SE/CS 2S03: Principles of Programming

Due on Oct. 29th

Dr. Jacques Carette

1 Goals

The goals of this assignment are:

1. get some understanding of records and arrays
2. write tests for your code (in a clever way)

2 The Task

This assignment involves writing several short routines. They will involve writing several classes. For all matrices, use `long` as the representation of the integer type (i.e. all matrices will be matrices of integers).

2.1 Six main classes

1. Create a `Matrix3x3flat` class which implements a 3×3 matrix using a single record with 9 fields.
2. Create a `Matrix3x3rc` class which implements a 3×3 matrix using a record of 3 rows; each row should be a record of 3 values.
3. Create a `Matrix3x3cr` class which implements a 3×3 matrix using a record of 3 columns; each column should be a record of 3 values.
4. Create a `MatrixArrayFlat` class which implements an $n \times n$ matrix using a 1D `Array`.
5. Create a `MatrixArrayRC` class which implements a $n \times n$ matrix using an `Array` of rows of `Array`s of values.
6. Create a `MatrixArrayCR` class which implements a $n \times n$ matrix using an `Array` of columns of `Array`s of values.

Important: all your classes should be part of a `cs2s03` package. [Read that package name again, you probably misread it]. This includes the extra code I gave you (see below).

2.2 Constructors

For each of these 6, provide a constructor which takes as input a single `Array` with 9 elements (and fails otherwise) to fill things in. This array is to be interpreted *row-wise*. In other words, the Array $[1, 2, 3, 4, 5, 6, 7, 8, 9]$ corresponds to the matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

For the 3 Array-based methods, provide a *second* contructor, which takes as input an integer n, and an array of size $n \times n$ (which throws an exception if this is not the case) to create the matrix. See the accompanying code for the exception code.

For `Matrix3x3cr` and `Matrix3x3rc`, create and use nested classes (call them `Row3` and `Column3`) for the rows and columns. Make these nested classes `private`.

2.3 `smultiply` method

1. For all 6, provide a `smultiply` method, which takes a single integer argument i, which computes the product $A^i = A \times A \times \ldots \times A$, where A is the matrix created by your constructor. Note that A^0 is the identity matrix and $A^1 = A$. It should throw an exception if $i < 0$.
2. This method should return a *new* matrix, of the same class as the current one. The easiest way to do this is to create new methods that allow you to *copy* a matrix (and also a method to create an identity matrix).
3. For the 3 Array-based versions, your `smultiply` method should work on arbitrarily-sized $n \times n$ matrices, not just 3×3.

2.4 Testing

You will also need to:

- in a new `Testing` class, create ten test matrices (as 9 element Arrays, aka valid input for the main constructor of your main classes). Make them public fields and call them `m01`, `m02`, ..., `m10`. One such matrix A should be such that A is not everywhere 0, but A^2 is.
- call each of the 6 methods on all 10 matrices, with i ranging from -1 to 3, and verify that they give the right answer (i.e. using JUnit).
- call each pair of methods on all 10 matrices, for i from -1 to 3, and make sure that each routine pairwise give the same answer.

2.5 Notes

- Yes, that means $6 \times 10 \times 5 = 300$ plus $6 \times 6 \times 10 \times 4 = 1800$ test cases. Automate this!
- Yes, the code in the first 3 versions will look alike a lot, and yet be subtly different. That's part of the learning objective of this assignment; even though this is clear 'in theory', seeing it (and doing it) in practice is quite enlightening.
- The same is true for the next 3 versions as well.
- The point of the constructor for exactly 9 entries (even for the Array-based classes) is to make the testing uniform.
- You may add a method to your classes to return a flat Array representation *for testing purposes only.* This is not the only way to do this, but it is rather convenient.
- Just to be very precise, your codes should look like:

  1.
     ```
     public class Matrix3x3flat {
         private Record9 mat;
     ```

     For this one, you may "explode" the record into 9 fields, that will also be considered correct (but is not as nice).

  2.
     ```
     public class Matrix3x3rc {
         private Row3 mat;
     ```

where `Row3` is a nested private class containing columns (of values).

3.
```
public class Matrix3x3cr {
    private Column3 mat;
```

where `Column3` is a nested private class containing rows (of values).

4.
```
public class MatrixArrayFlat {
    private long [] mat;
```

```
public class MatrixArrayRC {
    private long [] [] mat;
```

6.
```
public class MatrixArrayCR {
    private long [] [] mat;
```

Yes, this is the same as above, but will be interpreted differently by your code.

3 Submission Requirements

- A *single* zip file called `a3_<student_number>.zip` containing all your java files, including your JUnit test files.
- Make sure your classes are named as above.
- Extra files are OK.

4 Marking Scheme

- Programs which do not compile will be given a mark of 0, no matter how *close* your code might be to the correct answer.
- The code will be worth 60%, the tests 30%, style 10%.

5 Bonus

Each one of these will be worth extra marks:

- (easy) Implement your matrices using `BigInteger` instead of long. Due with assignment.
- (easy) Find a way to iterate over each of the classes, so that the tests contain a loop over 6 classes, within which is a loop over 10 arrays, within which is a loop from -1 to 3 to do the first 300 tests. Write loops in the same vein for the next 1800 tests as well. Due with assignment.
- (hard) Implement your matrices using a *generic ring type* instead of `long`. See Wikipedia for the definition of a ring. Due 2 weeks later.
- (very hard) Implement a generator (in any language, but more marks for Haskell/Scala) for this assignment which has a single method (i.e. there should not be 6 cases) that is parametrized by the choices (i.e. record/Array, flat/rc/cr, and dimension). Note it is significantly easier to have this generator take the dimension as a parameter [i.e. in theory you could generate record-based code for 10x10 matrices!]. Make sure your code works properly for 1x1 matrices too. Using an AST (rather than strings) is definitely preferred. This bonus is not due until **Dec. 3rd**. If you are going to attempt this, please speak with me first, as most people misunderstood what I meant last year.