

探究不变类&设计实现可变类/不变类

姓名：姜雨童
学号：3220103450

Chap 1: 探究不变类

1 任务要求

寻找JDK库中的不变类（至少2类），并进行源码分析，分析其为什么是不变的？文档说明其共性。

2 源码分析

这里选取 `String` 和 `Integer` 作为分析对象。

2.1 String

首先可以看到原作者在源码注释中对 `String` 类的不变性做了说明：String是一种常量，被创建后它的值就不可发生改变，因此它是线程安全的，可以被共享。

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared.

随后对源码进行分析。类的内部使用了一个 `private` 的数组变量 `value` 来存储字符串的数据，同时，该数组被声明为 `final` 来确保构造完成后它不能指向其他数组。因此，调用构造函数完成初始化后，字符数组的引用不能被外部改变。最后，类内没有提供 mutator methods（可修改字符串内容的方法），也没有任何外界可接触的函数返回了可修改的引用；进行字符串修改操作时，类会构建并返回一个新的字符串对象，而不改变原对象。

部分代码如下：

```

1  public final class String
2      implements java.io.Serializable, Comparable<String>, CharSequence {
3      /** The value is used for character storage. */
4      private final char value[];
5
6      /** Cache the hash code for the string */
7      private int hash; // Default to 0
8
9      /** use serialVersionUID from JDK 1.0.2 for interoperability */
10     private static final long serialVersionUID = -6849794470754667710L;
11
12     private static final ObjectStreamField[] serialPersistentFields =
13         new ObjectStreamField[0];
14
15     public String() {
16         this.value = "".value;
17     }
18
19     public String(String original) {
20         this.value = original.value;
21         this.hash = original.hash;
22     }
23
24     public String(char value[]) {
25         this.value = Arrays.copyOf(value, value.length);
26     }
27
28     ... // omitted
29 }
```

2.2 Integer

类似地，`Integer` 中使用一个 `private final` 变量来存储整数值，确保对象在调用构造函数创建实例之后就不可变。而 `Integer` 类也没有提供任何 mutator methods 或是外界可接触的返回了可修改引用的函数，因此不能修改。

部分代码如下：

```

1  public final class Integer extends Number implements Comparable<Integer> {
2      private final int value;
3
4      public Integer(int value) {
5          this.value = value;
6      }
7
8      public short shortValue() {
9          return (short)value;
10     }
11
12     public int intValue() {
13         return value;
14     }
15
16     ... // omitted
17 }
```

3 共性分析

所有的数据都是 `private` 类型的，类是 `final` 类型的来确保不可变性。并且类不提供能够修改对象状态或是返回可修改对象引用的方法。

Chap 2: 设计实现可变类/不变类

1 任务要求

设计并实现类 `MutableMatrix` 和 `InmutableMatrix`：

- 体现可变类和不可变类的区别
- 具备矩阵的一般运算操作
- 提供 `MutableMatrix(InmutableMatrix)` 和 `InmutableMatrix(MutableMatrix)` 构造函数
- 支持矩阵链式运算，如
 - `MutableMatrix m1, m2, m3;`
 - `.....`
 - `m1.add(m2).add(m3)`

撰写相关文档并进行测试说明和性能分析。

2 代码实现

2.1 可变/不可变的区别

在本次代码实现中，类的可变与不可变主要体现在两个方面：变量是否设置成不可更改的 `final`，以及返回的是变量的引用还是新实例。

可变矩阵类的数据矩阵 `data` 为 `private` 类型，构造函数也是直接指向传入数组的引用，进行运算后对当前对象进行修改，`getData` 函数返回该对象的引用；而不可变矩阵的数据矩阵 `data` 为不可修改引用的 `private final` 类型，构造时对传入数组进行深度拷贝，同时运算后/`getData` 函数返回新实例，因此类对象是不可修改的。

```

1  class MutableMatrix {
2      private double[][] data;
3
4      // Ctor
5      public MutableMatrix(double[][] data) {
6          this.data = data;
7      }
8
9      // add
10     public MutableMatrix add(MutableMatrix matrix) {
11         ... // omitted
12         return this;
13     }
14     ...
15     public double[][] getData() {
16         return data; // return reference to internal data
17     }
18 }

1  class ImmutableMatrix {
2      private final double[][] data;
3
4      // Ctor
5      public ImmutableMatrix(double[][] data) {
6          // deep copy of data
7          this.data = new double[data.length][data[0].length];
8          for (int i = 0; i < data.length; i++) {
9              System.arraycopy(data[i], 0, this.data[i], 0, data[i].length);
10         }
11     }
12
13     // add
14     public ImmutableMatrix add(ImmutableMatrix matrix) {
15         double[][] result = new double[this.data.length][this.data[0].length];
16         ... // omitted
17         return new ImmutableMatrix(result); // return new instance
18     }
19     ...
20     public double[][] getData() {
21         ImmutableMatrix im = new ImmutableMatrix(data); // return new instance
22         return im.data;
23     }
24 }

```

在测试结果中可以看到，对可变类的修改成功执行，而对不可变类则不会进行修改：

```
// test mutability / immutability
m1.getData()[0][0] = 10; // try to change mutable matrix
System.out.println(x:"\33[36mMutable Matrix(changed):\33[0m");
printMatrix(m1.getData());

im1.getData()[0][0] = 10; // try to change immutable matrix
System.out.println(x:"\33[36mImmutable Matrix(changed):\33[0m");
printMatrix(im1.getData());
```

```
Mutable Matrix(changed):
10.0 86.0
122.0 142.0
Immutable Matrix(changed):
1.0 2.0
3.0 4.0
```

2.2 矩阵一般运算操作 & 链式运算

这里对可变类与不可变类均实现了矩阵加法、减法，以及乘法的运算操作，并且支持链式运算：

```
// test mutable matrix
MutableMatrix m1 = new MutableMatrix(new double[][]{{1, 2}, {3, 4}});
MutableMatrix m2 = new MutableMatrix(new double[][]{{5, 6}, {7, 8}});

m1.add(m2).subtract(new MutableMatrix(new double[][]{{1, 1}, {1, 1}})); // line
System.out.println(x:"\33[32mMutable Matrix(add, sub):\33[0m");
printMatrix(m1.getData());

m1.multiply(m2);
System.out.println(x:"\33[32mMutable Matrix(mul):\33[0m");
printMatrix(m1.getData());

// test immutable matrix
ImmutableMatrix im1 = new ImmutableMatrix(new double[][]{{1, 2}, {3, 4}});
ImmutableMatrix im2 = new ImmutableMatrix(new double[][]{{5, 6}, {7, 8}});

ImmutableMatrix imResult = im1.add(im2).subtract(new ImmutableMatrix(new double[][]{{1, 1}, {1, 1}}));
System.out.println(x:"\33[33mImmutable Matrix(add, sub):\33[0m");
printMatrix(imResult.getData());

ImmutableMatrix imResult2 = im1.multiply(im2);
System.out.println(x:"\33[33mImmutable Matrix(mul):\33[0m");
printMatrix(imResult2.getData());
```

```
Mutable Matrix(add, sub):
5.0 7.0
9.0 11.0
Mutable Matrix(mul):
74.0 86.0
122.0 142.0
Immutable Matrix(add, sub):
5.0 7.0
9.0 11.0
Immutable Matrix(mul):
19.0 22.0
43.0 50.0
```

可以看到两次乘法运算的结果不一致，这是因为可变矩阵类对象 `m1` 在经过加减法后，值更改为 `{{5, 7}, {9, 11}}`，而不可变矩阵类对象 `im1` 的值并没有发生改变，仍然是 `{{1, 2}, {3, 4}}`。

The interface consists of two main input panels, '输入矩阵A' and '输入矩阵B', and a result display area on the right. Each input panel has buttons for '插入矩阵' (Insert Matrix) and '恢复矩阵' (Restore Matrix). Below these are input fields for matrix elements. In the top screenshot, matrix A is [[1, 2], [3, 4]] and matrix B is [[5, 6], [7, 8]]. The result matrix C is shown as [[74, 86], [122, 142]]. In the bottom screenshot, matrix A is [[1, 1], [2, 3]] and matrix B is [[5, 6], [7, 8]]. The result matrix C is shown as [[19, 22], [43, 50]].

2.3 互相转换的构造函数

因为函数以及调用的构造函数本身不对传入的对象做修改，且在不可变类的 `getData` 函数中已经实现了返回新实例而不是对象的引用，因此这里不需要对传入的不可变类做保护。

```

1 // convert fromImmutable
2     public static MutableMatrix fromImmutable(ImmutableMatrix immutableMatrix) {
3         return new MutableMatrix(immutableMatrix.getData());
4     }
5
6 // convert fromMutable
7     public static ImmutableMatrix fromMutable(MutableMatrix mutableMatrix) {
8         return new ImmutableMatrix(mutableMatrix.getData());
9     }

```

测试结果如下：

```

// test conversion
MutableMatrix m3 = MutableMatrix.fromImmutable(im1);
System.out.println(x:"\33[34mMutable Matrix(fromImmutable):\33[0m");
printMatrix(m3.getData());

ImmutableMatrix im3 = ImmutableMatrix.fromMutable(m1);
System.out.println(x:"\33[35mImmutable Matrix(fromMutable):\33[0m");
printMatrix(im3.getData());

```

```

Mutable Matrix(fromImmutable):
1.0 2.0
3.0 4.0
Immutable Matrix(fromMutable):
74.0 86.0
122.0 142.0

```

3 完整测试文档&结果

```

1 public class MatrixTest {
2
3     public static void main(String[] args) {
4         System.out.println("\33[44m===== Testing Matrices =====\33[0m\n");
5
6         // test mutable matrix
7         MutableMatrix m1 = new MutableMatrix(new double[][]{{1, 2}, {3, 4}});
8         MutableMatrix m2 = new MutableMatrix(new double[][]{{5, 6}, {7, 8}});
9
10        m1.add(m2).subtract(new MutableMatrix(new double[][]{{1, 1}, {1, 1}}));
11        System.out.println("\33[32mMutable Matrix(add, sub):\33[0m");
12        printMatrix(m1.getData());
13
14        m1.multiply(m2);
15        System.out.println("\33[32mMutable Matrix(mul):\33[0m");
16        printMatrix(m1.getData());
17
18
19        // test immutable matrix
20        ImmutableMatrix im1 = new ImmutableMatrix(new double[][]{{1, 2}, {3, 4}});
21        ImmutableMatrix im2 = new ImmutableMatrix(new double[][]{{5, 6}, {7, 8}});
22
23        ImmutableMatrix imResult = im1.add(im2).subtract(new ImmutableMatrix(new
double[][]{{1, 1}, {1, 1}}));
24        System.out.println("\33[33mImmutable Matrix(add, sub):\33[0m");
25        printMatrix(imResult.getData());
26
27        ImmutableMatrix imResult2 = im1.multiply(im2);
28        System.out.println("\33[33mImmutable Matrix(mul):\33[0m");
29        printMatrix(imResult2.getData());

```

```

30
31 // test conversion
32 MutableMatrix m3 = MutableMatrix.fromImmutable(im1);
33 System.out.println("\33[34mMutable Matrix(fromImmutable):\33[0m");
34 printMatrix(m3.getData());
35
36 ImmutableMatrix im3 = ImmutableMatrix.fromMutable(m1);
37 System.out.println("\33[35mImmutable Matrix(fromMutable):\33[0m");
38 printMatrix(im3.getData());
39
40 // test mutability / immutability
41 m1.getData()[0][0] = 10; // try to change mutable matrix
42 System.out.println("\33[36mMutable Matrix(changed):\33[0m");
43 printMatrix(m1.getData());
44
45 im1.getData()[0][0] = 10; // try to change immutable matrix
46 System.out.println("\33[36mImmutable Matrix(changed):\33[0m");
47 printMatrix(im1.getData());
48 }
49
50 private static void printMatrix(double[][] matrix) {
51     for (double[] row : matrix) {
52         for (double value : row) {
53             System.out.print(value + " ");
54         }
55         System.out.println();
56     }
57 }
58 }

```

===== Testing Matrices =====

```

Mutable Matrix(add, sub):
5.0 7.0
9.0 11.0
Mutable Matrix(mul):
74.0 86.0
122.0 142.0
Immutable Matrix(add, sub):
5.0 7.0
9.0 11.0
Immutable Matrix(mul):
19.0 22.0
43.0 50.0
Mutable Matrix(fromImmutable):
1.0 2.0
3.0 4.0
Immutable Matrix(fromMutable):
74.0 86.0
122.0 142.0
Mutable Matrix(changed):
10.0 86.0
122.0 142.0
Immutable Matrix(changed):
1.0 2.0
3.0 4.0

```

4 源代码

4.1 可变类

```

1  class MutableMatrix {
2      private double[][] data;
3
4      // Ctor
5      public MutableMatrix(double[][] data) {
6          this.data = data;
7      }
8
9      // add
10     public MutableMatrix add(MutableMatrix matrix) {
11         if (this.data.length != matrix.data.length || this.data[0].length !=
matrix.data[0].length) {
12             throw new IllegalArgumentException("\33[31mMatrices must have the same
dimensions.\33[0m");
13         }
14         for (int i = 0; i < this.data.length; i++) {
15             for (int j = 0; j < this.data[i].length; j++) {
16                 this.data[i][j] += matrix.data[i][j];
17             }
18         }
19         return this;
20     }
21     // subtract
22     public MutableMatrix subtract(MutableMatrix matrix) {
23         if (this.data.length != matrix.data.length || this.data[0].length !=
matrix.data[0].length) {
24             throw new IllegalArgumentException("\33[31mMatrices must have the same
dimensions.\33[0m");
25         }
26         for (int i = 0; i < this.data.length; i++) {
27             for (int j = 0; j < this.data[i].length; j++) {
28                 this.data[i][j] -= matrix.data[i][j];
29             }
30         }
31         return this;
32     }
33     // multiply
34     public MutableMatrix multiply(MutableMatrix matrix) {
35         if (this.data[0].length != matrix.data.length) { // mxn * nxp
36             throw new IllegalArgumentException("\33[31mInvalid dimensions for
multiplication.\33[0m");
37         }
38         double[][] result = new double[this.data.length][matrix.data[0].length]; //
mxp
39         for (int i = 0; i < this.data.length; i++) { // rows of m
40             for (int j = 0; j < matrix.data[0].length; j++) { // cols of p
41                 for (int k = 0; k < this.data[0].length; k++) { // cols of n
42                     result[i][j] += this.data[i][k] * matrix.data[k][j];
43                 }
44             }
45         }
46         this.data = result;

```



```

47         return this;
48     }
49
50     // convert fromImmutable
51     public static MutableMatrix fromImmutable(InmutableMatrix immutableMatrix) {
52         return new MutableMatrix(immutableMatrix.getData());
53     }
54
55     public double[][] getData() {
56         return data; // return reference to internal data, changes to this will change
the matrix
57     }
58 }

```

4.2 不变类

```

1  class InmutableMatrix {
2      private final double[][] data;
3
4      // Ctor
5      public InmutableMatrix(double[][] data) {
6          // deep copy of data
7          this.data = new double[data.length][data[0].length];
8          for (int i = 0; i < data.length; i++) {
9              System.arraycopy(data[i], 0, this.data[i], 0, data[i].length);
10         }
11     }
12
13     // add
14     public InmutableMatrix add(InmutableMatrix matrix) {
15         if (this.data.length != matrix.data.length || this.data[0].length !=
matrix.data[0].length) {
16             throw new IllegalArgumentException("\33[31mMatrices must have the same
dimensions.\33[0m");
17         }
18         double[][] result = new double[this.data.length][this.data[0].length];
19         for (int i = 0; i < this.data.length; i++) {
20             for (int j = 0; j < this.data[i].length; j++) {
21                 result[i][j] = this.data[i][j] + matrix.data[i][j];
22             }
23         }
24         return new InmutableMatrix(result); // return new instance
25     }
26     // subtract
27     public InmutableMatrix subtract(InmutableMatrix matrix) {
28         if (this.data.length != matrix.data.length || this.data[0].length !=
matrix.data[0].length) {
29             throw new IllegalArgumentException("\33[31mMatrices must have the same
dimensions.\33[0m");
30         }
31         double[][] result = new double[this.data.length][this.data[0].length];
32         for (int i = 0; i < this.data.length; i++) {
33             for (int j = 0; j < this.data[i].length; j++) {
34                 result[i][j] = this.data[i][j] - matrix.data[i][j];
35             }
36         }

```

```

37         return new ImmutableMatrix(result);
38     }
39     // multiply
40     public ImmutableMatrix multiply(ImmutableMatrix matrix) {
41         if (this.data[0].length != matrix.data.length) {
42             throw new IllegalArgumentException("\33[31mInvalid dimensions for
multiplication.\33[0m");
43         }
44         double[][] result = new double[this.data.length][matrix.data[0].length];
45         for (int i = 0; i < this.data.length; i++) {
46             for (int j = 0; j < matrix.data[0].length; j++) {
47                 for (int k = 0; k < this.data[0].length; k++) {
48                     result[i][j] += this.data[i][k] * matrix.data[k][j];
49                 }
50             }
51         }
52         return new ImmutableMatrix(result);
53     }
54
55     // convert fromMutable
56     public static ImmutableMatrix fromMutable(MutableMatrix mutableMatrix) {
57         return new ImmutableMatrix(mutableMatrix.getData());
58     }
59
60     public double[][] getData() {
61         ImmutableMatrix im = new ImmutableMatrix(data); // return new instance with
same data
62         return im.data;
63     }
64 }

```