设计模式实验（2）

实验四：策略模式和数据访问对象模式

2023春

哈尔滨工业大学（深圳）

本学期实验总体安排

实验项目	一	二	三	四	五	六
学时数	2	2	2	**2**	4（2+2）	4
实验内容	飞机大战功能分析	单例模式 工厂模式	Junit与单元测试	**策略模式 数据访问对象模式**	Swing 多线程	模板模式 观察者模式
分数	4	6	4	**6**	6	14
提交内容	UML类图、代码	UML类图、代码	单元测试代码 测试报告	**UML类图、代码**	代码	项目代码、实验报告、展示视频

实验课程共16个学时，6个实验项目，总成绩为40分。

目录

实验目的

- 理解策略模式和数据访问对象模式的意义，掌握模式结构；
- 掌握绘制策略和数据访问对象模式的UML类图；
- 熟练使用代码实现策略和数据访问对象模式。

实验任务

绘制类图、重构代码，完成以下功能：

1. 采用策略模式实现不同机型的弹道发射及火力道具的加成效果；

2. 采用数据访问对象模式实现玩家的得分排行榜。

注意： 结合飞机大战实例，完成模式UML类图设计后，再进行编码，先“设计”再“编码”！

实验步骤

1 弹道应用场景分析

应用场景分析

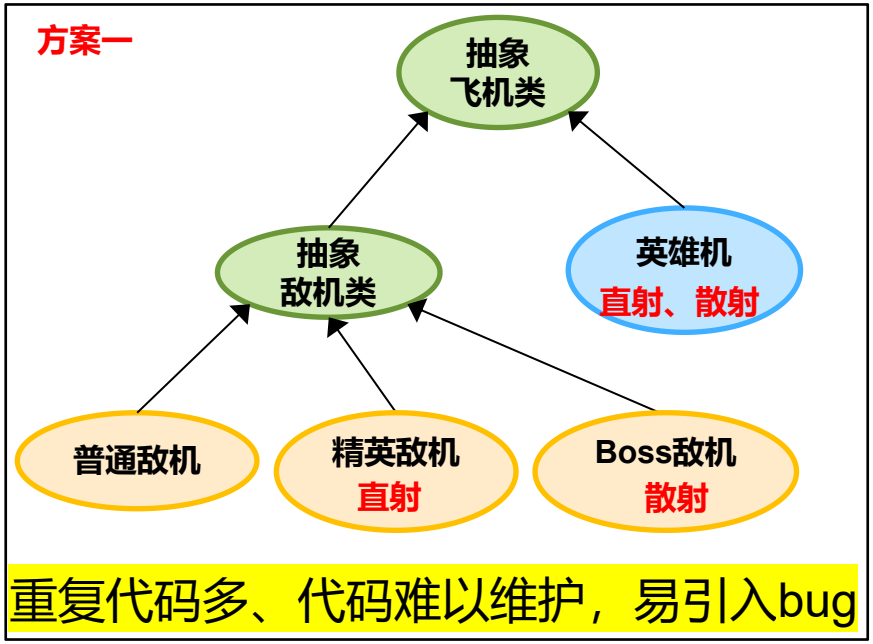
在飞机大战游戏中，英雄机直射，火力道具生效时弹道切换为散射；精英敌机直射；Boss敌机散射。

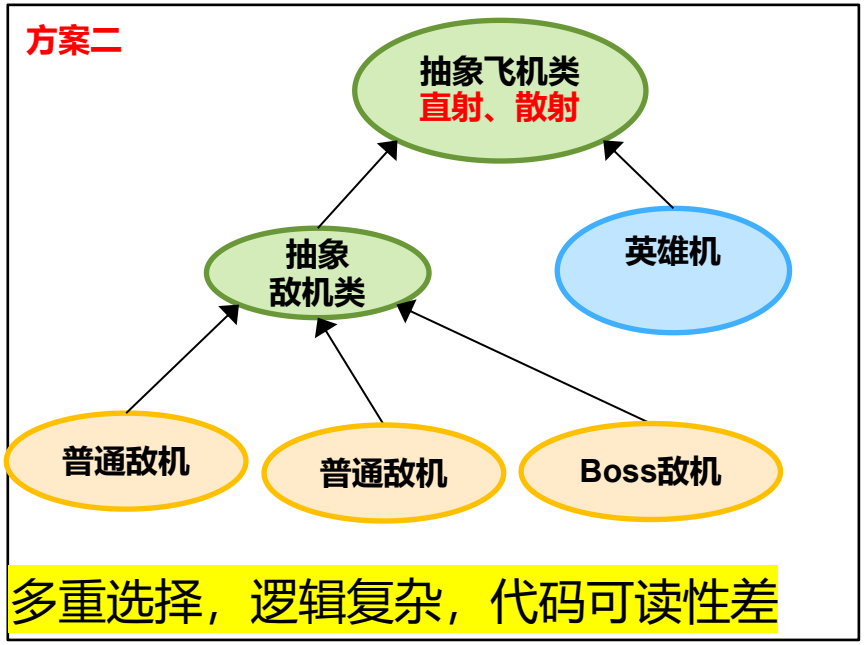
实验步骤

1 弹道应用场景分析

请思考：

1. 目前各种飞机的子弹发射在哪个类实现？如何实现火力道具？



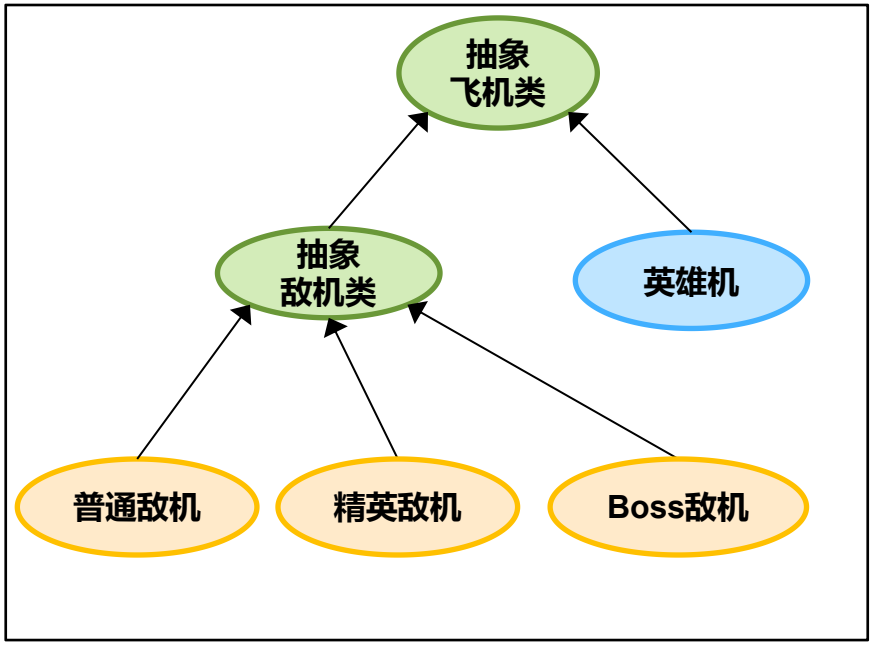


实验步骤

1 弹道应用场景分析

请思考：

2. 若增加一种新的机型或一种新的弹道需要修改那些类的代码？



违反
开闭原则 X

违反
合成复用原则 X

尽量使用对象组合，而不是继承来达到代码复用的目的。

实验步骤

2 绘制策略模式类图

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

关键代码：将这些算法封装成一个一个的类,实现同一个接口。

实验步骤

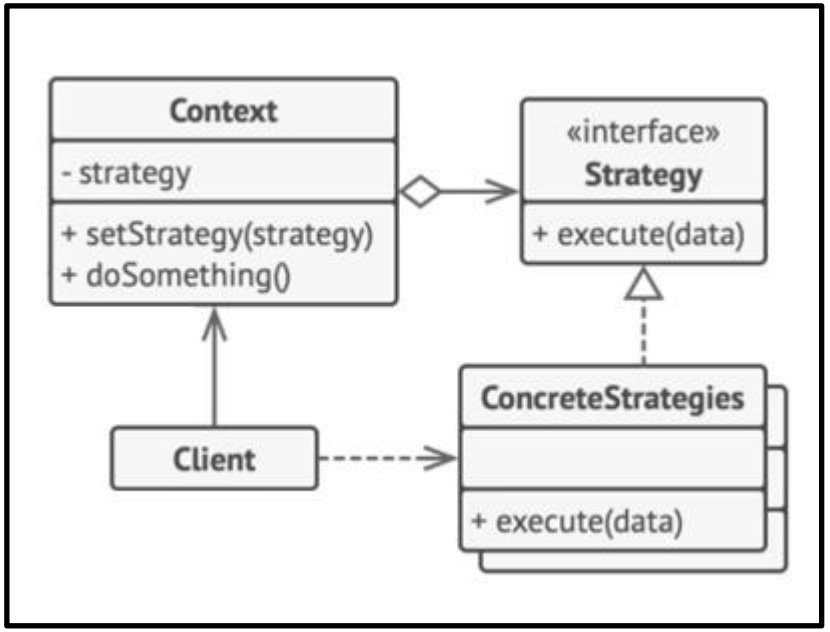
2 绘制策略模式类图

假如我们要实现一个计算器，完成加、减、乘、除四个功能。我们该如何绘制UML类图?

实验步骤

2 绘制策略模式类图

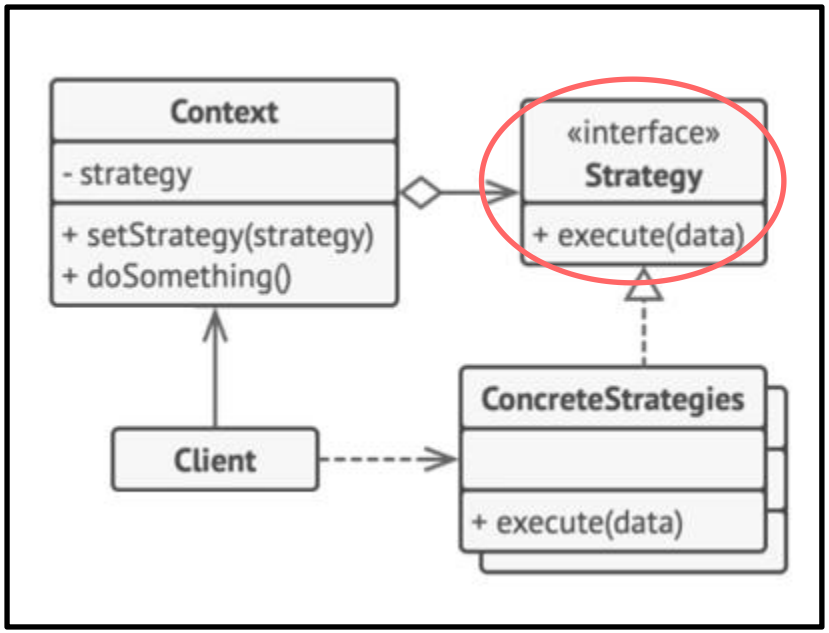
假如我们要实现一个计算器，完成加、减、乘、除四个功能。我们该如何绘制UML类图?

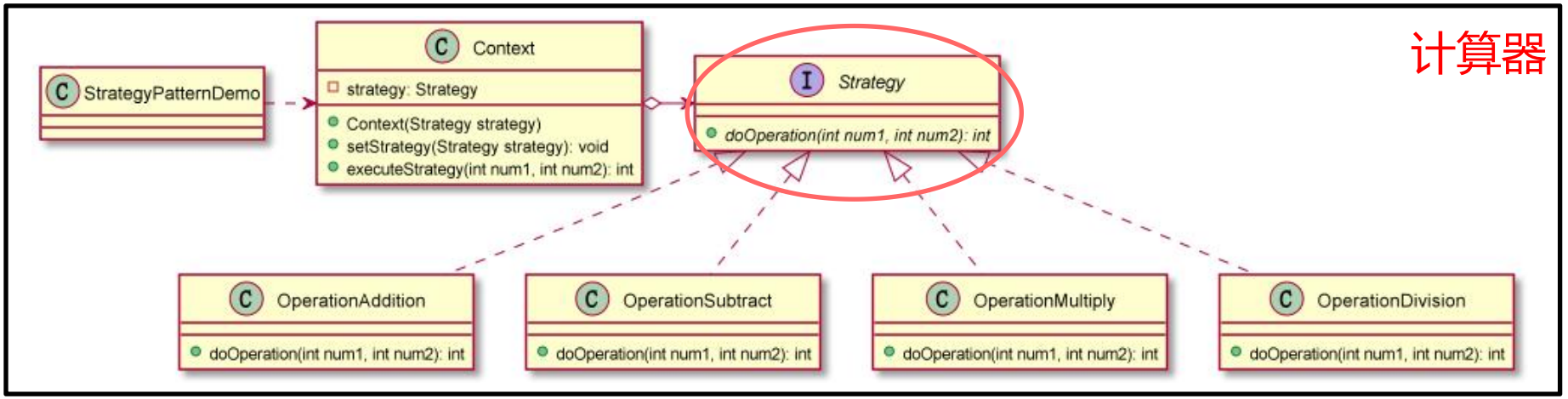


实验步骤

2 绘制策略模式类图

① 创建一个Strategy接口；

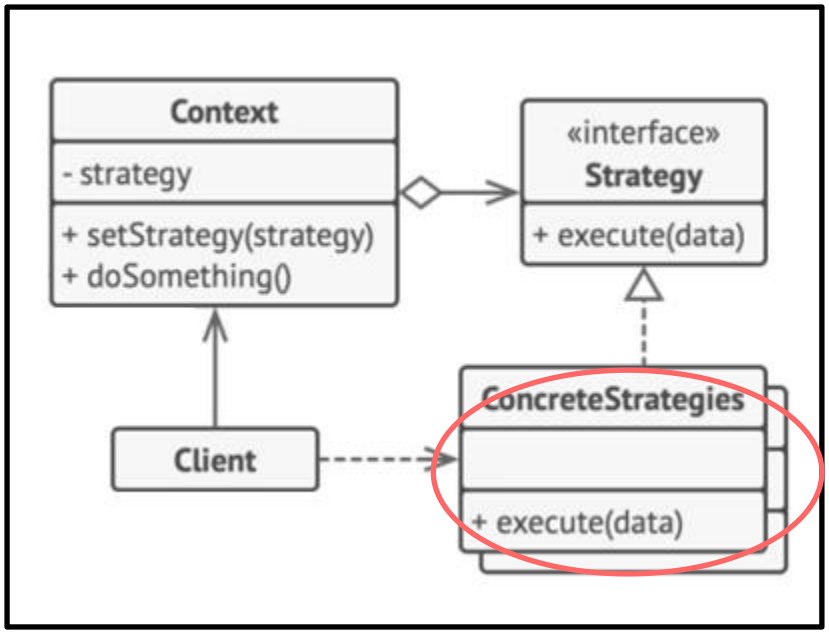


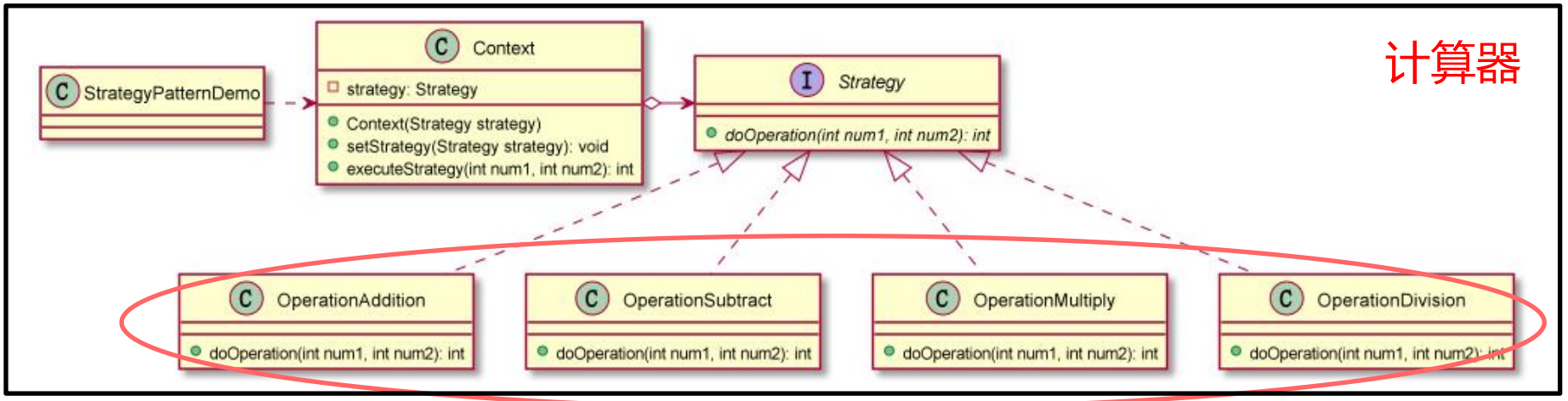


实验步骤

2 绘制策略模式类图

② 创建实现Strategy接口的实体策略类；

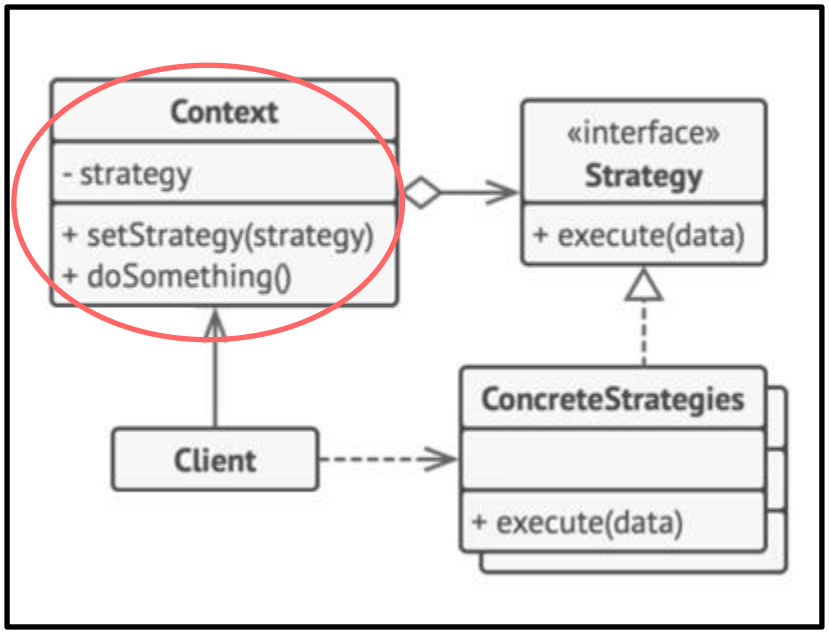


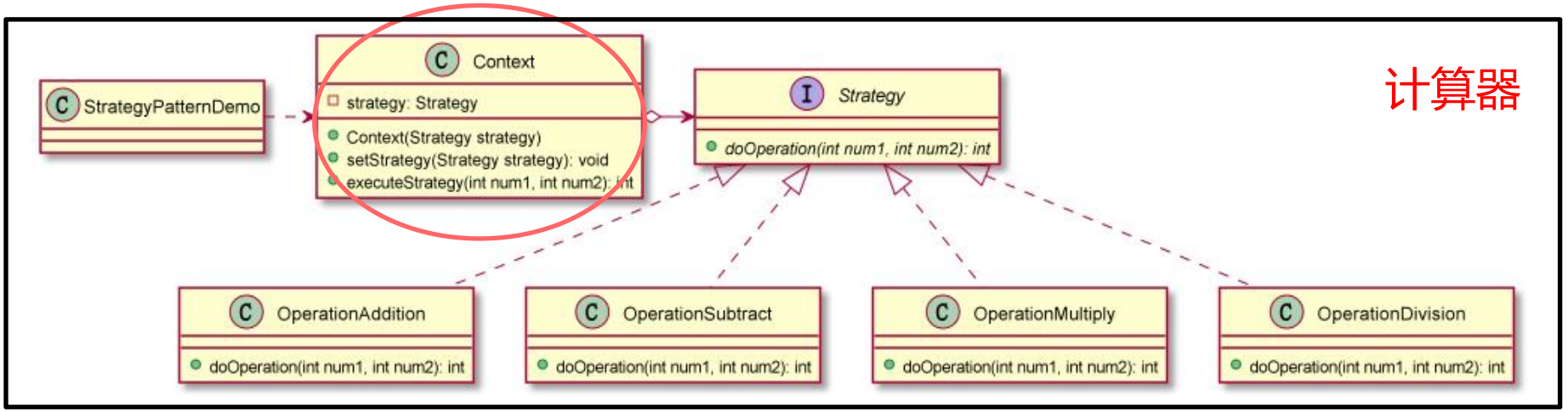


实验步骤

2 绘制策略模式类图

③ Context是一个使用了策略接口的类；

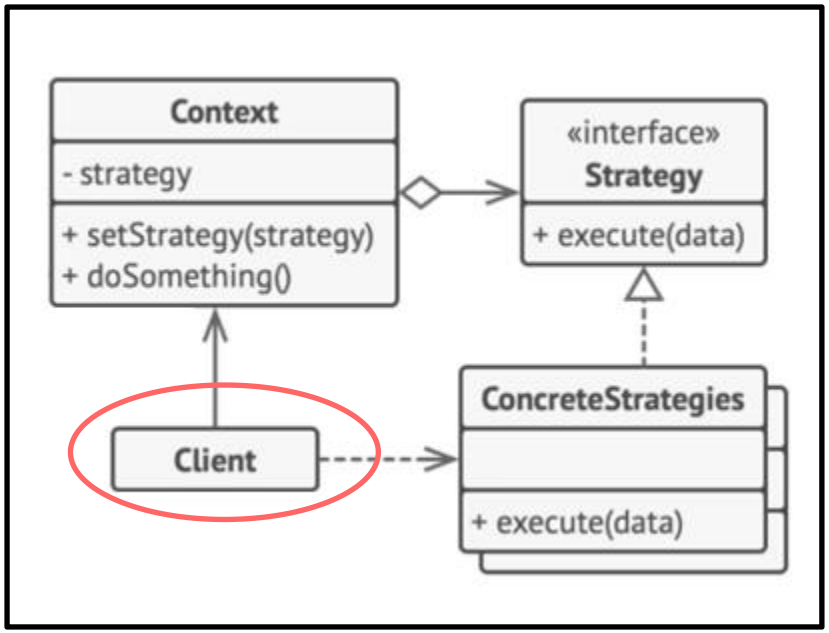


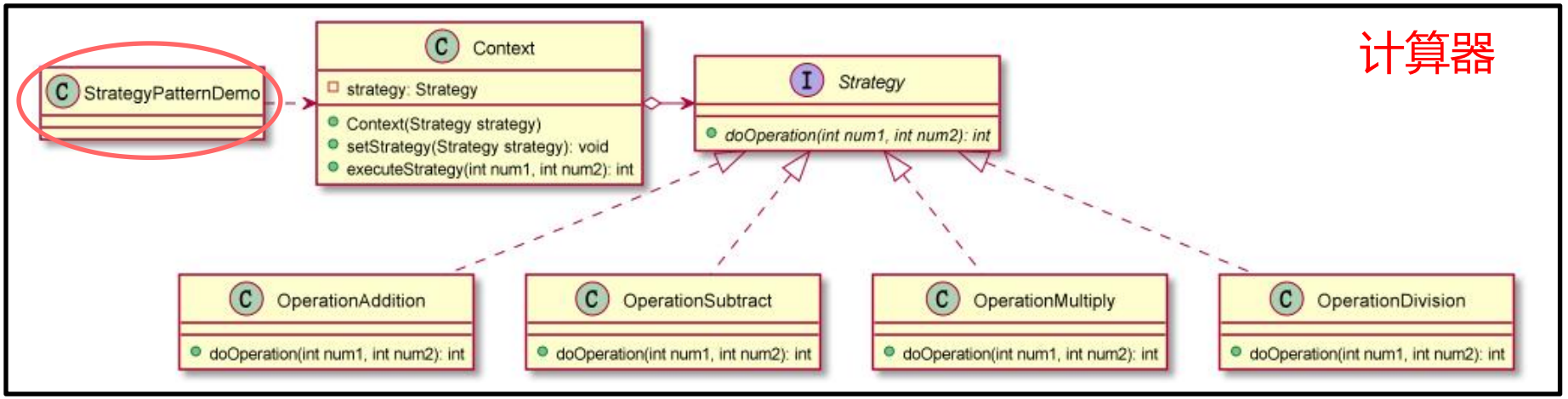


实验步骤

2 绘制策略模式类图

④ StrategyPatternDemo是客户端，在本例中演示Context在它所使用的策略改变时的行为变化。

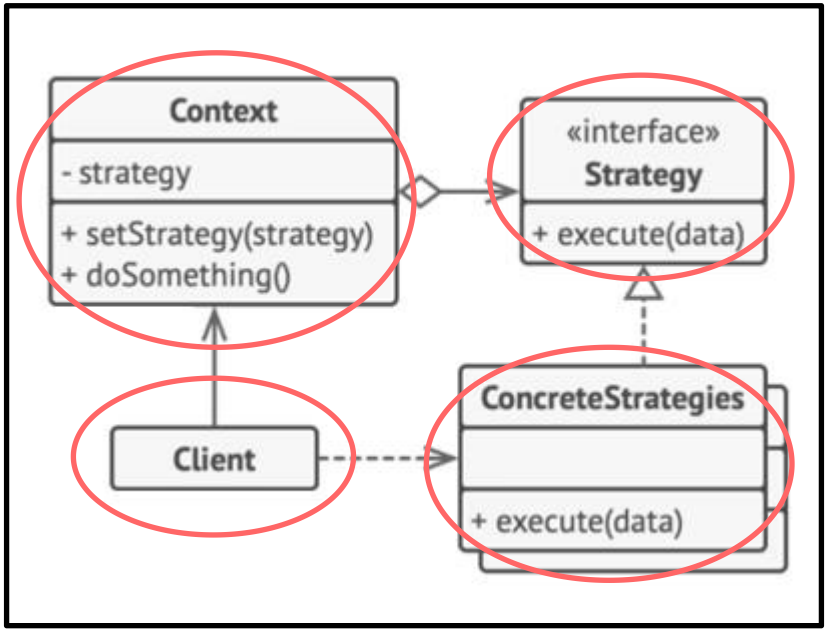




实验步骤

2 绘制策略模式类图

请思考：在飞机大战系统中，若采用策略模式实现不同的弹道，该如何设计？



实验步骤

3 重构代码，实现策略模式

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

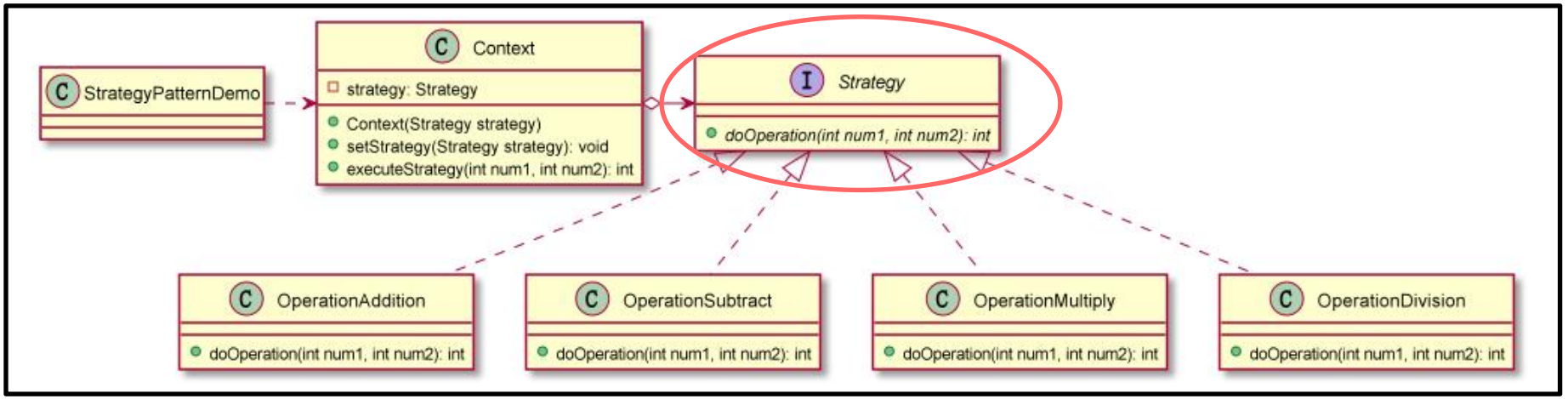
实验步骤

3 重构代码，实现策略模式

- 策略模式代码示例（计算器）

① 创建一个接口，充当抽象策略角色。

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



实验步骤

3 重构代码，实现策略模式

- 策略模式代码示例（计算器）

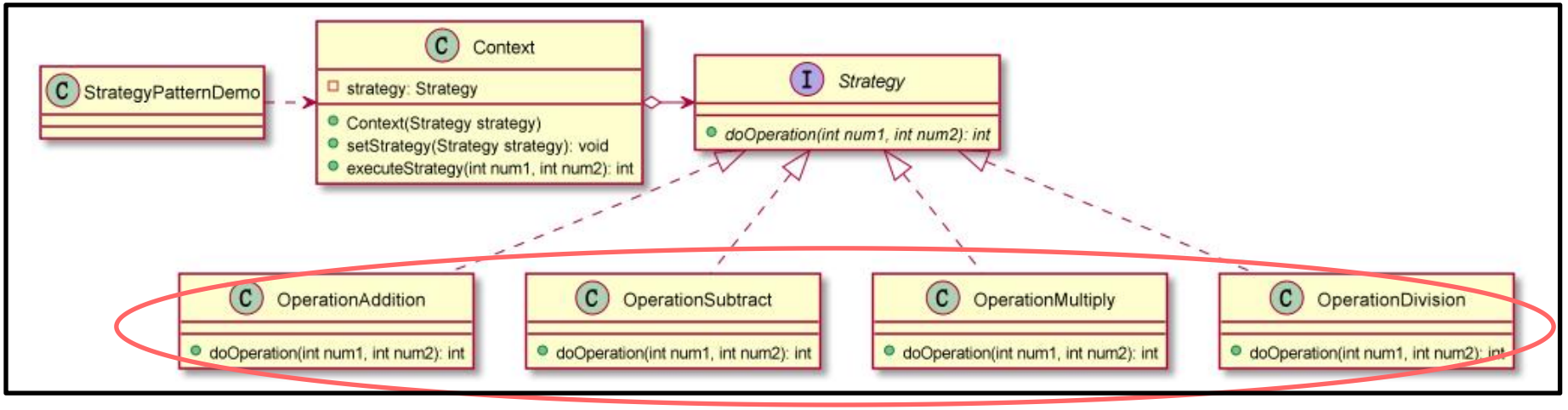
② 创建实现接口的实体类，充当具体策略角色；

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

```java
public class OperationSubtract implements Strategy{

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

```java
public class OperationMultiply implements Strategy{

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

...



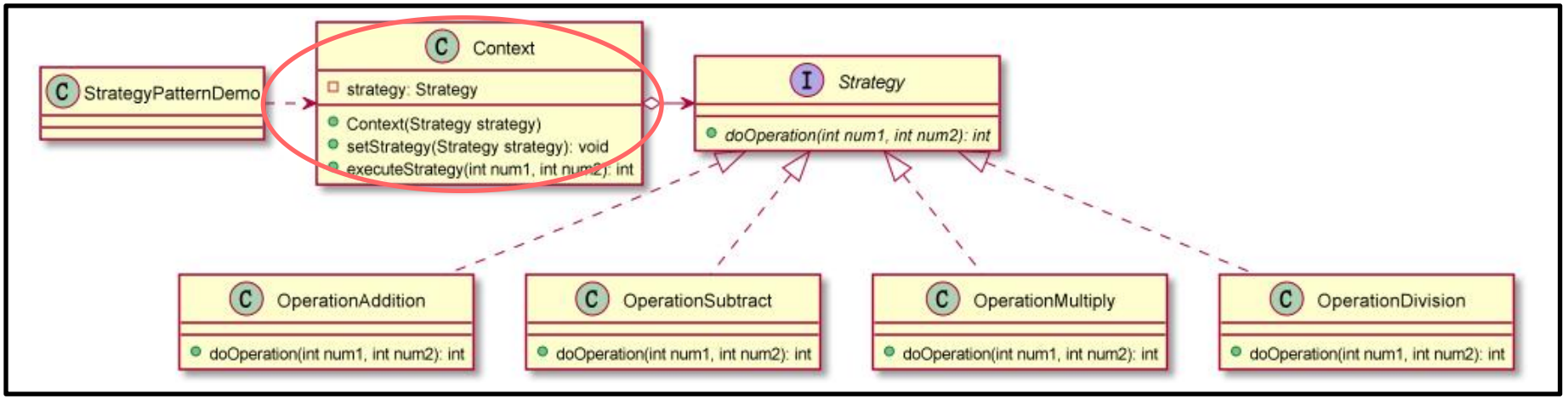
实验步骤

3 重构代码，实现策略模式

- 策略模式代码示例（计算器）

③ 创建 Context 类；

```
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);
    }
}
```



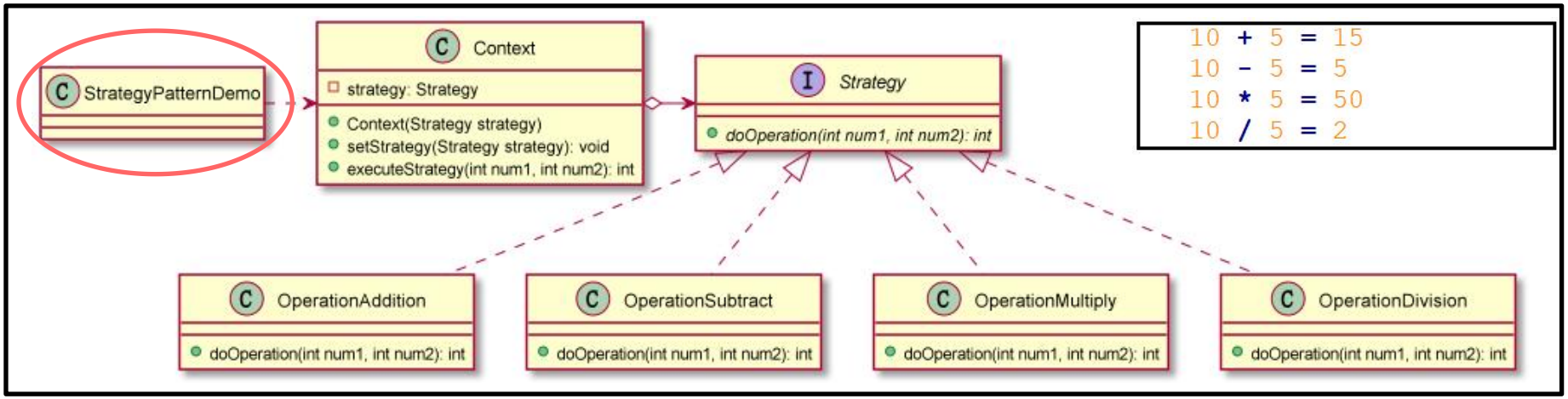
实验步骤

3 重构代码，实现策略模式

- 策略模式代码示例（计算器）

④ 客户端使用 Context 来查看当策略 Strategy 发生变化时行为的变化。

```
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));
    }
}
```



实验步骤

4 排行榜应用场景分析

应用场景分析

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。玩家可以删除某条选中记录。

内容包括：名次、玩家名、得分和记录时间。

（本次实验无需与用户交互，控制台输出即可）

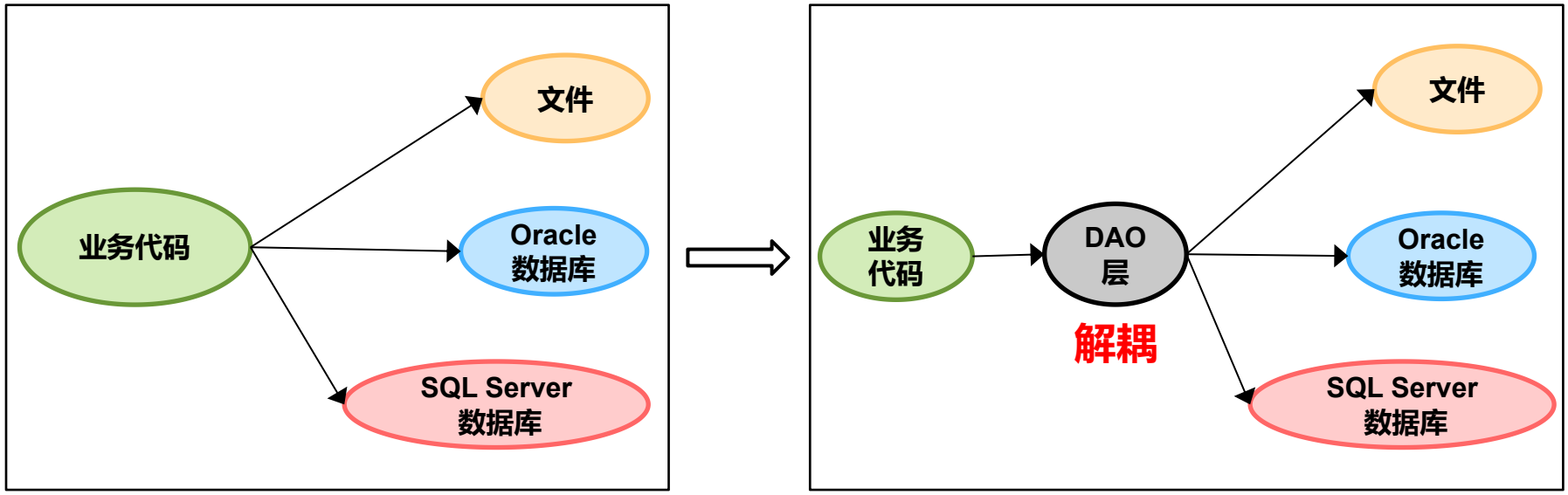
实验步骤

4 排行榜应用场景分析

请思考：

1. 玩家得分数据存储在哪里？如何获取和修改玩家的得分？

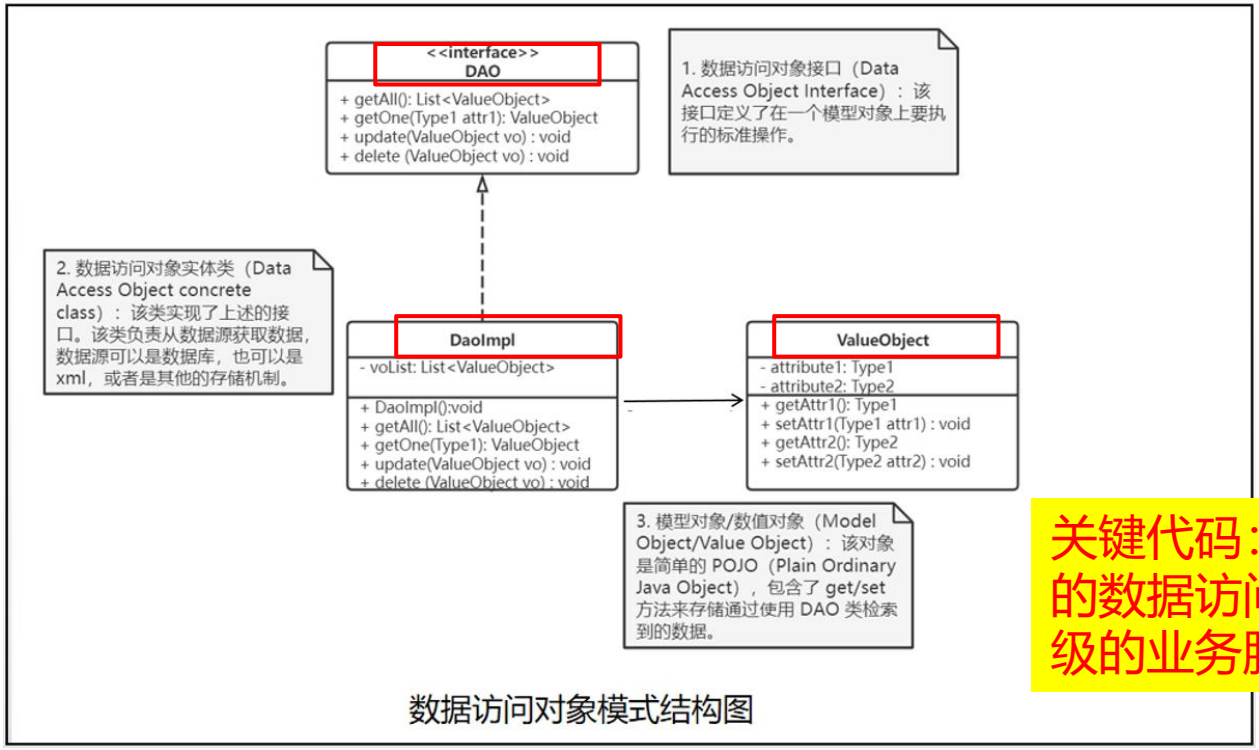
2. 若更换数据源，需要改动哪些类的代码？



实验步骤

5 绘制数据访问对象模式结构图

数据访问对象模式（Data Access Object Pattern）也叫做 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

关键代码：分离低级的数据访问操作和高级的业务服务

实验步骤

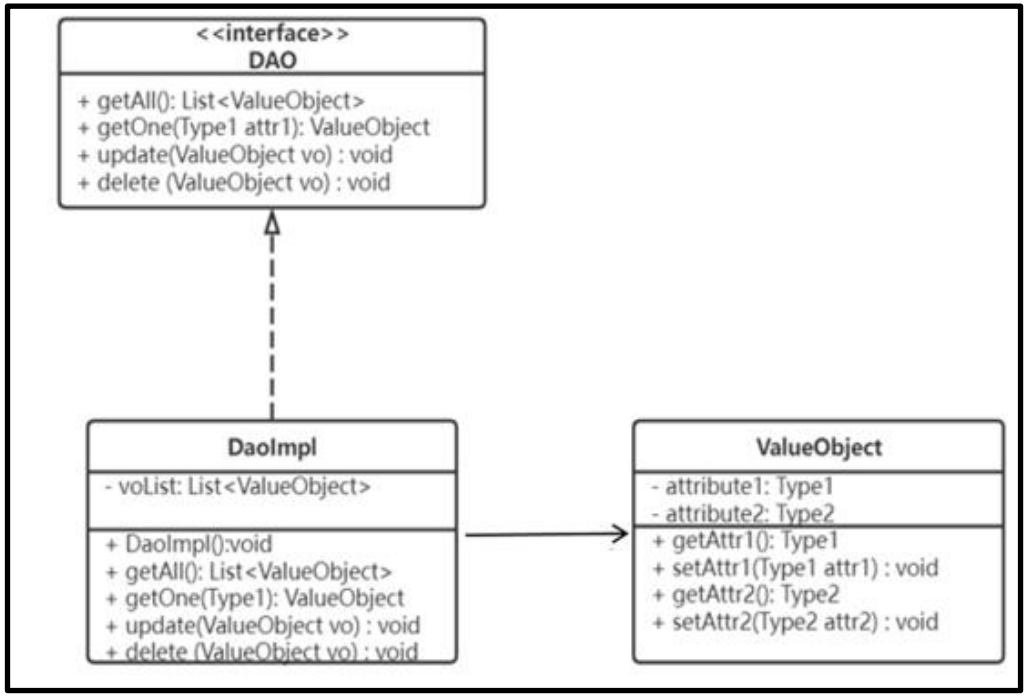
5 绘制数据访问对象模式类图

假如我们实现一个图书管理系统，实现查询所有图书、按编号查询、增加、删除图书的功能。我们该如何绘制UML类图?

实验步骤

5 绘制数据访问对象模式类图

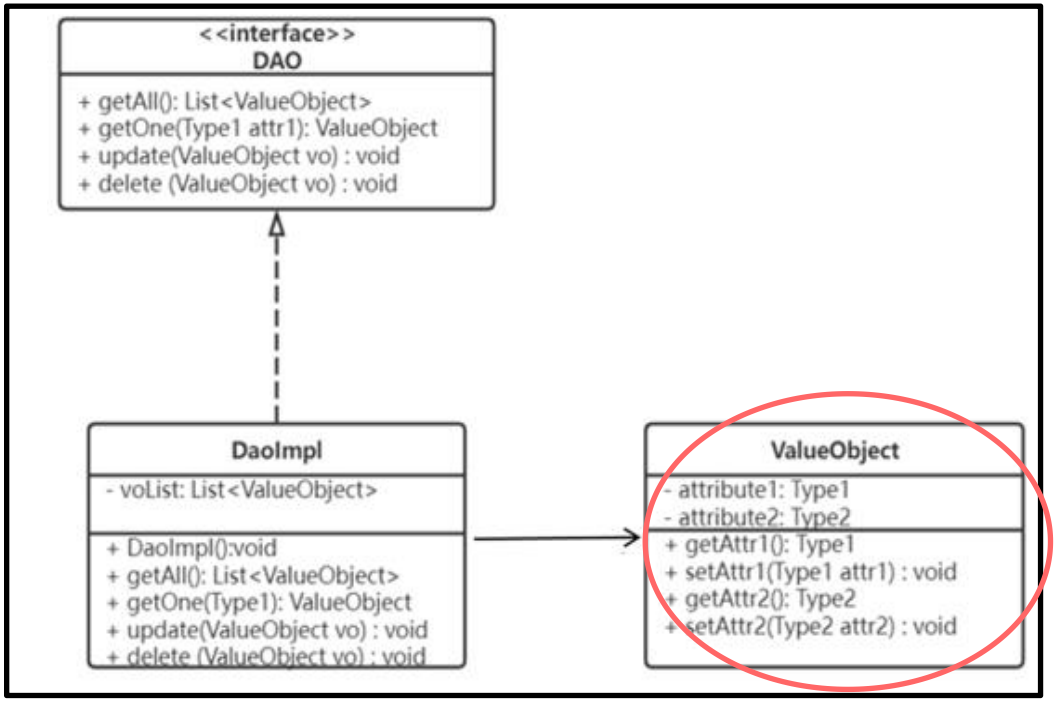
假如我们实现一个图书管理系统，实现查询所有图书、按编号查询、增加、删除图书的功能。我们该如何绘制UML类图?

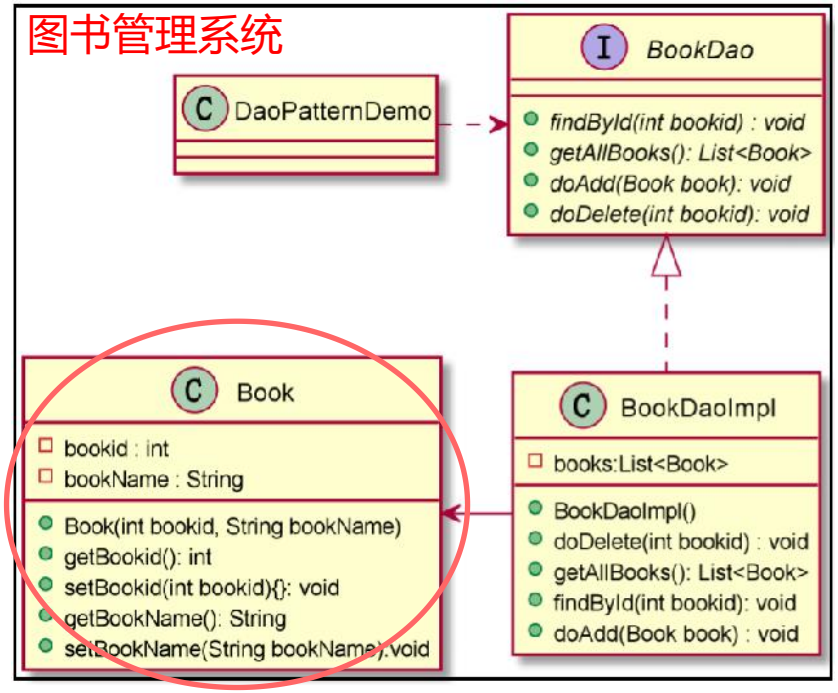


实验步骤

5 绘制数据访问对象模式类图

① 创建一个作为数值对象的实体类Book 。

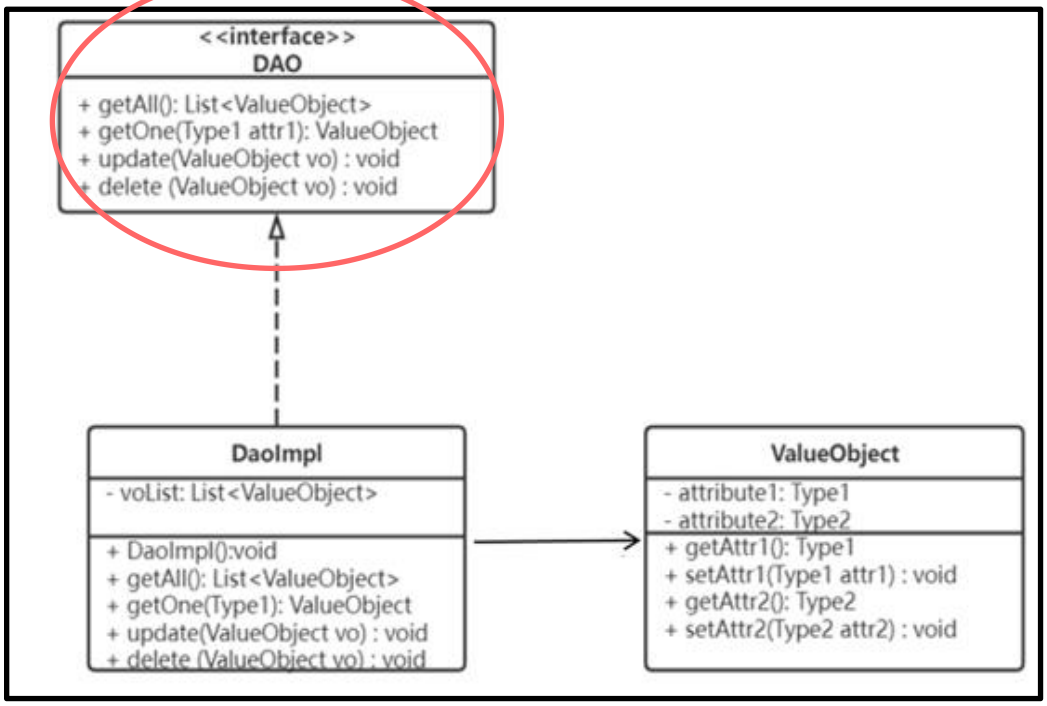


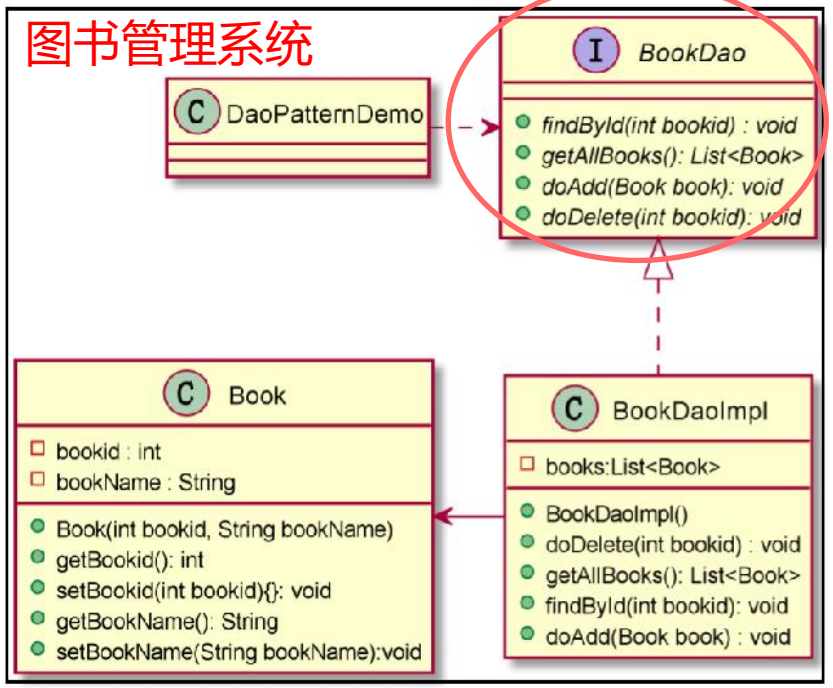


实验步骤

5 绘制数据访问对象模式类图

② 创建数据访问对象接口BookDao，提供对Book对象的标准操作。

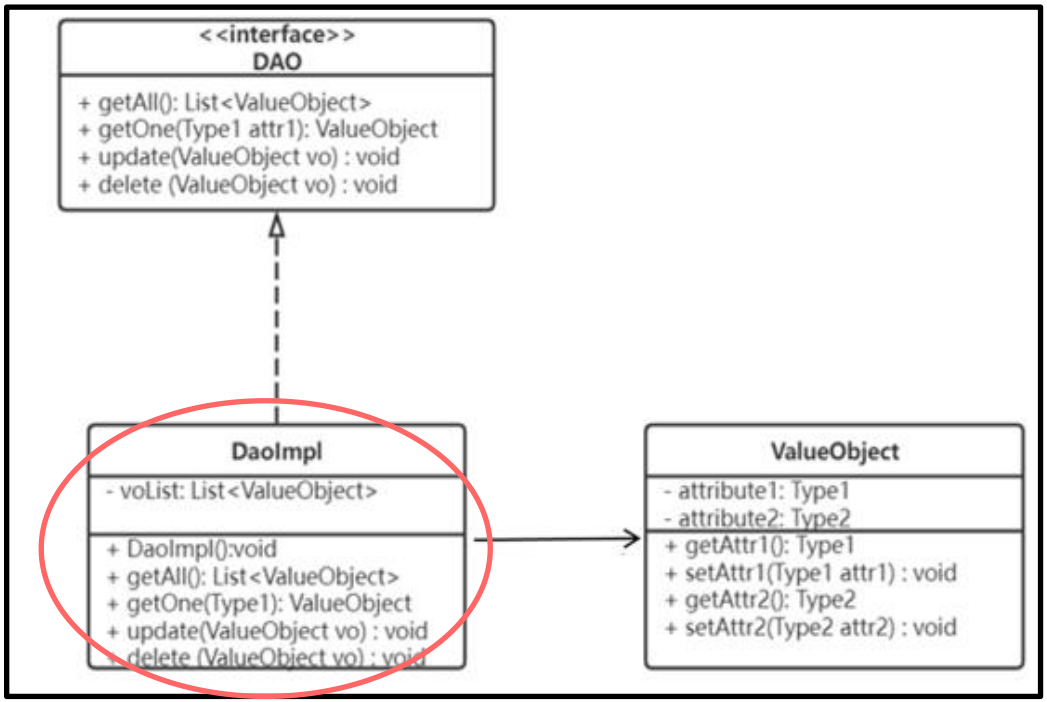


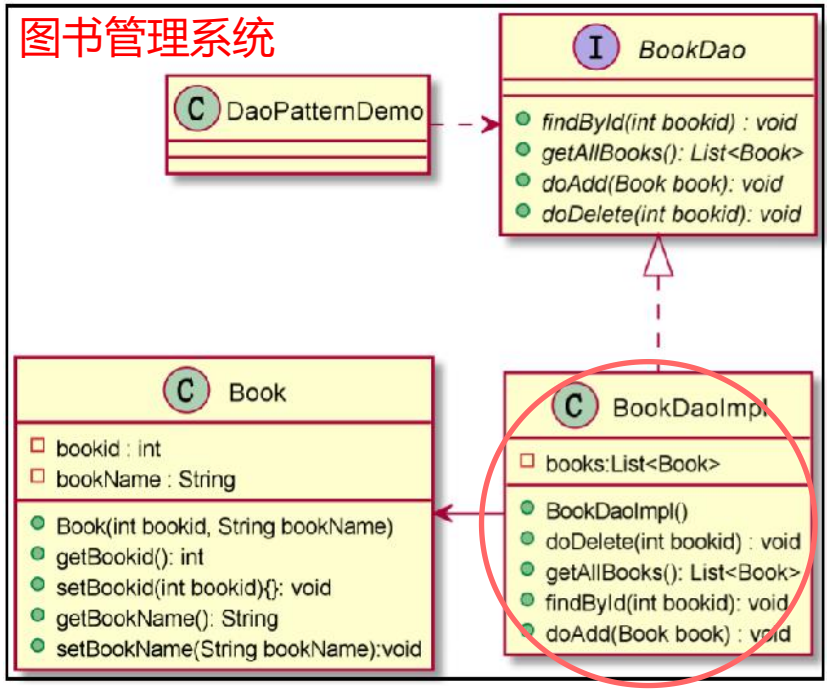


实验步骤

5 绘制数据访问对象模式类图

③ 创建实现了数据访问对象接口的实体类BookDaoImpl 。

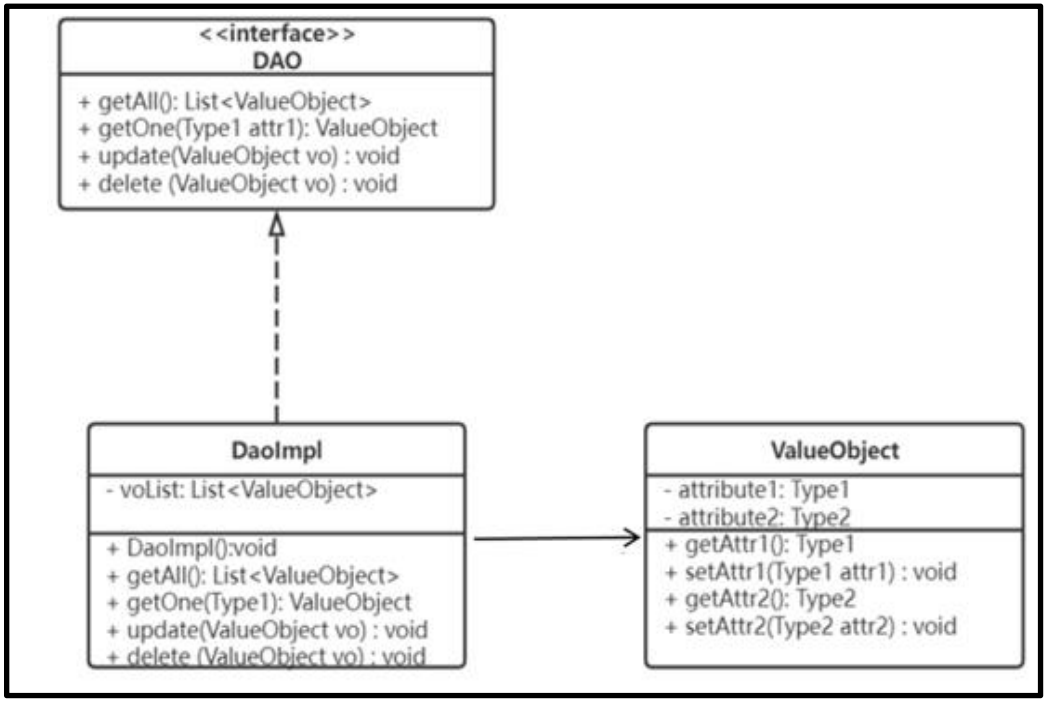


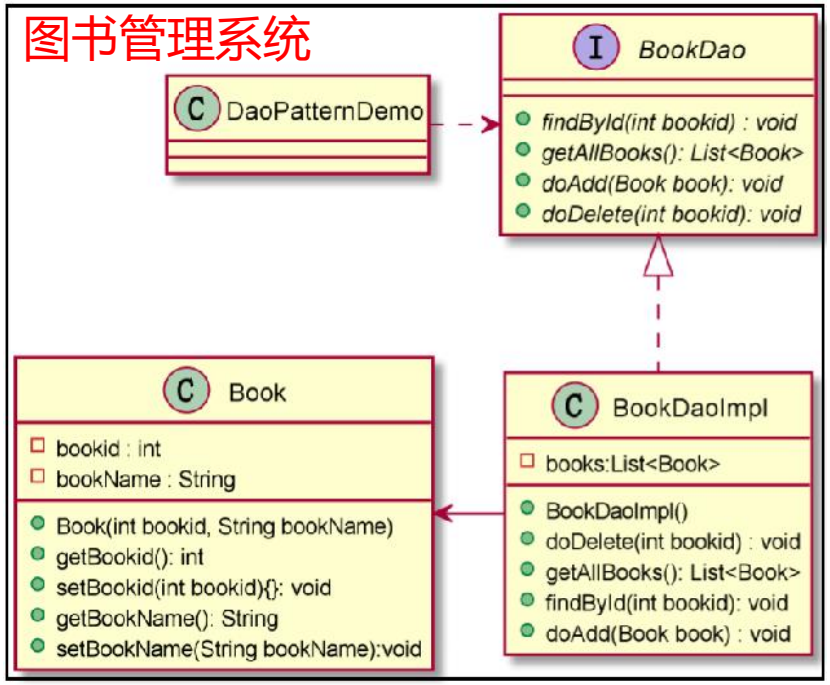


实验步骤

5 绘制数据访问对象模式类图

请思考：结合排行榜，数值对象实体类需要哪些属性？DAO接口需要提供哪些方法？

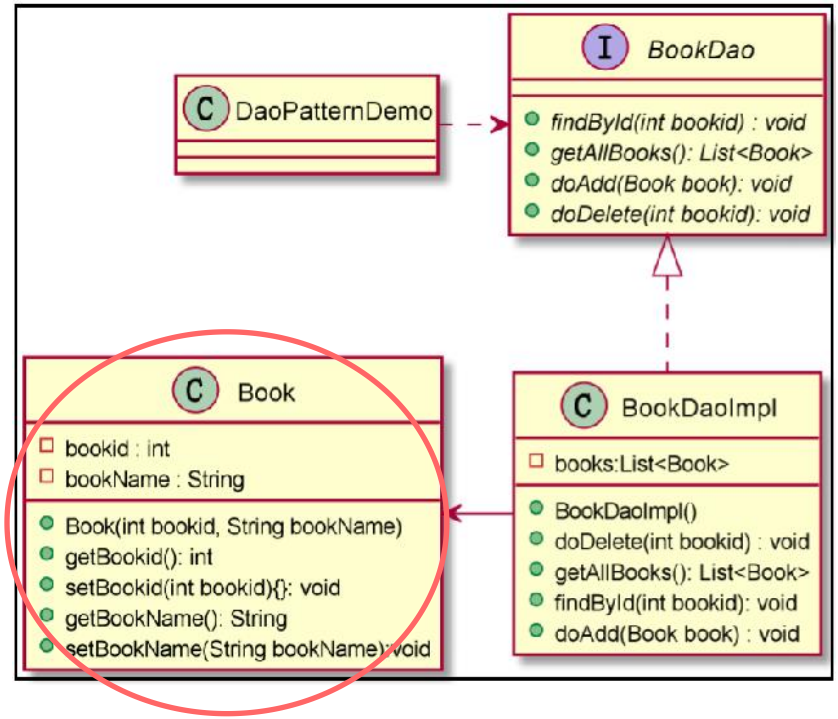




实验步骤

6 重构代码，实现数据访问对象模式

- 数据访问对象模式代码示例（图书管理）：



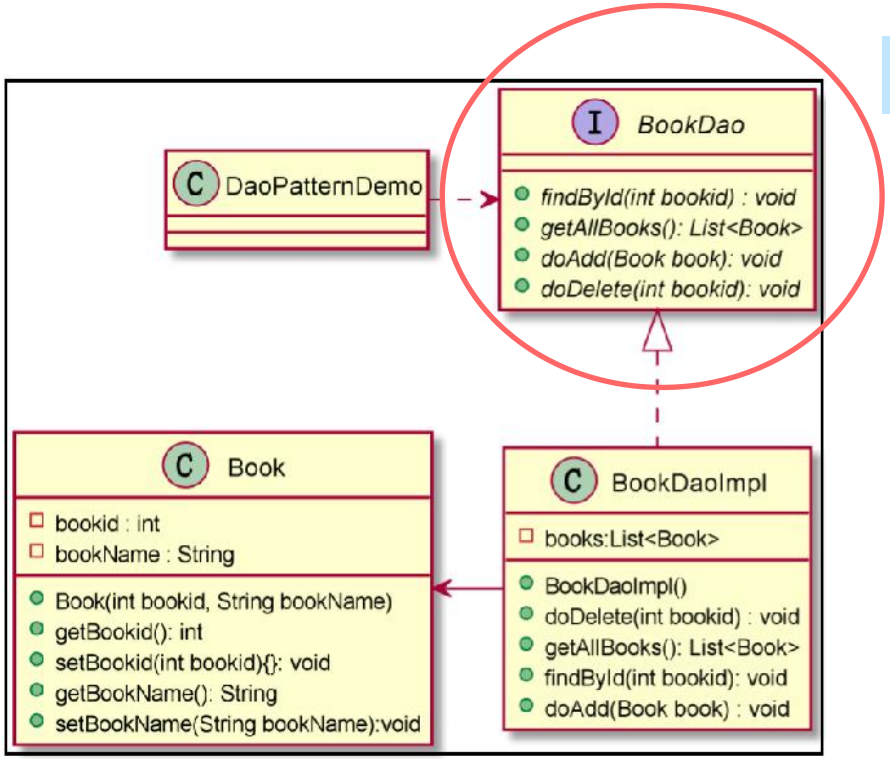
① 创建一个数值对象Book实体类；

```
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;
    }
}
```

实验步骤

6 重构代码，实现数据访问对象模式

- 数据访问对象模式代码示例（图书管理）：



② 创建数据访问对象DAO接口；

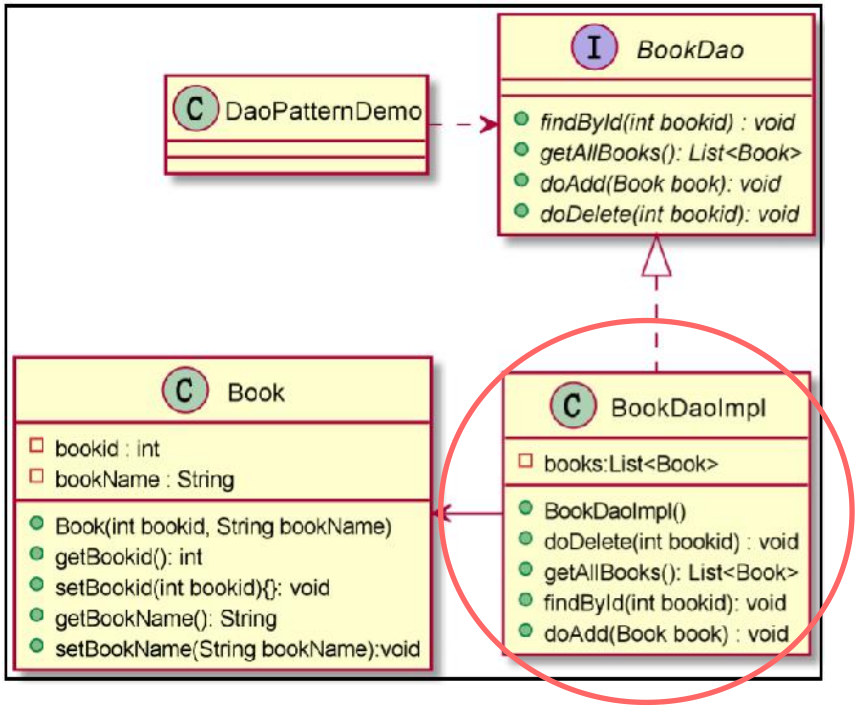
```
public interface BookDao {

    void findById(int bookid);

    List<Book> getAllBooks();

    void doAdd(Book book);

    void doDelete(int bookid);
}
```

实验步骤

6 重构代码，实现数据访问对象模式

- 数据访问对象模式代码示例（图书管理）：

3、创建实现了上述接口的DAO实现类；



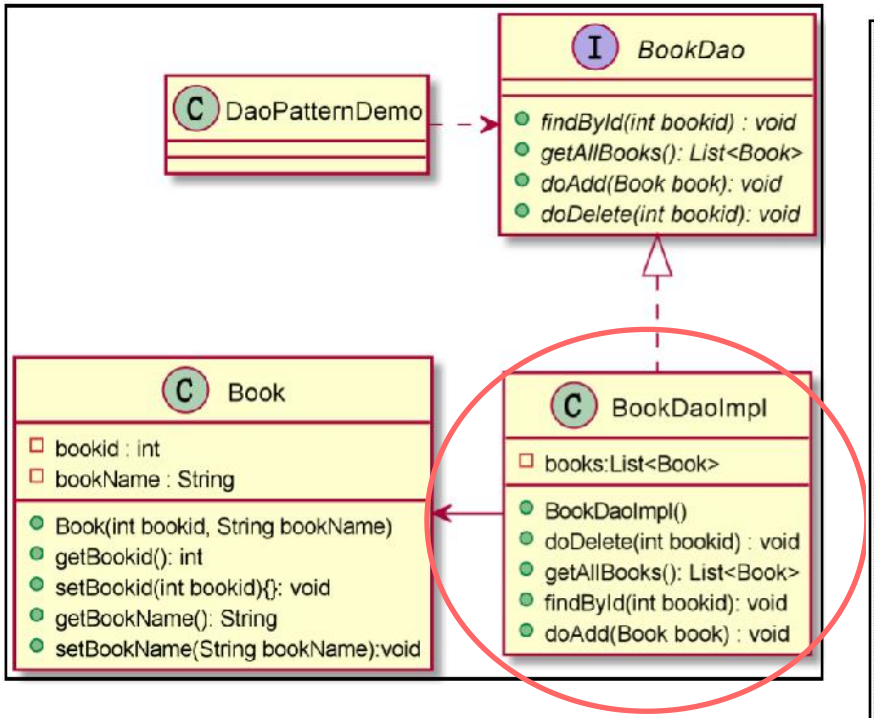

```java
public class BookDaoImpl implements BookDao {

    //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"));
    }

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

1

实验步骤

6 重构代码，实现数据访问对象模式

- 数据访问对象模式代码示例（图书管理）：



3、创建实现了上述接口的DAO实现类；

2

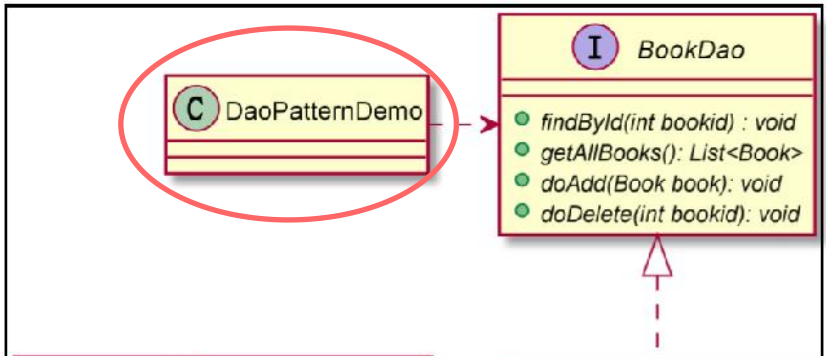
```
    //查找图书
    @Override
    public void findById(int bookid) {
        for (Book item : books) {
            if (item.getBookid() == bookid) {
                System.out.println("Find Book: ID [" + bookid +
                        "], Book Name [" + item.getBookName() + "]");
                return;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

    //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!");
    }

    //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() + "]");
    }
}
```

实验步骤

6 重构代码，实现数据访问对象模式

- 数据访问对象模式代码示例（图书管理）：

4、使用DaoPatternDemo来演示数据访问对象模式的用法。

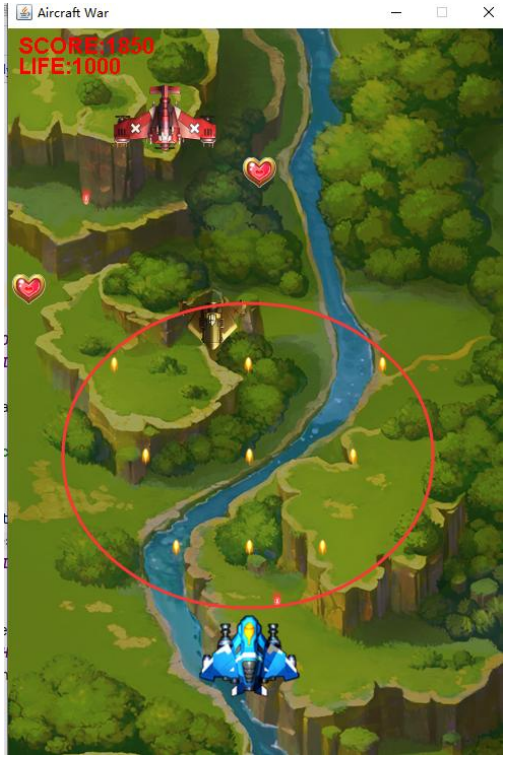



```
Book ID [1001], Book Name : [Clean Code]
Book ID [1002], Book Name : [Design Patterns]
Book ID [1003], Book Name : [Effective Java]

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]
```

```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() + "]");
        }

        //查找图书
        bookDao.findById(1002);

        //删除图书
        bookDao.doDelete(1002);

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

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

本次实验的目标

- ✓ 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。

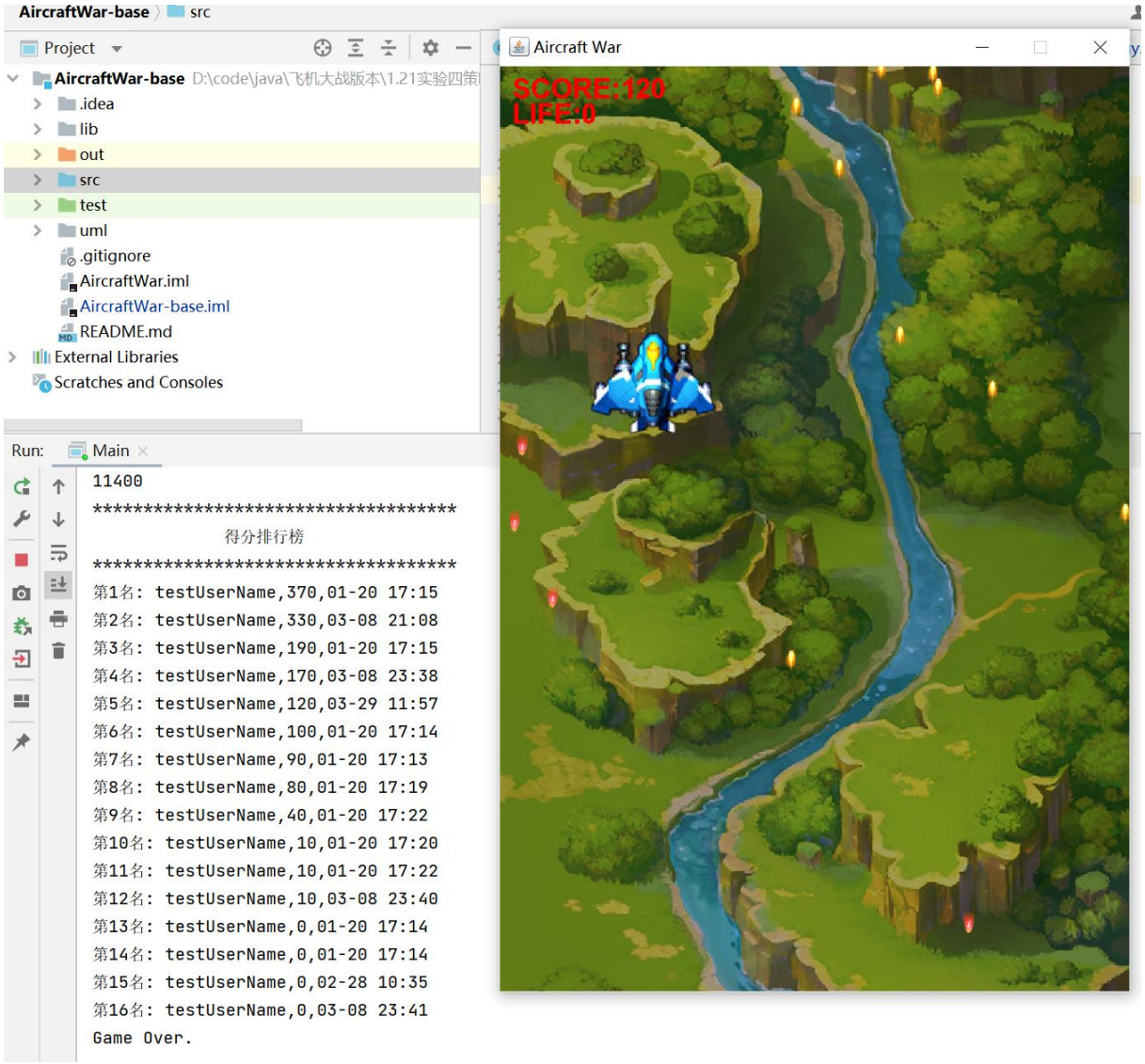


注意：

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。

本次实验的目标


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
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.
Aircraft War
SCORE:120
LIFE:0

作业提交

- 提交内容

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）

- 截止时间

实验课后一周内提交至HITsz Grader 作业提交平台，具体截止日期参考平台发布。

登录网址：： http://grader.tery.top:8000/#/login

同学们
请开始实验吧！