关于实验课

1. 使用腾讯课堂上课，如遇到技术故障将改用腾讯会议；
2. 为方便考勤，请同学们将昵称改成“学号-真实姓名”；
3. 上课不定时发起签到，请同学们不要迟到早退。

面向对象的软件构造导论

实验四：策略模式和数据访问对象模式

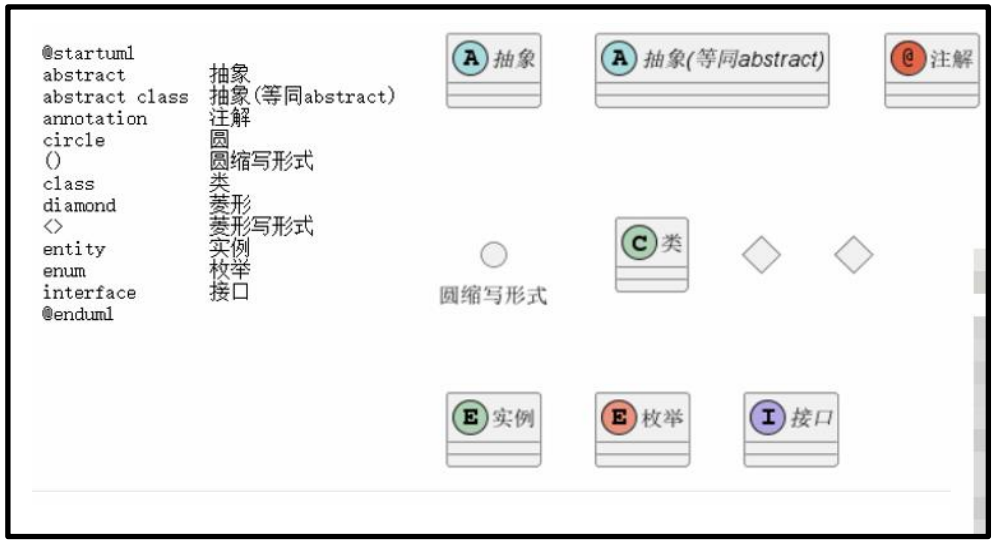
2022春

哈尔滨工业大学（深圳）

实验中存在的问题

- UML类图的规范性
  - 关系的图标

  - 接口、抽象类、具体类的图标



实验中存在的问题

- UML类图的规范性
- 职责分配的合理性（道具掉落、道具生效）

实验中存在的问题

- 下面2段来自Game的代码，完成的都是初始化英雄机的工作，哪个更好一些?

A

```
// 初始化英雄机
heroAircraft = HeroAircraft.getHeroAircraft(
        locationX: WINDOW_WIDTH / 2,
        locationY: WINDOW_HEIGHT - ImageManager.HERO_IMAGE.getHeight() ,
        speedX: 0,  speedY: 0,  hp: 1000);
```

B

```
// 初始化英雄机
heroAircraft = HeroAircraft.getHeroAircraft();
```

实验中存在的问题

- UML类图的规范性
- 职责分配的合理性（道具掉落、道具生效）
- 创建细节的封装（英雄机、敌机、道具）

实验中存在的问题

- UML类图的规范性
- 职责分配的合理性（道具掉落、道具生效）
- 创建细节的封装（英雄机、敌机、道具）
- 多态的使用（敌机工厂、道具工厂）

实验中存在的问题

Game类

```
/**
 * 敌机工厂
 */
protected EnemyFactory enemyFactory;

/**
 * 产生小敌机和精英敌机。
 *
 * @return 产生的敌机，可以为空链表（即表示不产生）；不能返回null！
 */
protected List<AbstractEnemyAircraft>  produceEnemy() {
    List<AbstractEnemyAircraft> res = new LinkedList<>();
    // 产生敌机产生
    if (enemyAircrafts.size() < enemyMaxNumber) {
        if (random.nextDouble() < 0.5) {
            //普通敌机
            enemyFactory = new MobFactory();
        }
        else{
            //精英机
            enemyFactory = new EliteFactory();
        }
        res.add(enemyFactory.createEnemyAircraft());
    }
    return res;
}
```

实验中存在的问题

Game类

```
if (enemyAircraft.crash(bullet)) {
    // 敌机撞击到英雄机子弹
    // 敌机损失一定生命值
    enemyAircraft.decreaseHp(bullet.getPower());
    bullet.vanish();
    if (enemyAircraft.notValid()) {
        //获得分数，产生道具补给
        score += enemyAircraft.score();
        flyingSupplies.addAll(enemyAircraft.generateSupplies());

    }
}
```

实验中存在的问题

Game类

```
/**
 * 敌机工厂
 */
protected MobFactory mobFactory;
protected EliteFactory eliteFactory;
protected BossFactory bossFactory;
/**
 * 产生小敌机和精英敌机。
 *
 * @return 产生的敌机，可以为空链表（即表示不产生）；不能返回null！
 */
protected List<AbstractEnemyAircraft>  produceEnemy() {
    List<AbstractEnemyAircraft> res = new LinkedList<>();

    // 产生敌机产生
    if (enemyAircrafts.size() < enemyMaxNumber) {
        if (random.nextDouble() < 0.5) {
            //普通敌机
            mobFactory = new MobFactory();
            res.add(mobFactory.createEnemyAircraft());
        }
        else{
            //精英机
            eliteFactory = new EliteFactory();
            res.add(eliteFactory.createEnemyAircraft());
        }
    }
    return res;
}
```

几个注意

- 注意绘制规范的UML类图；
- 注意每个类职责分配的合理性；
- 注意对细节的封装；
- 注意多态的使用。

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	**4**	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

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

- 深入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；
- 掌握绘制策略模式和数据访问对象模式的UML类图；
- 熟练使用代码实现策略模式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得分排行榜 。

实验步骤

1 子弹发射应用场景分析

在飞机大战游戏中，英雄机直射，碰到火力道具时改为散射；精英敌机直射；Boss敌机散射。

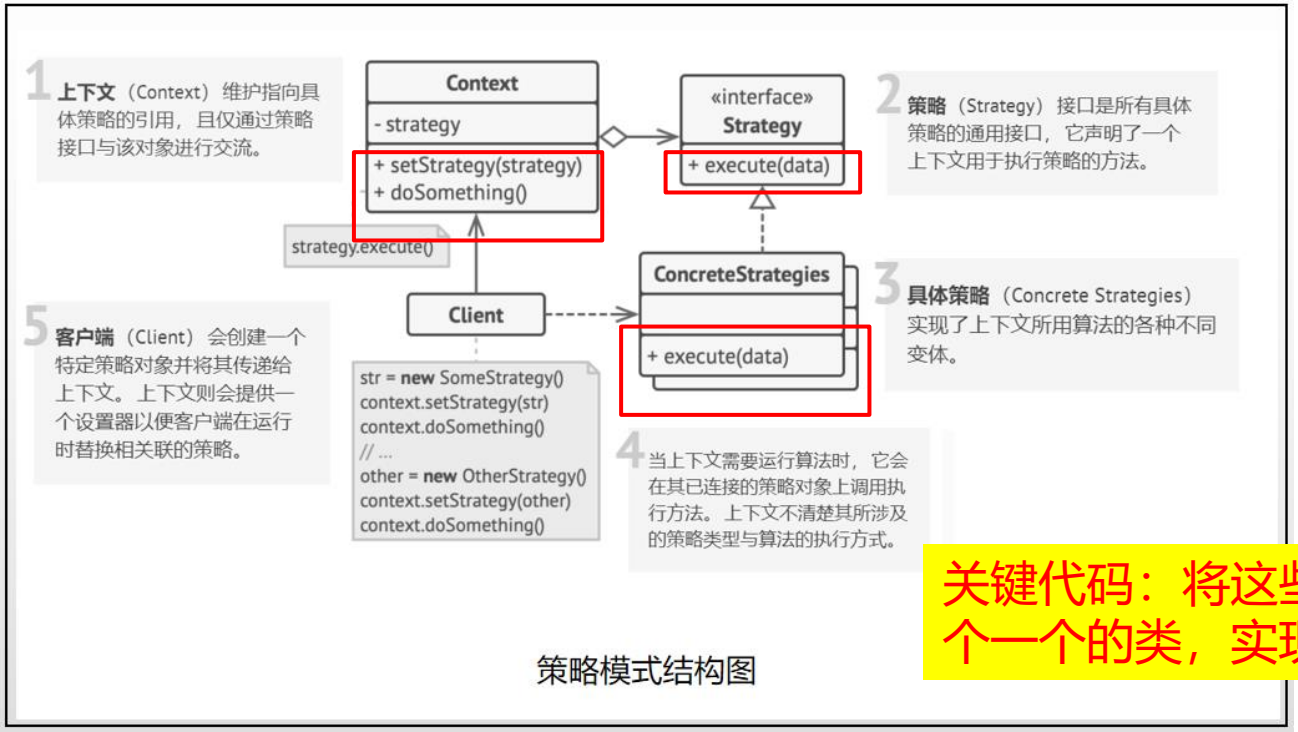
注意：直射只有纵向速度，散射有纵向速度和横向速度。

策略模式

实验步骤

2 绘制策略模式类图

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。



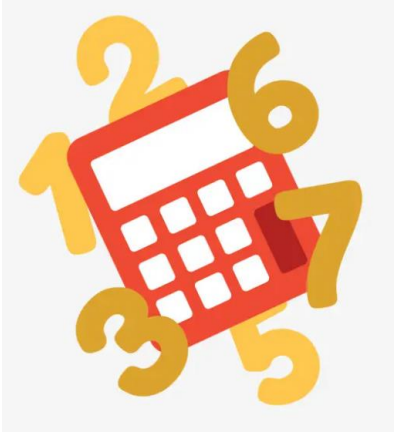
策略模式结构图

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

假如我们要用策略模式实现一个计算器，完成加、减、乘、除功能。我们该如何绘制UML类图？



当你很茫然的时候，你会做什么呢？

A、举头望明月

B、低头思故乡

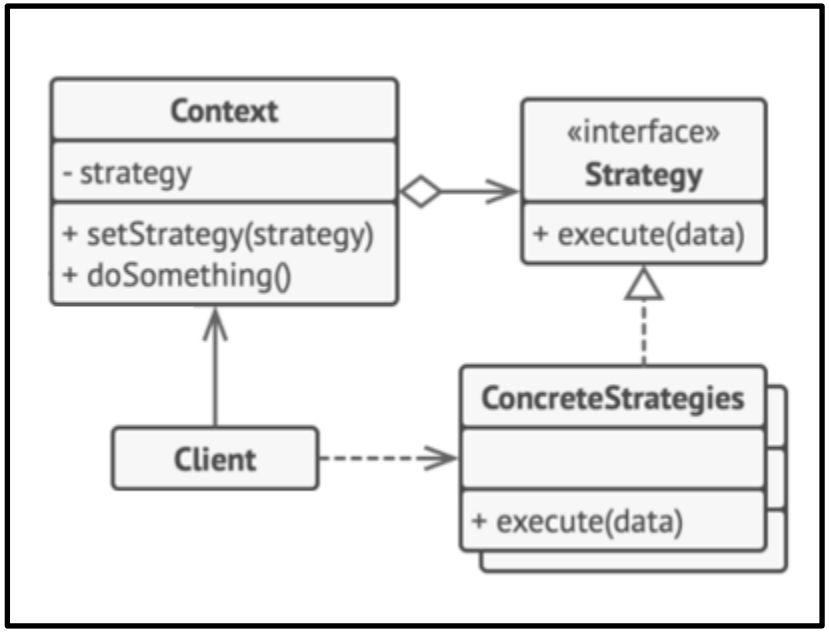
C、躺平

D、搬出策略模式的模板类图

实验步骤

2 绘制策略模式类图

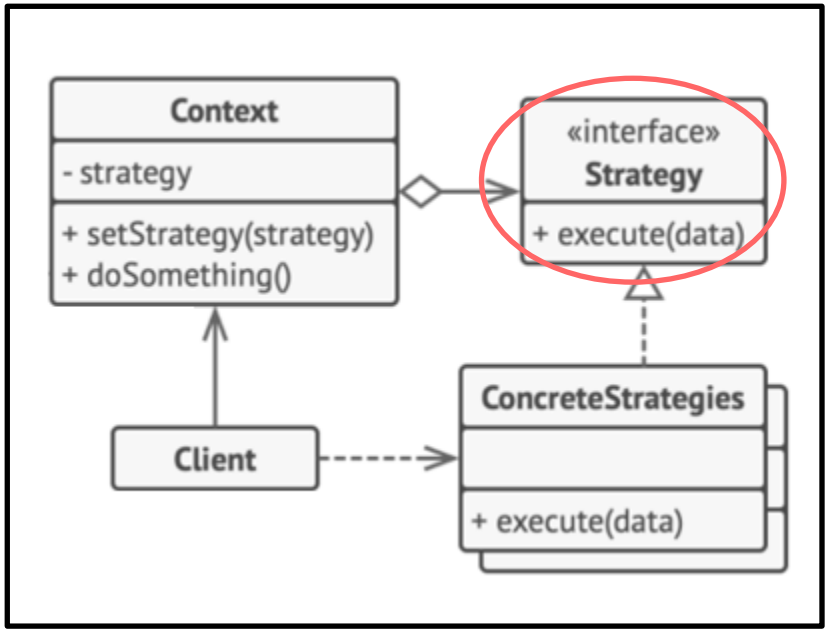
假如我们要用策略模式实现一个计算器，完成加、减、乘、除功能。我们该如何绘制UML类图？

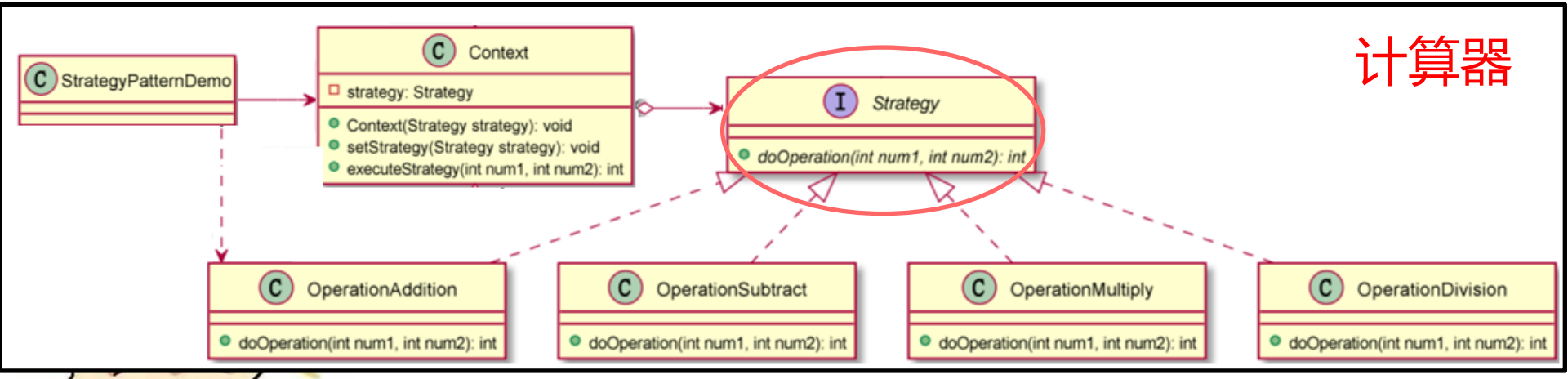


实验步骤

2 绘制策略模式类图

1、创建一个Strategy接口；

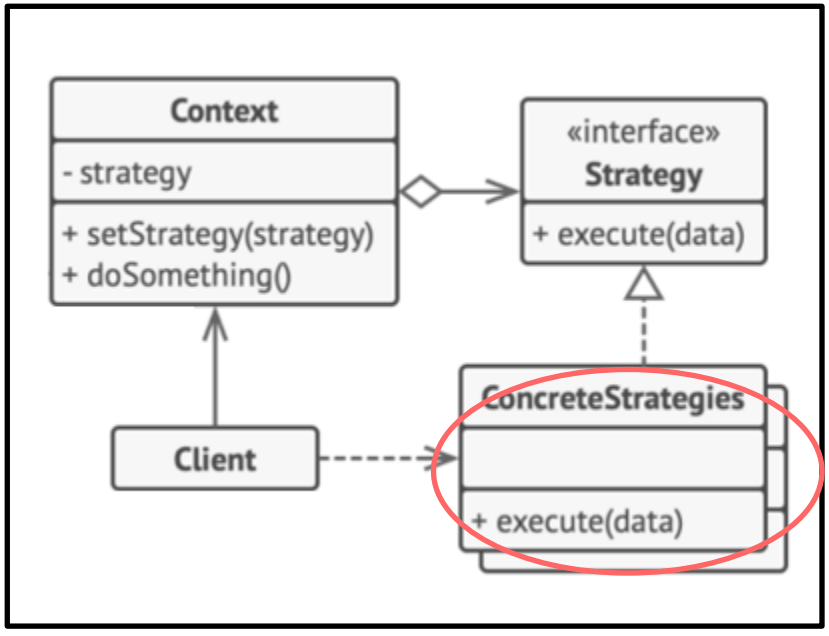


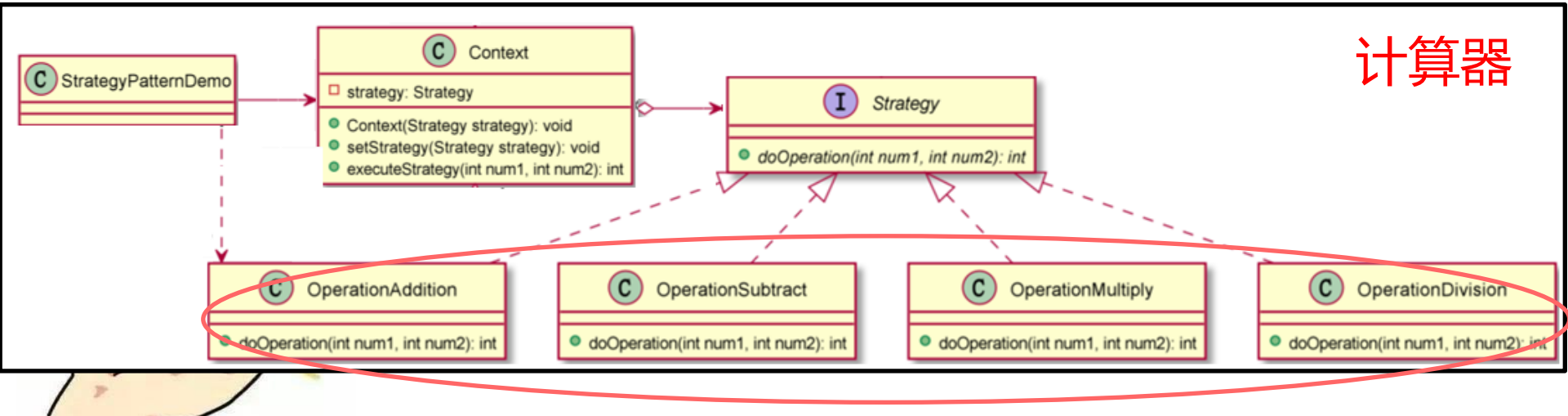


实验步骤

2 绘制策略模式类图

2、创建实现Strategy接口的实体策略类；

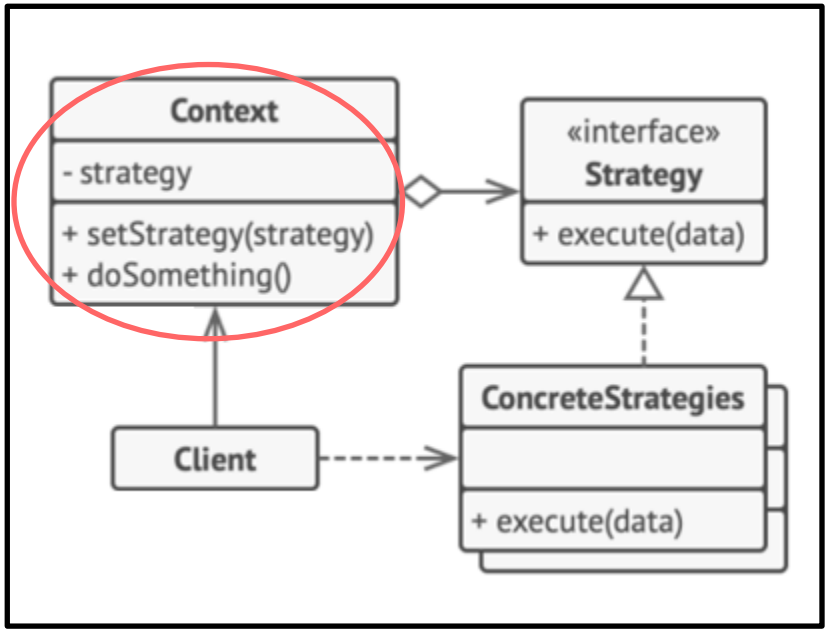


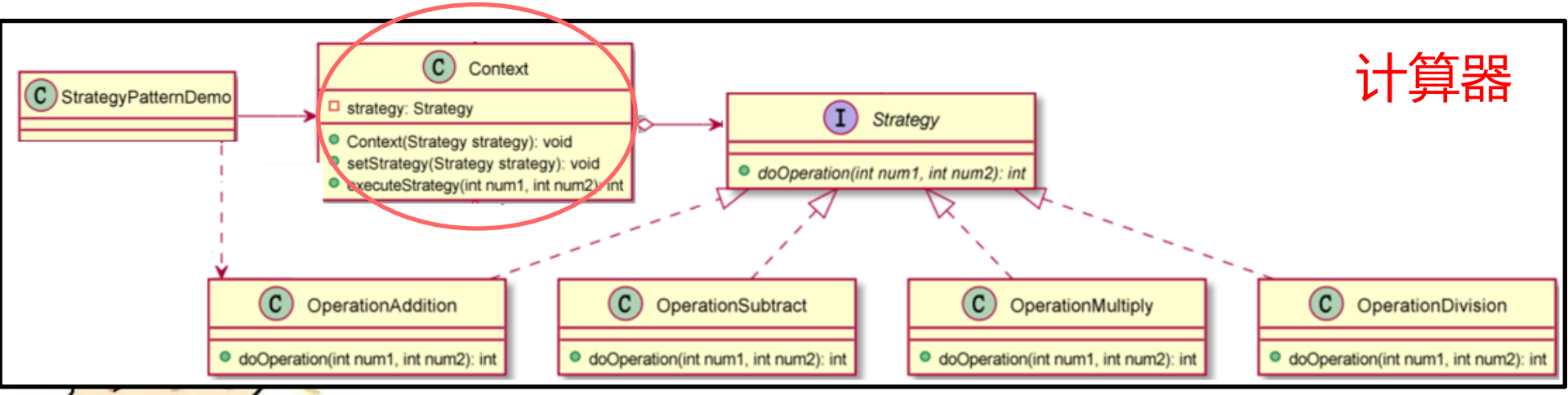


实验步骤

2 绘制策略模式类图

3、Context是一个使用了策略接口的类；

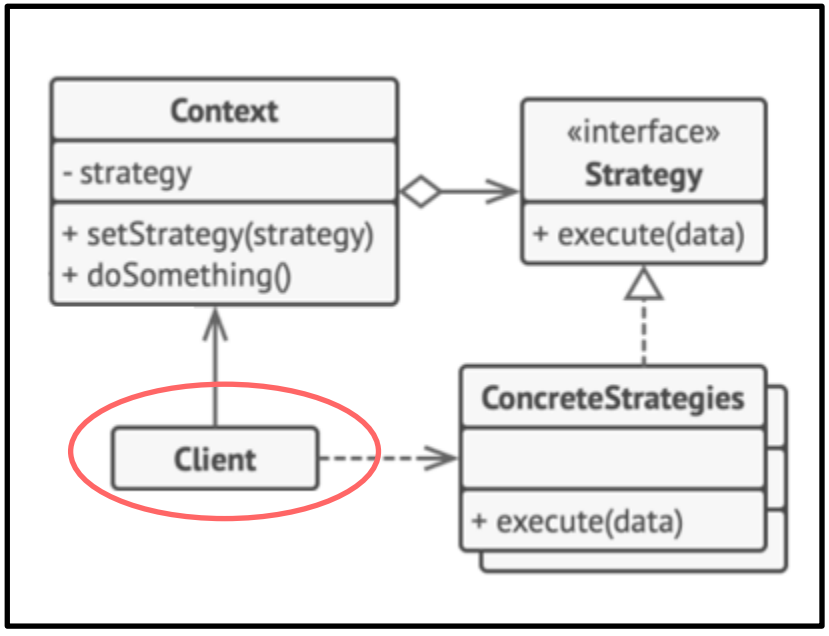


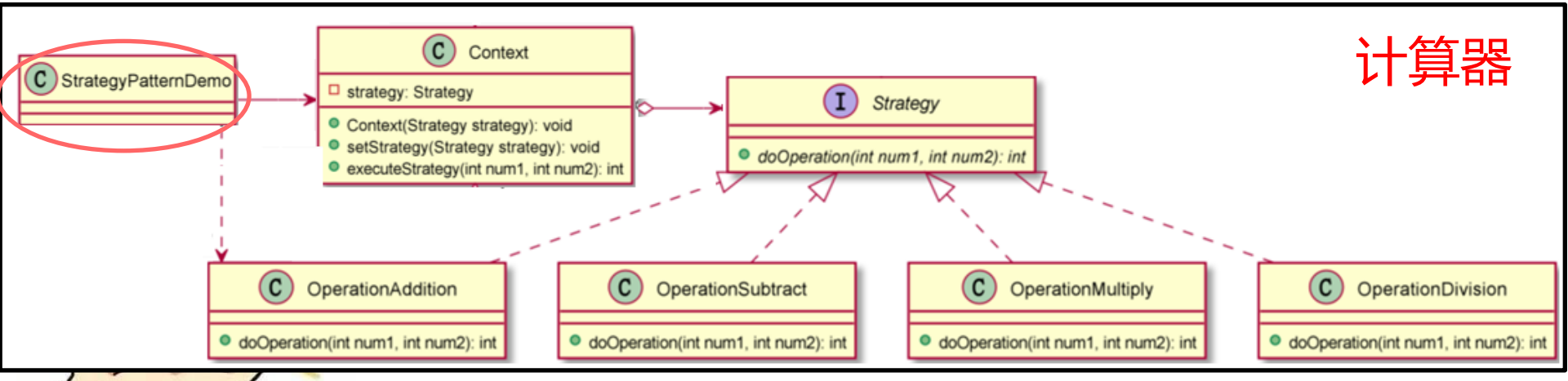


实验步骤

2 绘制策略模式类图

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

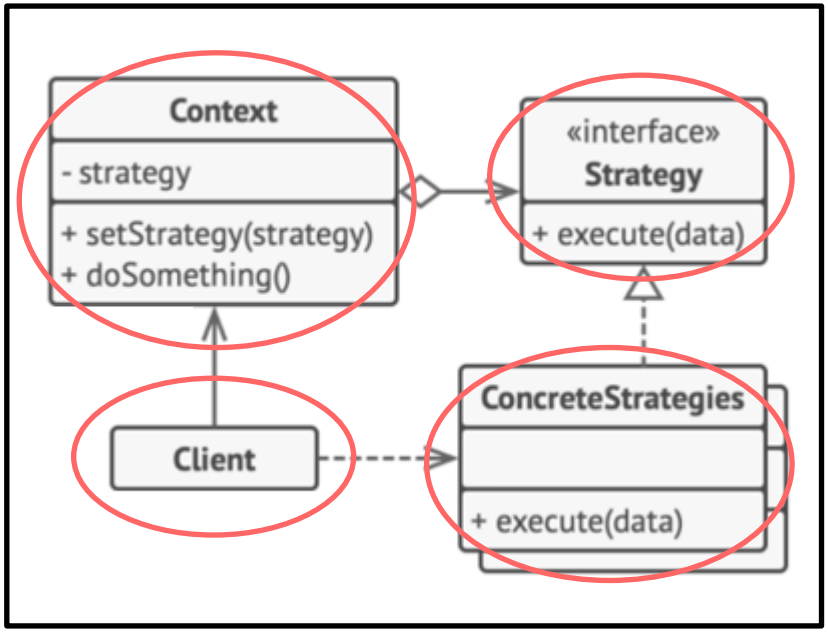




实验步骤

2 绘制策略模式类图

下面同学们想一下，结合我们飞机大战，我们要设计不同的射击策略，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。

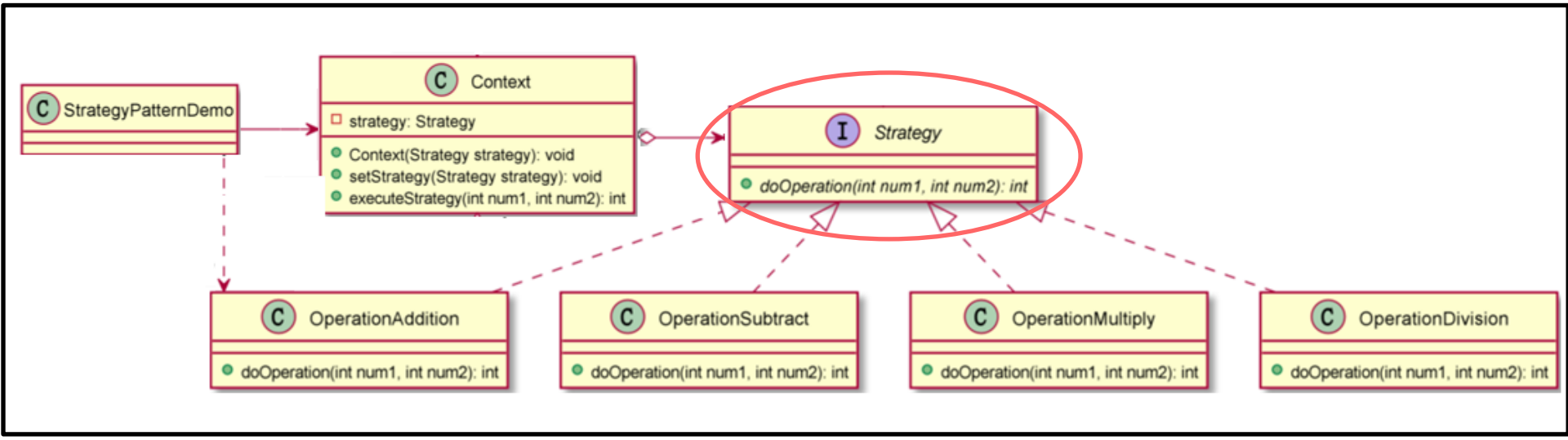
实验步骤

3 重构代码，实现策略模式

- 策略模式代码示例（计算器）

① 创建一个策略接口；

```
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）

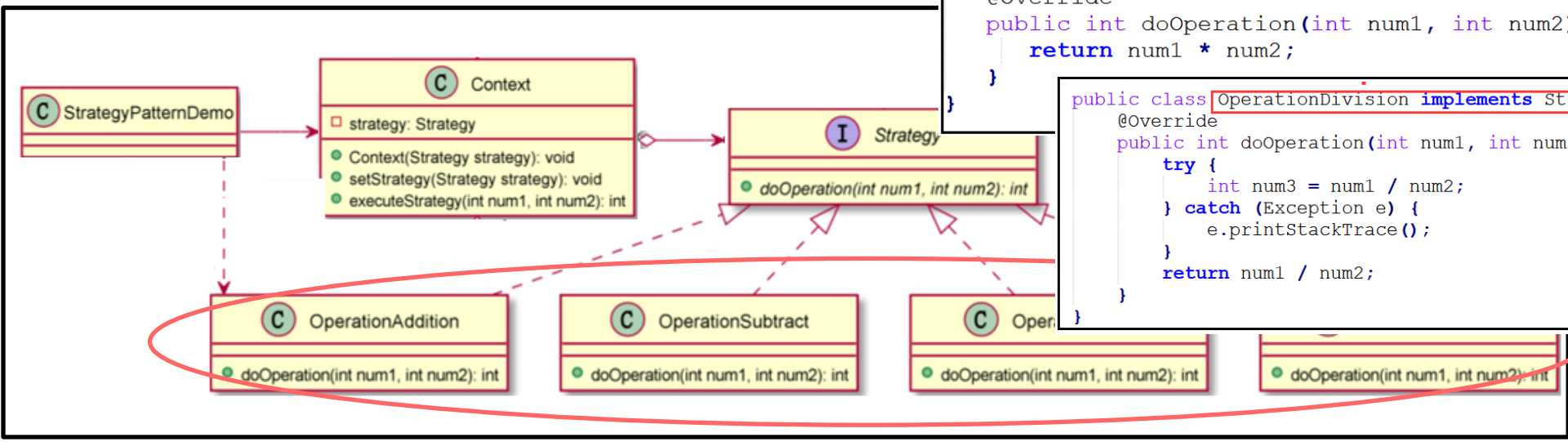
② 创建实现接口的实体类，充当具体策略角色；

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

```
public class OperationSubtract implements Strategy{

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

```
public class OperationMultiply implements St

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

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



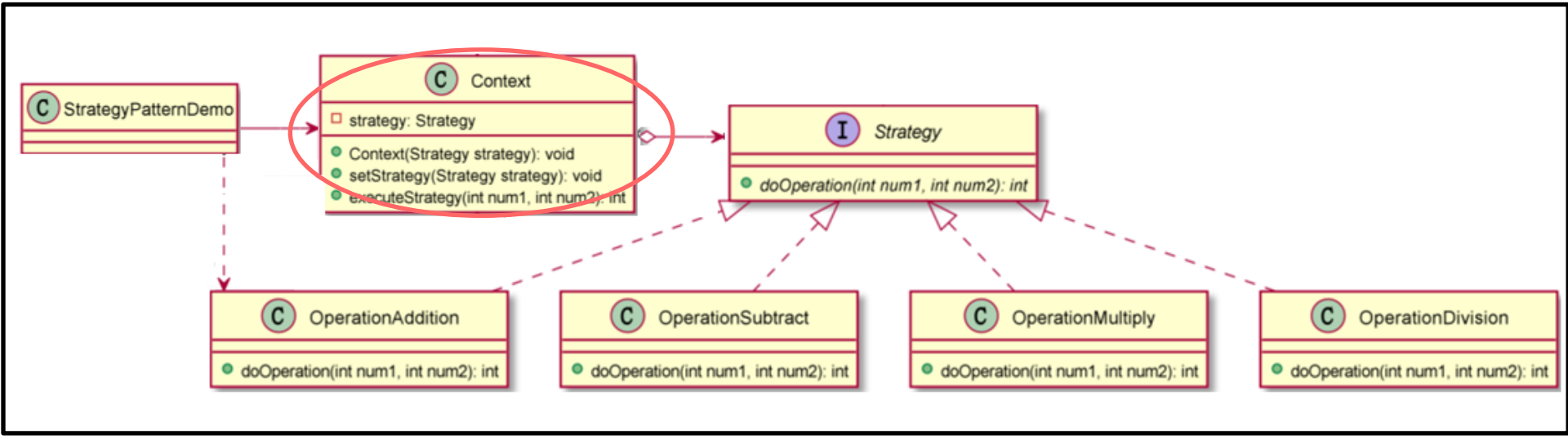
实验步骤

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

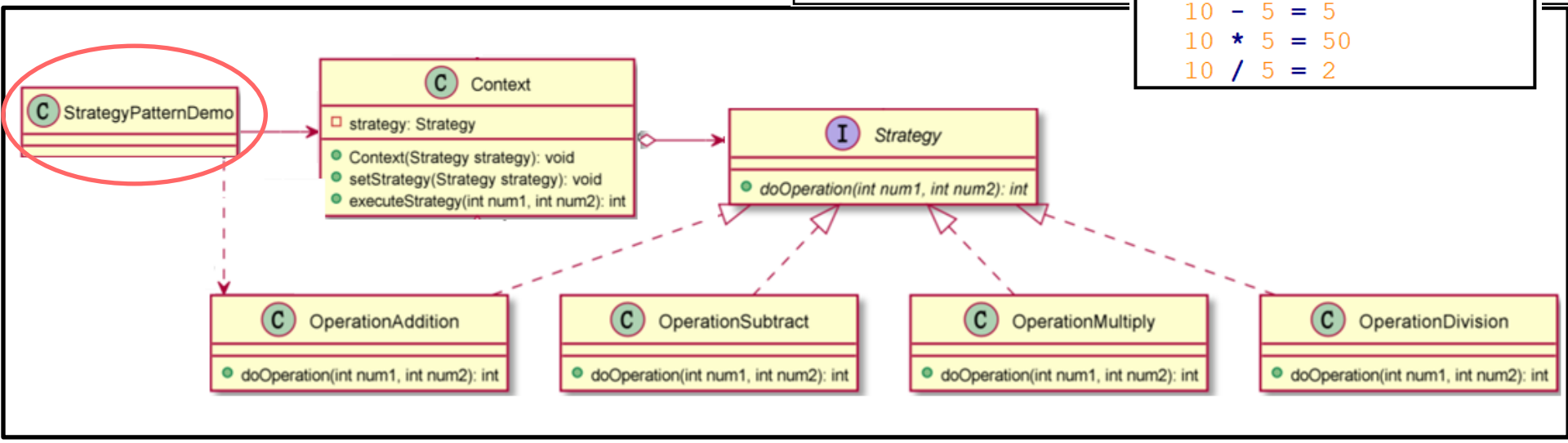
3 重构代码，实现策略模式

- 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));
    }
}
```

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



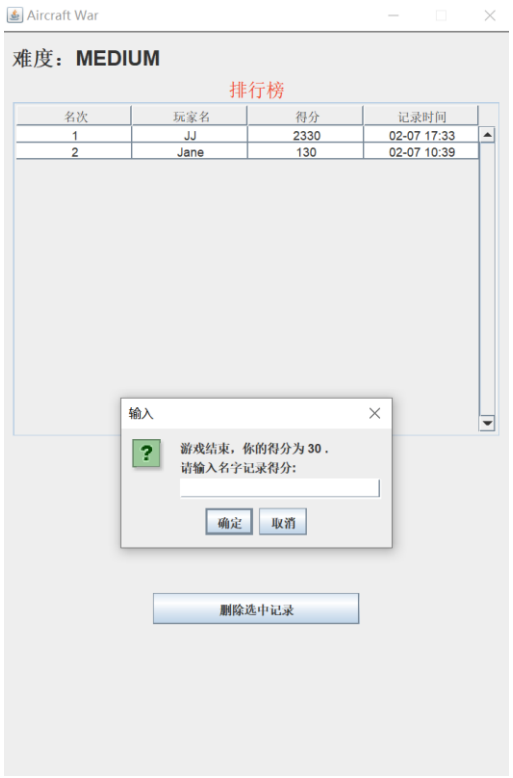
实验步骤

4 排行榜应用场景分析

游戏结束后，显示该难度的得分排行榜。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。

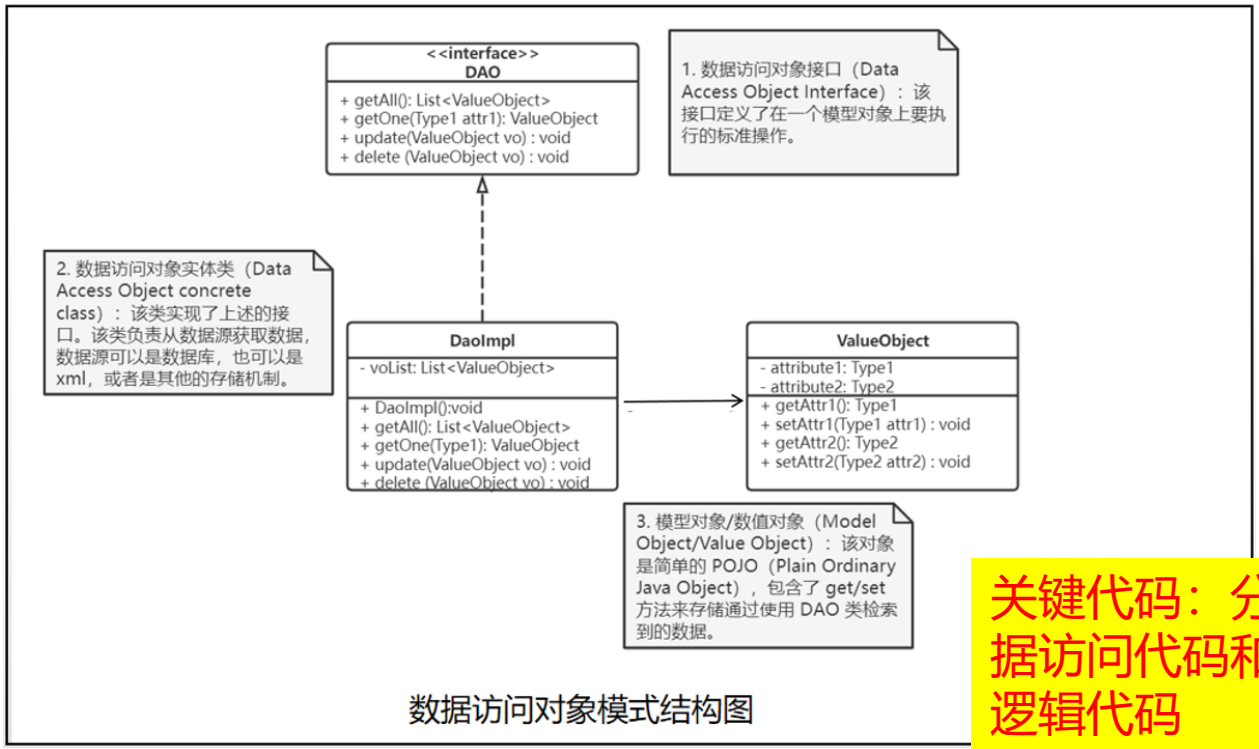
数据访问对象模式



实验步骤

5 绘制数据访问对象模式类图

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。



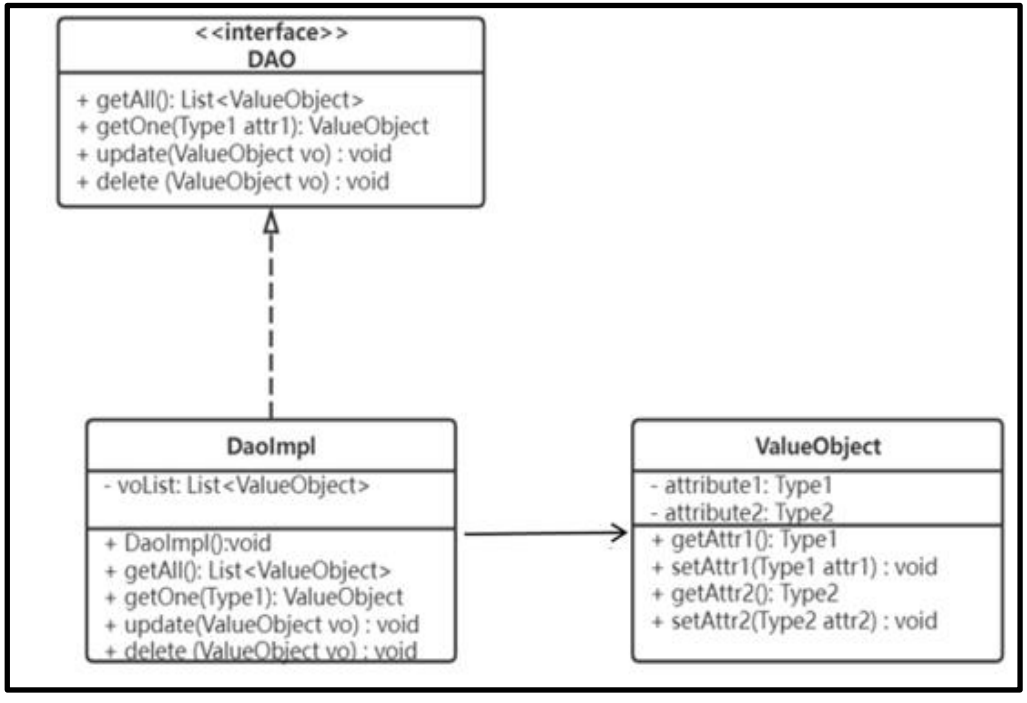
数据访问对象模式结构图

关键代码：分离低层的数据访问代码和高层的业务逻辑代码

实验步骤

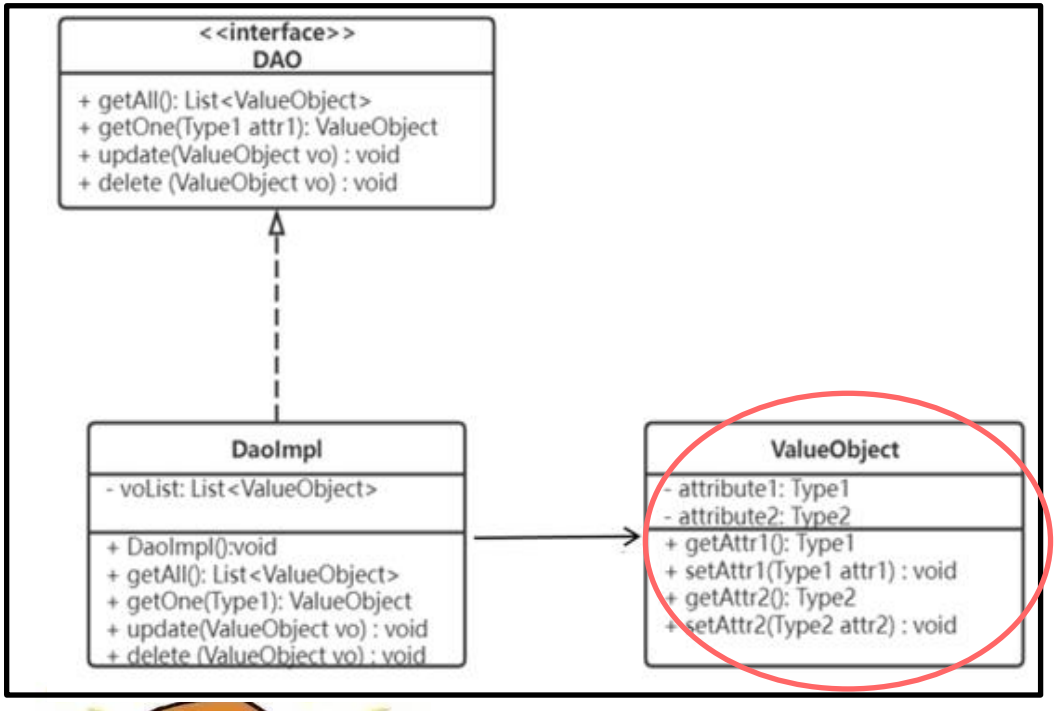
5 绘制数据访问对象模式类图

假如我们要用数据访问对象模式实现一个图书管理系统，实现查询所有图书、按编号查询图书、增加图书、删除图书的功能。我们该如何绘制UML类图？

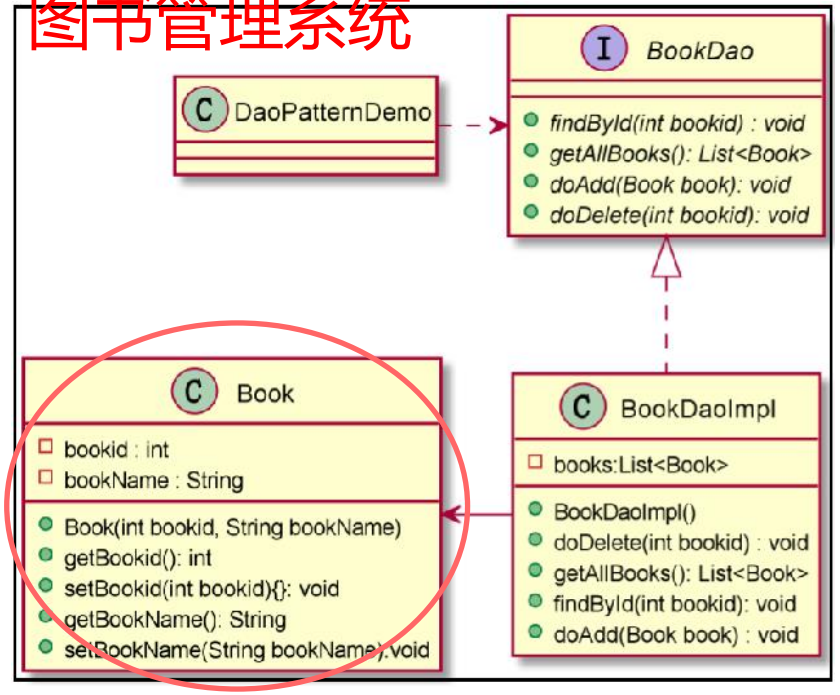


实验步骤

5 绘制数据访问对象模式类图

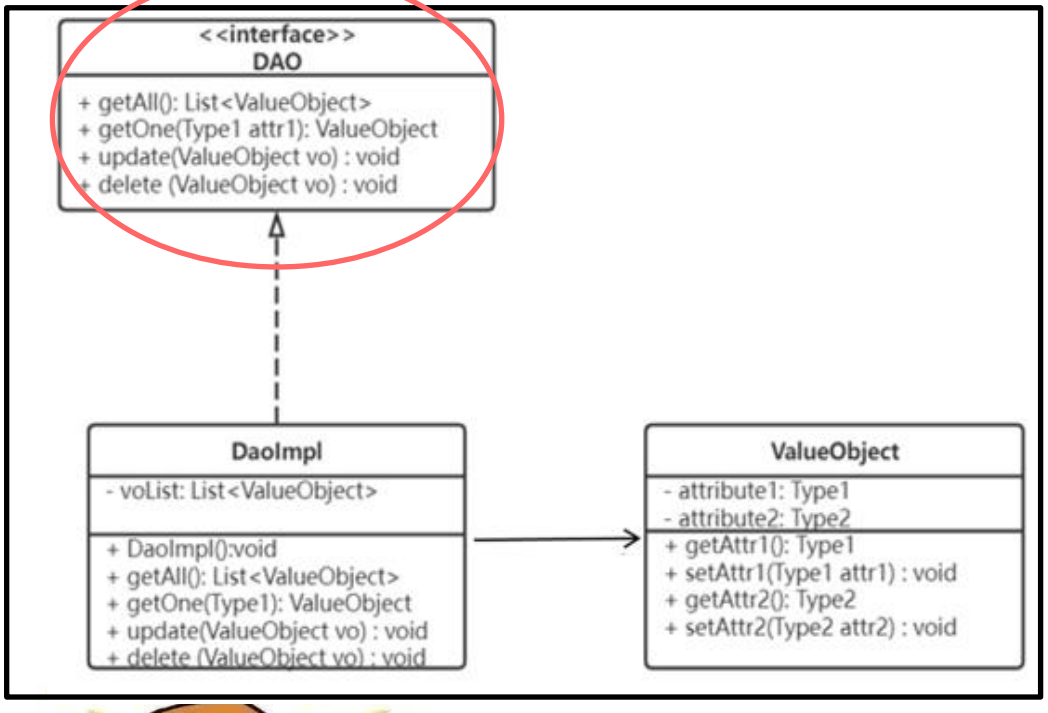


图书管理系统

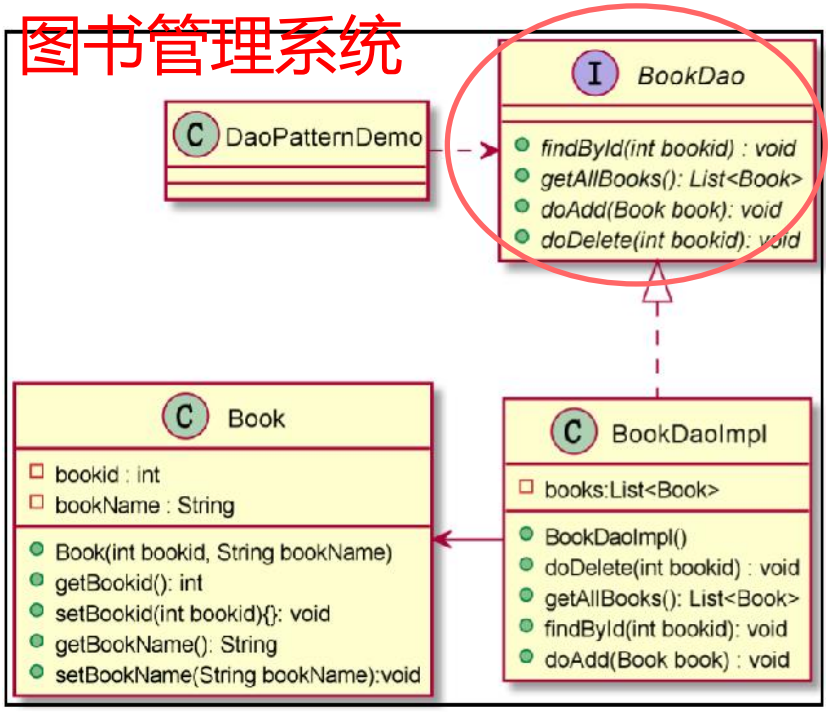


实验步骤

5 绘制数据访问对象模式类图

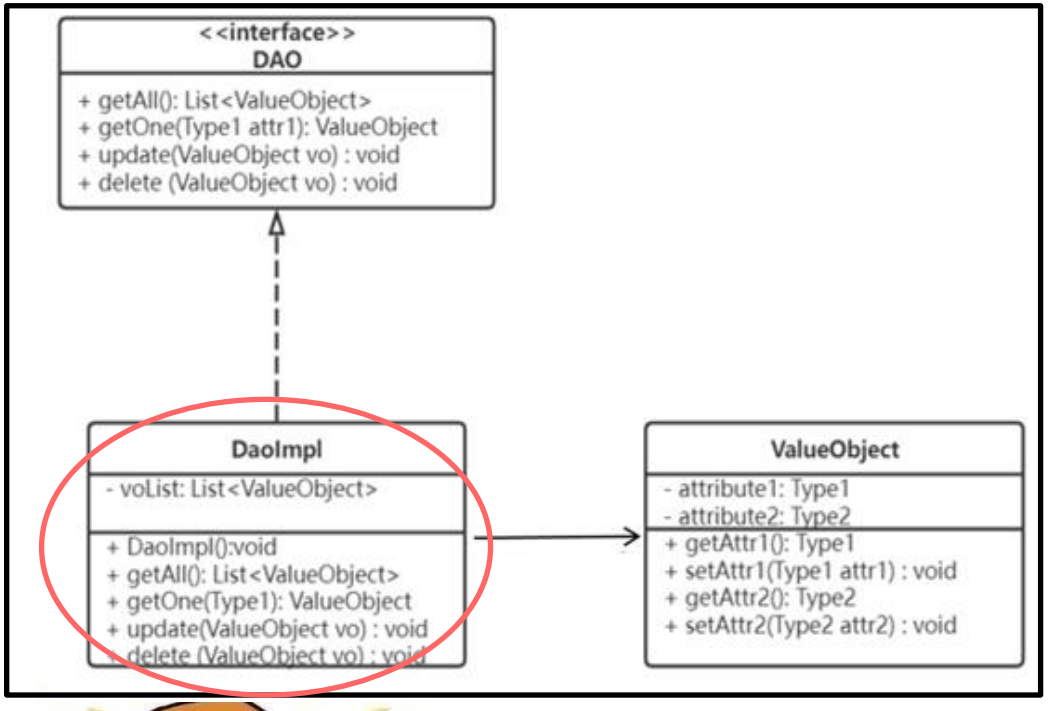


图书管理系统

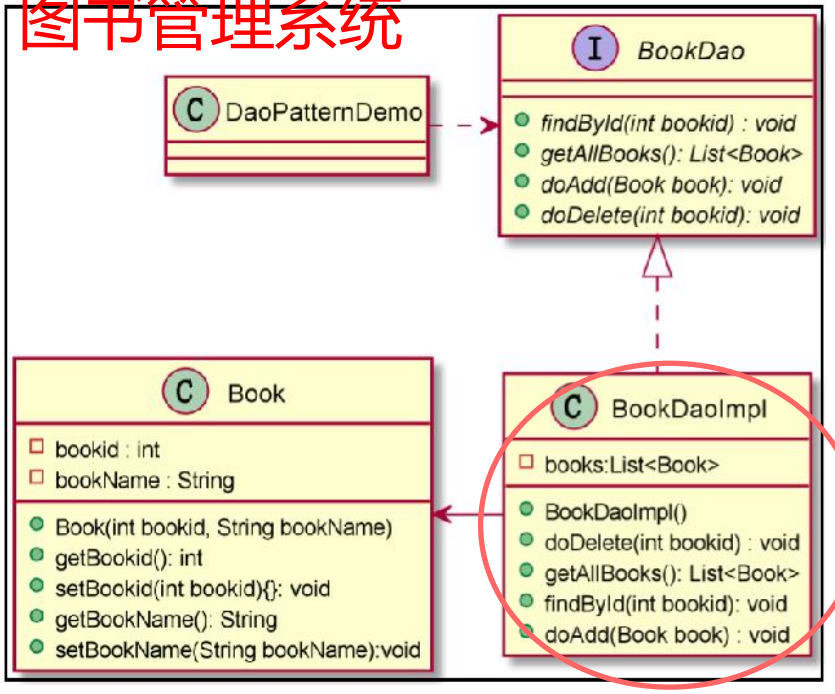


实验步骤

5 绘制数据访问对象模式类图

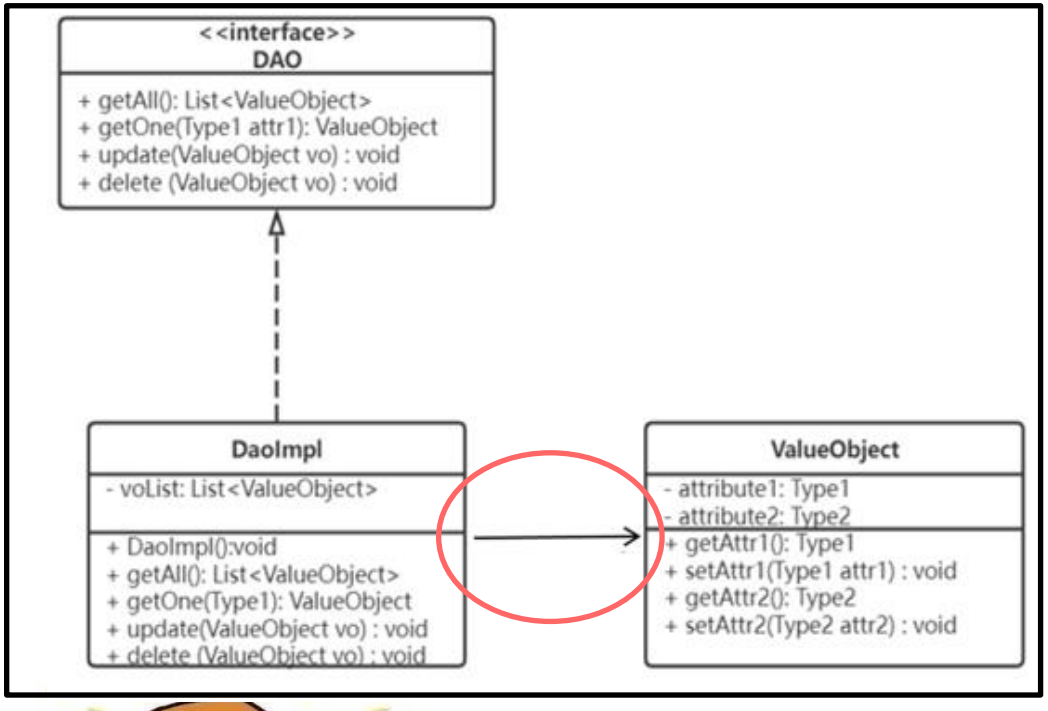


图书管理系统

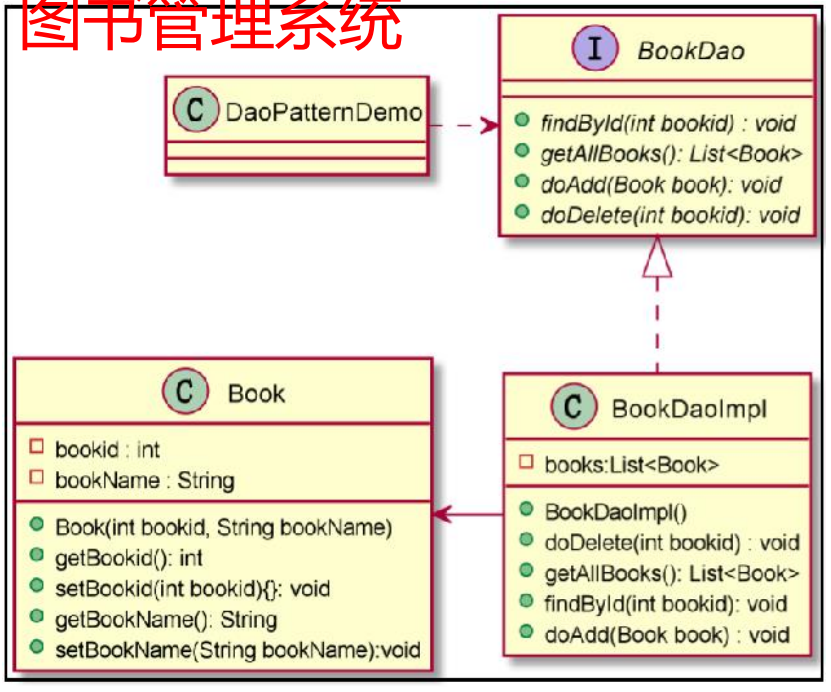


实验步骤

5 绘制数据访问对象模式类图



图书管理系统

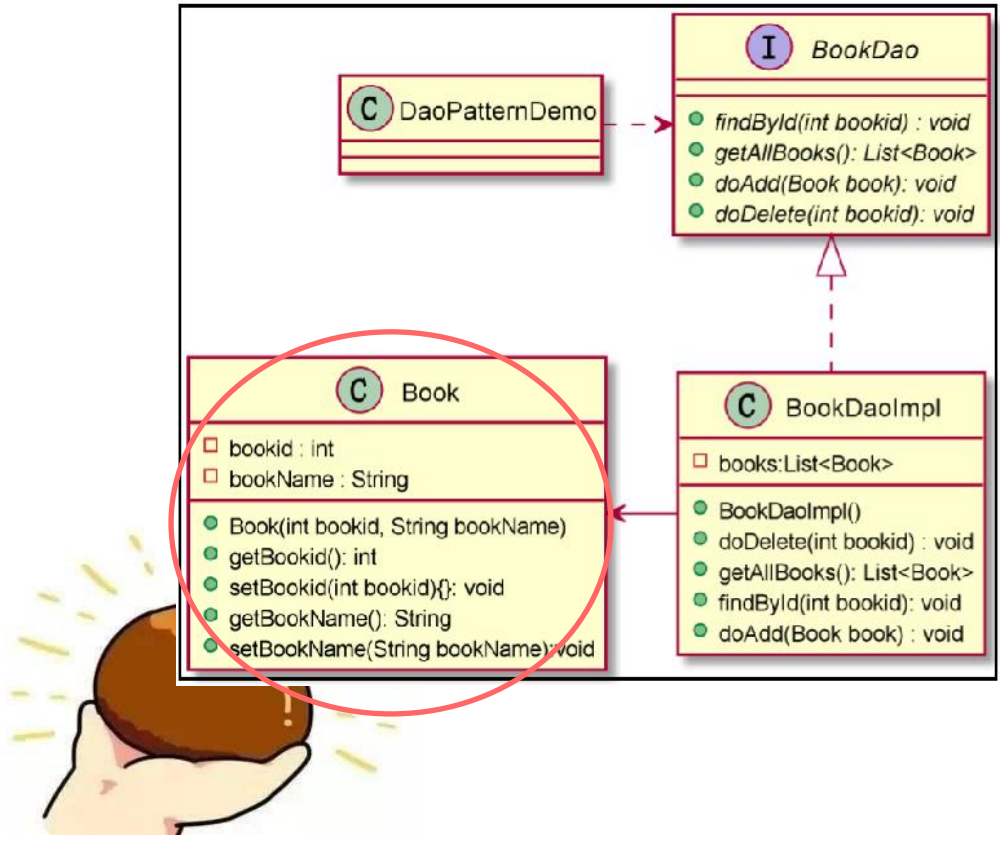


结合得分排行榜，思考一下ValueObject类需要哪些属性？DAO接口需要提供哪些方法？

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）：



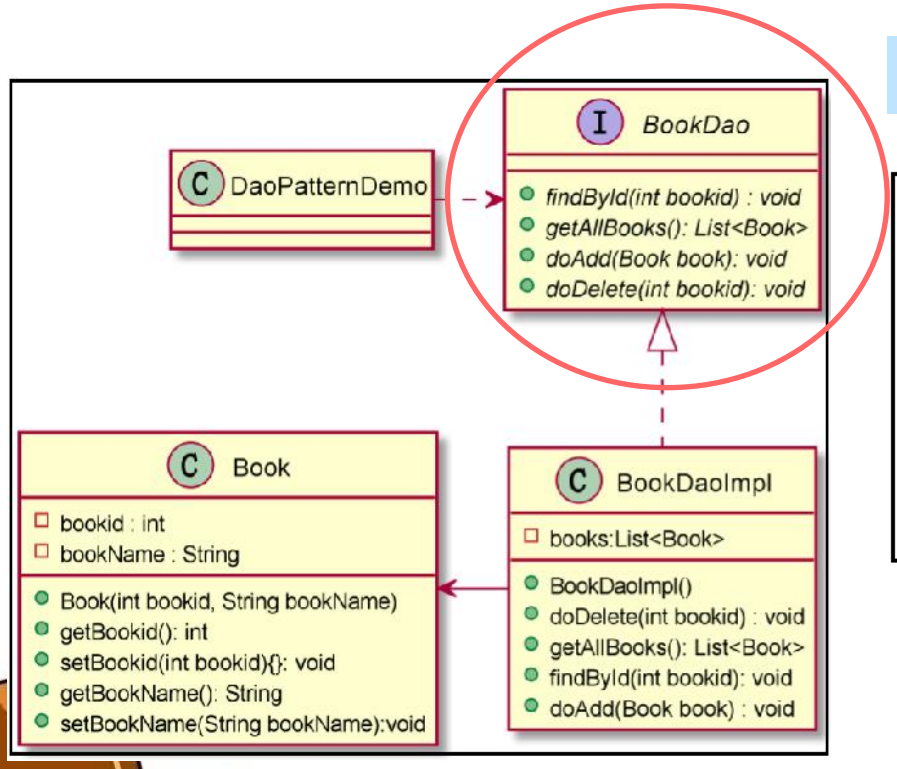
1、创建一个数值对象Book实体类；

```
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；

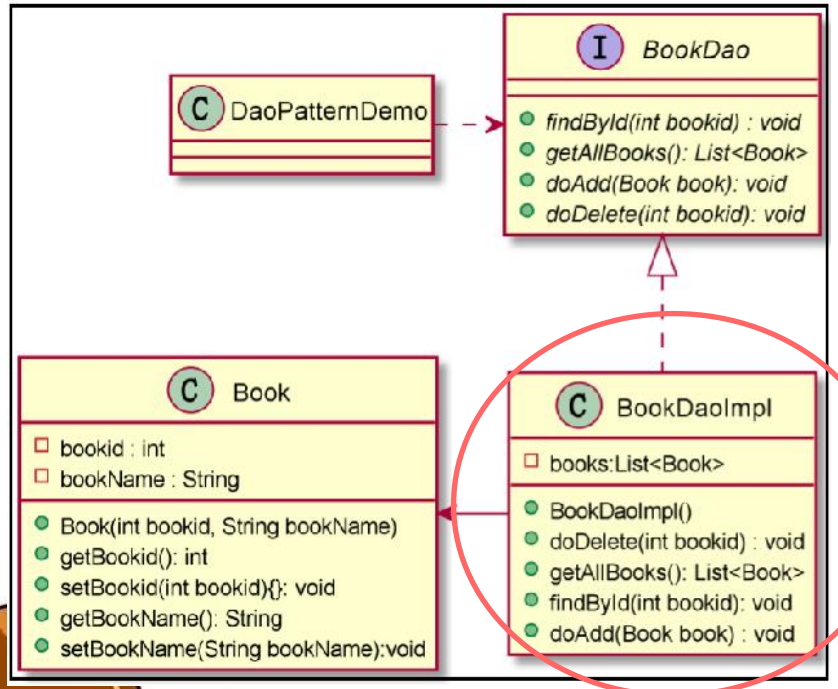
```
public interface BookDao {

    void findById(int bookid);

    List<Book> getAllBooks();

    void doAdd(Book book);

    void doDelete(int bookid);
}
```

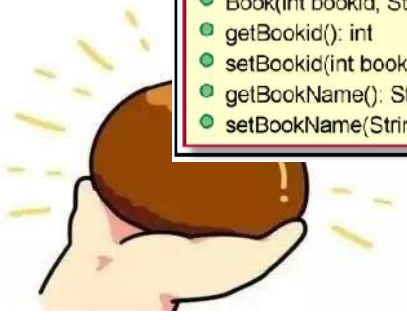
实验步骤

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；





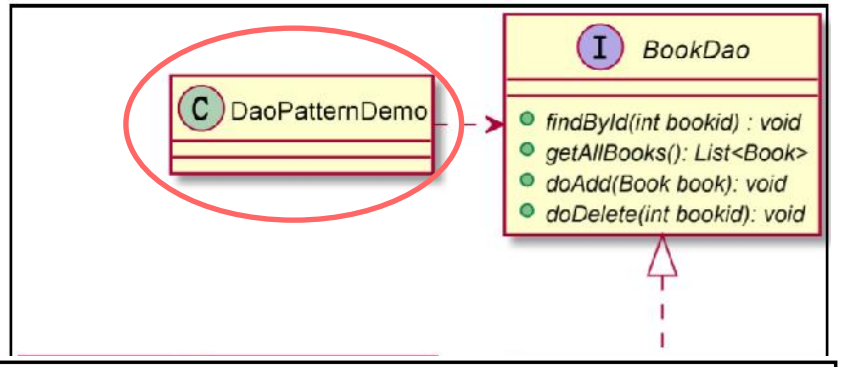
```
public class BookDaoImpl implements BookDao {

    //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;
    }
```

```
    //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) {
```

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）：




```
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]
```

4、使用DaoPatternDemo来演示数据访问对象模式的用法。

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

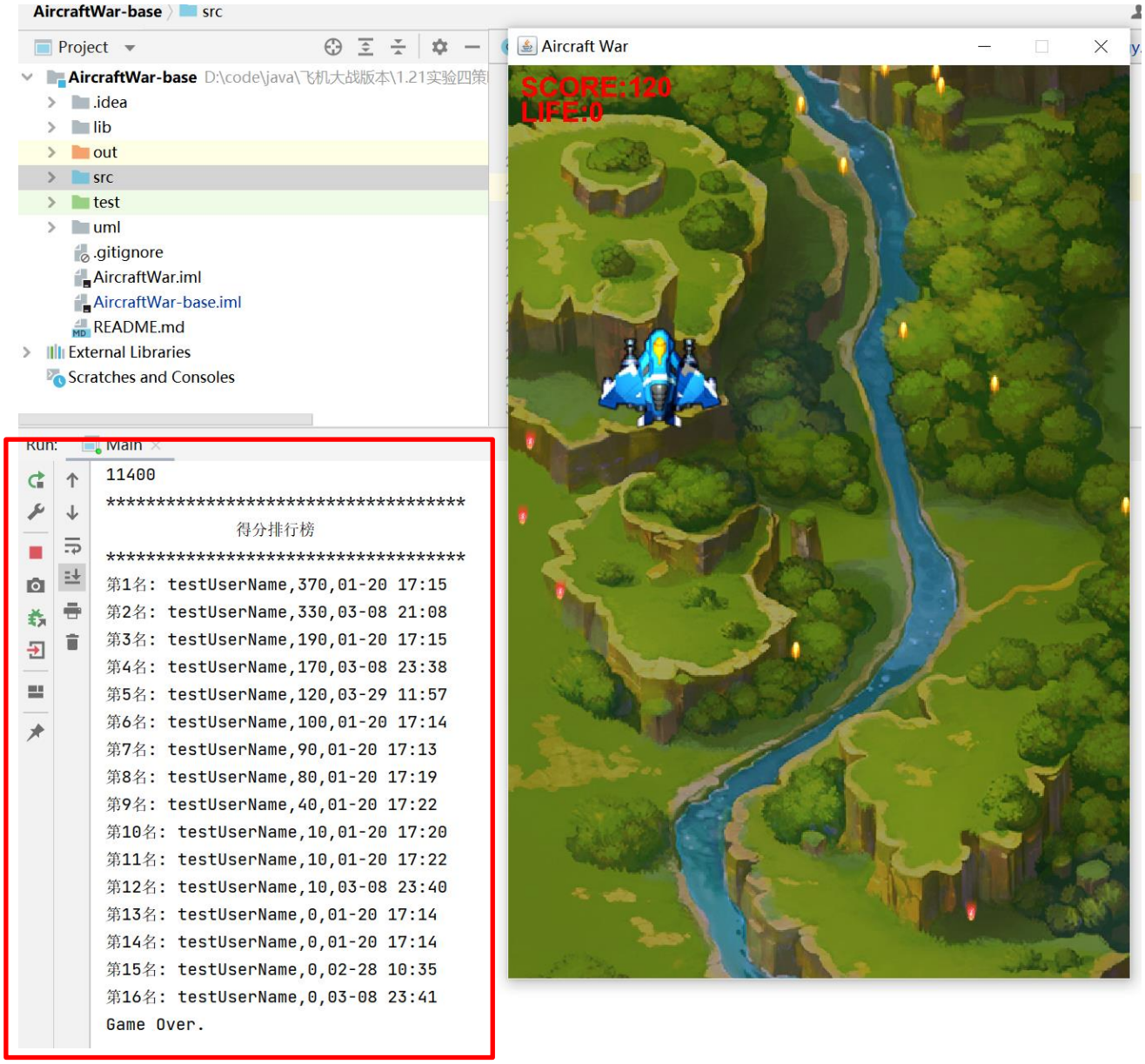
实验要求

本次实验提交版本需完成以下功能：

- ✓ 至少设计两种子弹发射弹道，如：直射、散射；
- ✓ 完成Boss机：得分每隔一定值产生一次Boss机，一个时刻只能有一个Boss机；
- ✓ 火力道具生效：直射变成散射（本次实验未涉及多线程，故火力改变后无法恢复，实验五继续完善即可）；
- ✓ 每局游戏结束后自动记录该局得分，并在控制台打印输出得分排行榜，得分数据存储在文件中（无需实现界面和玩家交互）。

注意：每次实验提交当次实验的版本。

实验要求


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 截图报告

- 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

作业提交

为方便批改，请同学们将参数做如下修改：

1、增大道具的掉落几率，比如30%掉落火力道具、 30%掉落加血道具、 30%掉落炸弹道具，还有10%不掉落道具；

2、英雄机血量上限设置为1000。

同学们
请开始实验吧！