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：

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {
    /** The value is used for character storage. */
    private final char value[];

    /** Cache the hash code for the string */
    private int hash; // Default to 0

    /** use serialVersionUID from JDK 1.0.2 for interoperability */
    private static final long serialVersionUID = -6849794470754667710L;

    private static final ObjectStreamField[] serialPersistentFields =
        new ObjectStreamField[0];

    public String() {
        this.value = "".value;
    }

    public String(String original) {
        this.value = original.value;
        this.hash = original.hash;
    }

    public String(char value[]) {
        this.value = Arrays.copyOf(value, value.length);
    }

    ... // omitted
}
```

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：

```java
public final class Integer extends Number implements Comparable<Integer> {
    private final int value;

    public Integer(int value) {
        this.value = value;
    }

    public short shortValue() {
        return (short)value;
    }

    public int intValue() {
        return value;
    }

    ... // omitted
}
```

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
  - MutableMatrixm1, 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。

```java
class MutableMatrix {
    private double[][] data;

    // Ctor
    public MutableMatrix(double[][] data) {
        this.data = data;
    }

    // add
    public MutableMatrix add(MutableMatrix matrix) {
        ... // omitted
        return this;
    }
    ...
    public double[][] getData() {
        return data; // return reference to internal data
    }
}
```

```java
class InmutableMatrix {
    private final double[][] data;

    // Ctor
    public InmutableMatrix(double[][] data) {
        // deep copy of data
        this.data = new double[data.length][data[0].length];
        for (int i = 0; i < data.length; i++) {
            System.arraycopy(data[i], 0, this.data[i], 0, data[i].length);
        }
    }

    // add
    public InmutableMatrix add(InmutableMatrix matrix) {
        double[][] result = new double[this.data.length][this.data[0].length];
        ... // omitted
        return new InmutableMatrix(result); // return new instance
    }
    ...
    public double[][] getData() {
        InmutableMatrix im = new InmutableMatrix(data); // return new instance
        return im.data;
    }
}
```

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：

```java
// 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：

```java
// 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
InmutableMatrix im1 = new InmutableMatrix(new double[][]{{1, 2}, {3, 4}});
InmutableMatrix im2 = new InmutableMatrix(new double[][]{{5, 6}, {7, 8}});

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

InmutableMatrix 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}}`。

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。

```
// convert fromInmutable
    public static MutableMatrix fromInmutable(InmutableMatrix inmutableMatrix) {
        return new MutableMatrix(inmutableMatrix.getData());
    }

// convert fromMutable
    public static InmutableMatrix fromMutable(MutableMatrix mutableMatrix) {
        return new InmutableMatrix(mutableMatrix.getData());
    }
```

测试结果如下：

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

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

```
Mutable Matrix(fromInmutable):
1.0 2.0
3.0 4.0
Inmutable Matrix(fromMutable):
74.0 86.0
122.0 142.0
```

3 完整测试文档&结果

```
public class MatrixTest {

    public static void main(String[] args) {
        System.out.println("\33[44m====== Testing Matrices ======\33[0m\n");

        // 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}}));
        System.out.println("\33[32mMutable Matrix(add, sub):\33[0m");
        printMatrix(m1.getData());

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


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

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

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

```java

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

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

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

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

    private static void printMatrix(double[][] matrix) {
       for (double[] row : matrix) {
           for (double value : row) {
               System.out.print(value + " ");
           }
           System.out.println();
       }
   }
}
```

```
====== 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(fromInmutable):
1.0 2.0
3.0 4.0
Inmutable 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

```
class MutableMatrix {
    private double[][] data;

    // Ctor
    public MutableMatrix(double[][] data) {
        this.data = data;
    }

    // add
    public MutableMatrix add(MutableMatrix matrix) {
        if (this.data.length != matrix.data.length || this.data[0].length != matrix.data[0].length) {
            throw new IllegalArgumentException("\33[31mMatrices must have the same dimensions.\33[0m");
        }
        for (int i = 0; i < this.data.length; i++) {
            for (int j = 0; j < this.data[i].length; j++) {
                this.data[i][j] += matrix.data[i][j];
            }
        }
        return this;
    }
    // subtract
    public MutableMatrix subtract(MutableMatrix matrix) {
        if (this.data.length != matrix.data.length || this.data[0].length != matrix.data[0].length) {
            throw new IllegalArgumentException("\33[31mMatrices must have the same dimensions.\33[0m");
        }
        for (int i = 0; i < this.data.length; i++) {
            for (int j = 0; j < this.data[i].length; j++) {
                this.data[i][j] -= matrix.data[i][j];
            }
        }
        return this;
    }
    // multiply
    public MutableMatrix multiply(MutableMatrix matrix) {
        if (this.data[0].length != matrix.data.length) { // mxn * nxp
            throw new IllegalArgumentException("\33[31mInvalid dimensions for multiplication.\33[0m");
        }
        double[][] result = new double[this.data.length][matrix.data[0].length]; // mxp
        for (int i = 0; i < this.data.length; i++) { // rows of m
            for (int j = 0; j < matrix.data[0].length; j++) { // cols of p
                for (int k = 0; k < this.data[0].length; k++) { // cols of n
                    result[i][j] += this.data[i][k] * matrix.data[k][j];
                }
            }
        }
        this.data = result;
```

```
        return this;
    }

    // convert fromInmutable
    public static MutableMatrix fromInmutable(InmutableMatrix inmutableMatrix) {
        return new MutableMatrix(inmutableMatrix.getData());
    }

    public double[][] getData() {
        return data; // return reference to internal data, changes to this will change the matrix
    }
}
```

4.2 不变类

```
class InmutableMatrix {
    private final double[][] data;

    // Ctor
    public InmutableMatrix(double[][] data) {
        // deep copy of data
        this.data = new double[data.length][data[0].length];
        for (int i = 0; i < data.length; i++) {
            System.arraycopy(data[i], 0, this.data[i], 0, data[i].length);
        }
    }

    // add
    public InmutableMatrix add(InmutableMatrix matrix) {
        if (this.data.length != matrix.data.length || this.data[0].length != matrix.data[0].length) {
            throw new IllegalArgumentException("\33[31mMatrices must have the same dimensions.\33[0m");
        }
        double[][] result = new double[this.data.length][this.data[0].length];
        for (int i = 0; i < this.data.length; i++) {
            for (int j = 0; j < this.data[i].length; j++) {
                result[i][j] = this.data[i][j] + matrix.data[i][j];
            }
        }
        return new InmutableMatrix(result); // return new instance
    }
    // subtract
    public InmutableMatrix subtract(InmutableMatrix matrix) {
        if (this.data.length != matrix.data.length || this.data[0].length != matrix.data[0].length) {
            throw new IllegalArgumentException("\33[31mMatrices must have the same dimensions.\33[0m");
        }
        double[][] result = new double[this.data.length][this.data[0].length];
        for (int i = 0; i < this.data.length; i++) {
            for (int j = 0; j < this.data[i].length; j++) {
                result[i][j] = this.data[i][j] - matrix.data[i][j];
            }
        }
```

```
37          return new InmutableMatrix(result);
38      }
39      // multiply
40      public InmutableMatrix multiply(InmutableMatrix 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 InmutableMatrix(result);
53      }
54
55      // convert fromMutable
56      public static InmutableMatrix fromMutable(MutableMatrix mutableMatrix) {
57          return new InmutableMatrix(mutableMatrix.getData());
58      }
59
60      public double[][] getData() {
61          InmutableMatrix im = new InmutableMatrix(data); // return new instance with
same data
62          return im.data;
63      }
64  }
```