：执行程序，输出结果：

```
Book ID [1001], Book Name : [Clean Code]
Book ID [1002], Book Name : [Design Patterns]
Book ID [1003], Book Name : [Effective Java]
---------------------------------------------------------------------------
Find Book: ID [1002], Book Name [Design Patterns]
---------------------------------------------------------------------------
Delete Book: ID [1002]
---------------------------------------------------------------------------
Add new Book: ID [1004], Book Name [Thinking In java]
---------------------------------------------------------------------------
Book ID [1001], Book Name : [Clean Code]
Book ID [1003], Book Name : [Effective Java]
Book ID [1004], Book Name : [Thinking In java]
```

5. 实验要求

实验课前，预习并理解策略模式和数据访问模式的基本要素，包括模式名称、问题描述、解决方案和应用效果。

本次实验提交版本需完成以下功能：

✓ 使用策略模式重构代码，实现直射、散射两种子弹发射弹道；
✓ 火力道具生效后，英雄机弹道由直射改为散射；
✓ 每局游戏结束后在控制台打印输出得分排行榜，得分数据存储在文件中。

注意：

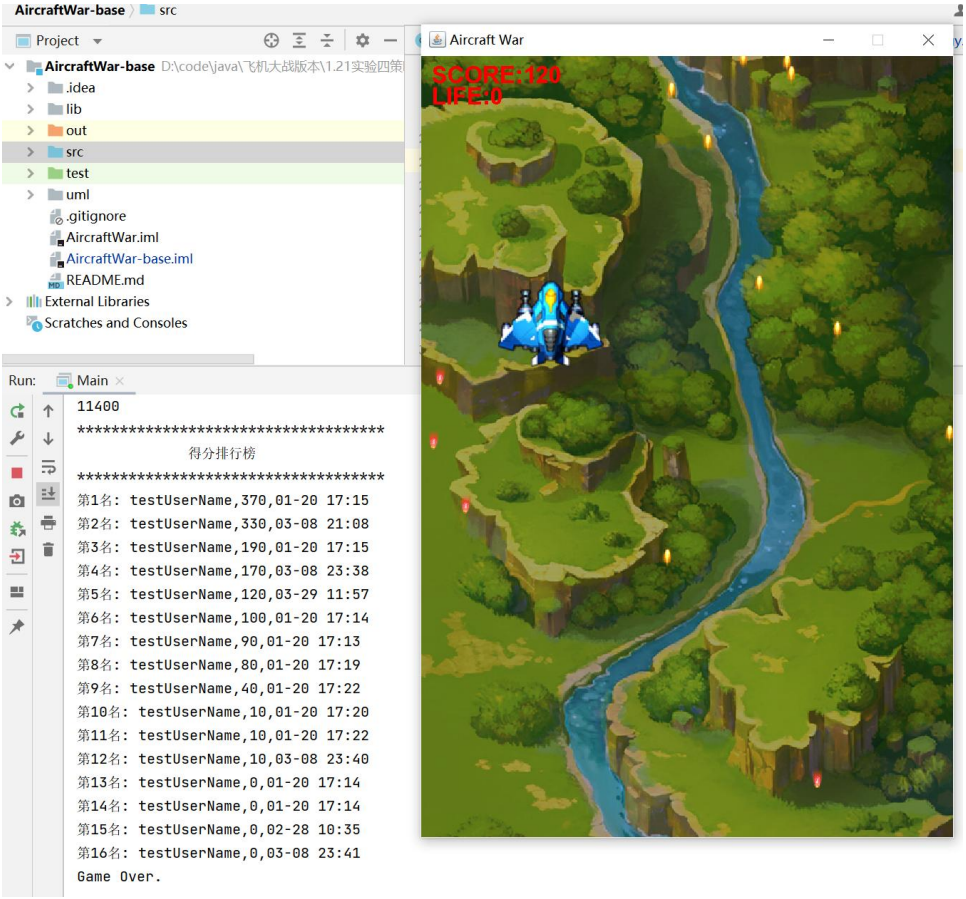
(1) 本次实验未涉及多线程，故弹道改变后无法恢复，实验五继续完善即可。
(2) 得分排行榜无需实现界面和玩家交互。

✧ **提交内容**

包括：

① 项目压缩包（整个项目压缩成 zip 包提交，包含代码、uml 图等）
② 实验截图报告（设计模式类图和说明，请使用报告模板）

参考输出格式：


AircraftWar-base
src
Project
AircraftWar-base D:\code\java\飞机大战版本\1.21实验四第
.idea
lib
out
src
test
uml
.gitignore
AircraftWar.iml
AircraftWar-base.iml
README.md
External Libraries
Scratches and Consoles
Aircraft War
SCORE:120
LIFE:0
Run:
Main
11400
****
得分排行榜
****
第1名: testUserName,370,01-20 17:15
第2名: testUserName,330,03-08 21:08
第3名: testUserName,190,01-20 17:15
第4名: testUserName,170,03-08 23:38
第5名: testUserName,120,03-29 11:57
第6名: testUserName,100,01-20 17:14
第7名: testUserName,90,01-20 17:13
第8名: testUserName,80,01-20 17:19
第9名: testUserName,40,01-20 17:22
第10名: testUserName,10,01-20 17:20
第11名: testUserName,10,01-20 17:22
第12名: testUserName,10,03-08 23:40
第13名: testUserName,0,01-20 17:14
第14名: testUserName,0,01-20 17:14
第15名: testUserName,0,02-28 10:35
第16名: testUserName,0,03-08 23:41
Game Over.