浙江大学计算机学院

Java 程序设计课程报告

2024—2025 学年秋冬学期

题目	JDK 不变类与矩阵设计
学号	3220104147
学生姓名	余卓耘
所在专业	软件工程
所在班级	软工 2202

目　　录

1 引言

本次实验完成 Homework2 的两个任务。

1．1 JDK 中的不变类

(1) String 类

String 类是不可变类，其值在创建后不能被修改。这是因为 String 类的所有字段都是 final 的，并且类本身是 final 的，这意味着它不能被继承。String 类提供了很多方法，但是这些方法都不会改变原有字符串的内容，而是返回一个新的字符串对象。

例子：

```
String str = "Hello";
str = str + " World"; // 创建新对象
```

(2) Integer 类

Integer 类是不可变类，其值在创建后不能被修改。因为它的构造方法会创建一个包含给定整数值的新 Integer 对象，这个对象的状态是固定的。Integer 类提供了很多方法来操作整数，但这些方法都不会改变 Integer 对象的状态，而是返回新的 Integer 对象或者原始值。

例子：

```
Integer num = 10;
num = num + 1; // 创建新对象
```

1．2 源码分析

(1) String 类

```java
public final class String {
    private final char[] value; // 内部数组是final的
    private final int hash;     // hash值也是final的

    // 构造方法创建新对象
    public String(char[] value) {
        this.value = Arrays.copyOf(value, value.length);
    }

    // 所有修改操作都会返回新对象
    public String concat(String str) {
        return new String(this.value + str.value);
    }
}
```

(2) Integer 类

```java
public final class Integer {
    private final int value; // 内部值是final的

    // 操作都返回新对象
    public Integer(int value) {
        this.value = value;
    }
}
```

分析共性：

①final 字段：不变类的字段通常是 final 的，这意味着它们在初始化后不能被重新赋值。

②没有可变方法：不变类不提供修改其状态的方法。任何看似修改状态的方法实际上都会返回一个新的对象。

线程安全：由于状态不可变，不变类的对象是线程安全的，不需要额外的同步。

③安全性：不变对象可以自由地在多个线程或多个客户端之间共享，而不担心数据一致性问题。

2 矩阵设计

2．1 功能模块设计

本程序的类设计如下：

(1) **MutableMatrix（可变矩阵）**：提供可变矩阵的实现，允许矩阵元素的修改。

(2) **ImmutableMatrix（不可变矩阵）**：提供不可变矩阵的实现，一旦创建，矩阵元素不能被修改。

(3) **MatrixTest（矩阵测试）**：包含程序的入口点和测试功能，用于演示和测试矩阵类的功能。

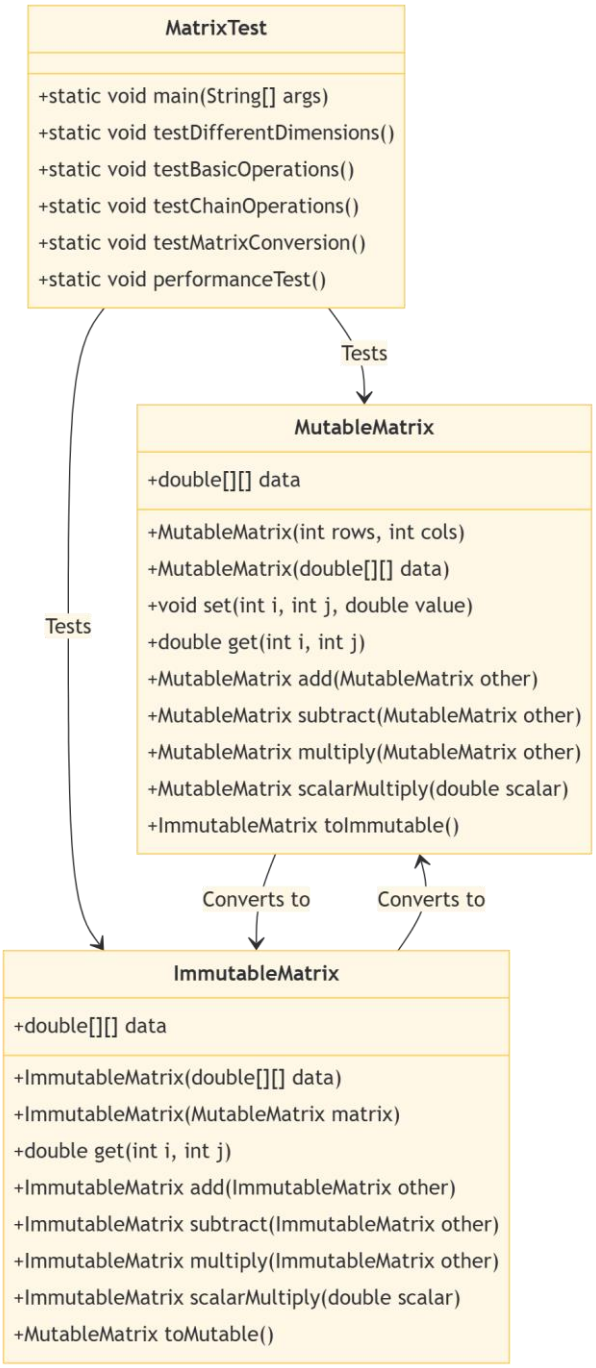
本程序需实现的主要功能有：

(1) **可变矩阵与不可变矩阵的转换**：提供方法将可变矩阵转换为不可变矩阵，反之亦然。

(2) **链式运算**：支持矩阵的链式运算，如 m1.add(m2).add(m3)。

(3)**矩阵的基本运算**：包括加法、减法、乘法和标量乘法。

程序的总体模块如图 1 所示：



2. 2　测试设计

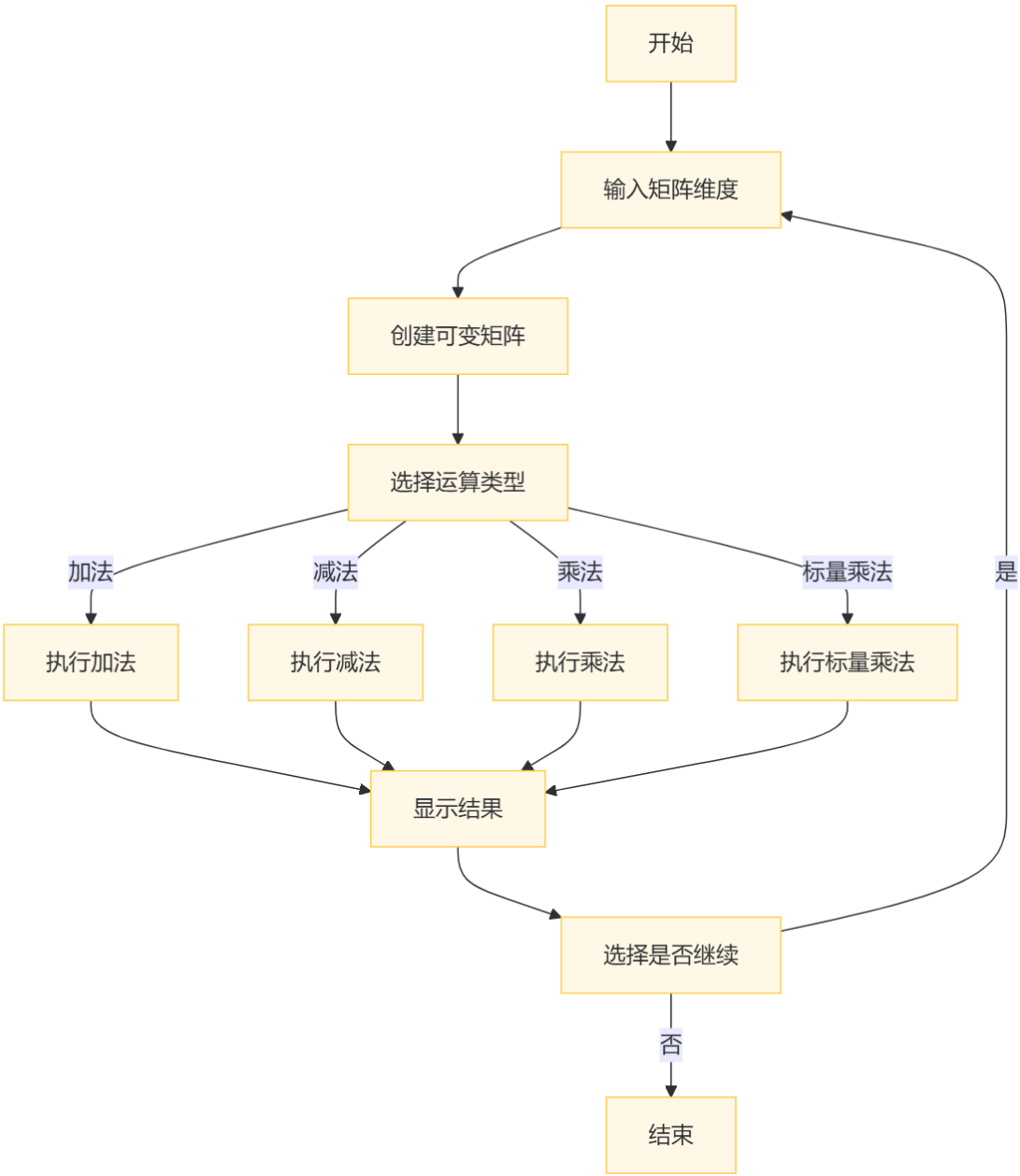
测试模块设计如下：

(1)**单元测试**：对每个矩阵运算方法进行单独测试，确保其正确性。

(2)**集成测试**：测试可变矩阵与不可变矩阵之间的转换功能。

(3)**性能测试**：测试不同维度矩阵的运算性能，包括加法、乘法和链式操作。

测试设计流程图如下：


开始
输入矩阵维度
创建可变矩阵
选择运算类型
加法
减法
乘法
标量乘法
执行加法
执行减法
执行乘法
执行标量乘法
显示结果
选择是否继续
是
否
结束

3 测试与运行

3.1 程序测试

在程序代码基本完成后，经过不断的调试与修改，最后测试本次所设计的矩阵计算模块戏能够正常运行。具体运行结果见 3.2 部分截图。

3.2 程序运行

主界面如图所示：

```
PS D:\User\Desktop\JAVA\2024hw\ZJU_JAVA_2024\hw2> javac -encoding UTF-8 -d . MutableMatrix.java ImmutableMatrix.java Mat
rixTest.java
PS D:\User\Desktop\JAVA\2024hw\ZJU_JAVA_2024\hw2> java hw2.MatrixTest

矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
```

测试如下所示：

测试不同维度矩阵运算

```
矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
1
输入第一个矩阵的维度 (行 列):
2 3
输入第二个矩阵的维度 (行 列):
3 2
输入第一个矩阵的元素 (2x3):
1 2 3
4 5 6
输入第二个矩阵的元素 (3x2):
1 2
3 4
5 6
矩阵乘法结果:
22.00 28.00 3.00
49.00 64.00 6.00
```

测试基本运算：

```
矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
2
输入测试矩阵维度 (行 列):
2 2
输入第一个矩阵的元素 (2x2):
1 2
3 4
输入第二个矩阵的元素 (2x2):
5 6
7 8
加法结果:
6.00 8.00
10.00 12.00
减法结果:
1.00 2.00
3.00 4.00
输入标量乘数:
2.5
标量乘法结果:
2.50 5.00
7.50 10.00
```

测试链式矩阵：

```
矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
3
输入矩阵维度 (行 列):
2 2
输入第一个矩阵的元素 (2x2):
1 1
1 1
输入第二个矩阵的元素 (2x2):
2 2
2 2
输入第三个矩阵的元素 (2x2):
3 3
3 3
链式运算 (m1.add(m2).add(m3)) 结果:
6.00 6.00
6.00 6.00
```

测试可变、不可变矩阵转换

（普通、含有负数、行列数量不同的矩阵）

```
矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
4
输入矩阵维度 (行 列):
2 2
输入可变矩阵的元素 (2x2):
1.5 2.0
3.5 4.0
原始可变矩阵:
1.50 2.00
3.50 4.00
转换为不可变矩阵:
1.50 2.00
3.50 4.00
再次转换为可变矩阵:
1.50 2.00
3.50 4.00
```

```
矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
4
输入矩阵维度 (行 列):
3 3
输入可变矩阵的元素 (3x3):
-1.5 2.7 -3.2
4.1 -5.9 6.3
-7.8 8.4 -9.2
原始可变矩阵:
-1.50 2.70 -3.20
4.10 -5.90 6.30
-7.80 8.40 -9.20
转换为不可变矩阵:
-1.50 2.70 -3.20
4.10 -5.90 6.30
-7.80 8.40 -9.20
再次转换为可变矩阵:
-1.50 2.70 -3.20
4.10 -5.90 6.30
-7.80 8.40 -9.20
```

```
矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
4
输入矩阵维度 (行 列):
2 3
输入可变矩阵的元素 (2x3):
1.1 2.2 3.3
4.4 5.5 6.6
原始可变矩阵:
1.10 2.20 3.30
4.40 5.50 6.60
转换为不可变矩阵:
1.10 2.20 3.30
4.40 5.50 6.60
再次转换为可变矩阵:
1.10 2.20 3.30
4.40 5.50 6.60
```

性能如下所示：

```
矩阵运算测试系统
1. 测试不同维度矩阵运算
2. 测试基本运算(加减乘、标量乘法)
3. 测试链式运算
4. 测试可变矩阵与不可变矩阵转换
5. 性能测试
0. 退出
5

=== 性能测试开始 ===

测试 10x10 维度矩阵:

加法操作:
可变矩阵: 0.014 ms
不可变矩阵: 0.036 ms

乘法操作:
可变矩阵: 0.072 ms
不可变矩阵: 0.090 ms

链式操作 (add->multiply->scalarMultiply):
可变矩阵: 0.092 ms
不可变矩阵: 0.100 ms

测试 50x50 维度矩阵:

加法操作:
可变矩阵: 0.116 ms
不可变矩阵: 0.175 ms

乘法操作:
可变矩阵: 5.889 ms
不可变矩阵: 5.467 ms

链式操作 (add->multiply->scalarMultiply):
可变矩阵: 1.335 ms
不可变矩阵: 1.237 ms

测试 100x100 维度矩阵:

加法操作:
可变矩阵: 0.426 ms
不可变矩阵: 0.650 ms

乘法操作:
可变矩阵: 4.354 ms
不可变矩阵: 6.149 ms

链式操作 (add->multiply->scalarMultiply):
可变矩阵: 3.150 ms
不可变矩阵: 3.197 ms
```

```
测试 200x200 维度矩阵：

加法操作：
可变矩阵：2.402 ms
不可变矩阵：3.002 ms

乘法操作：
可变矩阵：17.550 ms
不可变矩阵：16.230 ms

链式操作 (add->multiply->scalarMultiply)：
可变矩阵：12.946 ms
不可变矩阵：19.667 ms

测试 500x500 维度矩阵：

加法操作：
可变矩阵：1.891 ms
不可变矩阵：4.340 ms

乘法操作：
可变矩阵：259.768 ms
不可变矩阵：254.665 ms

链式操作 (add->multiply->scalarMultiply)：
可变矩阵：254.460 ms
不可变矩阵：263.692 ms
```

3. 结论分析

汇总性能测试结果如下表：

矩阵维度	操作类型	可变矩阵(ms)	不可变矩阵(ms)
10x10	加法	0.014	0.036
10x10	乘法	0.072	0.090
10x10	链式操作	0.092	0.100
50x50	加法	0.116	0.175
50x50	乘法	5.889	5.467
50x50	链式操作	1.335	1.237
100x100	加法	0.426	0.650
100x100	乘法	4.354	6.149
100x100	链式操作	3.150	3.197
200x200	加法	2.402	3.002
200x200	乘法	17.550	16.230
200x200	链式操作	12.946	19.667
500x500	加法	1.891	4.340
500x500	乘法	259.768	254.665
500x500	链式操作	254.460	263.692

根据性能测试结果，我们可以得出以下结论：

- 加法操作：在所有测试的矩阵维度中，不可变矩阵的加法操作普遍比可变矩阵耗时更长。这可能是因为不可变矩阵在执行加法时需要创建新的矩阵对象。
- 乘法操作：不可变矩阵的乘法操作同样比可变矩阵耗时，且随着矩阵维度的增加，性能差距逐渐增大。这表明在处理大规模数据时，可变矩阵可能更适合。
- 链式操作：在链式操作中，不可变矩阵的性能劣势更加明显，尤其是在高维度矩阵上。这强调了在需要频繁修改矩阵内容的应用场景中，可变矩阵的优势。

5. 总结

在本次实验中，我学到了矩阵运算的基本原理以及如何在 Java 中实现可变和不可变矩阵。通过性能测试，我了解到了不同类型矩阵在不同操作下的性能差异，这对于选择适当的数据结构具有重要意义。

在实验过程中，我遇到了一些困难，比如在实现链式操作时，如何正确地返回新的矩阵对象而不是修改原有对象。通过阅读相关文档和代码审查，我逐渐理解了不可变对象的设计模式，并成功实现了所需的功能。

通过这次实验，我不仅提高了编程技能，还加深了对数据结构和算法性能影响的理解。这些经验将对我未来的学习和工作大有裨益。

参考文献

[1] 耿祥义. Java 大学实用教程[M]. 北京：清华大学出版社，2009.

[2] 耿祥义. Java 课程设计[M]. 北京：清华大学出版社，2008..

[3] 丁振凡. Java 语言实验教程[M]. 北京：北京邮电大学出版社，2005.

[4] 郑莉. Java 语言程序设计[M]. 北京：清华大学出版社，2006.