LAB ASSIGNMENT IV

Instructions

- This assignment must be done with ***eclipse IDE***, go through the ***eclipse IDE*** *handout* if you find difficulties.
- Do utilize the JDK documentation and e-books for any doubts.

Aggregation

If a class have an entity reference, it is known as Aggregation. Aggregation represents 'HAS-A' relationship. Consider a situation, Employee object contains many information's such as id, name, email ID etc. It contains one more object named address, which contains its own information's such as city, state, country, zip code etc. as given below.

```
class Employee{
int id;
String name;
Address address;//Address is a class
...
}
```

In such case, Employee has an entity reference address, so relationship is Employee 'HAS-A' address.

Why use Aggregation?

- For Code Reusability.

Embedding object inside a class

- Now that we have created a Fraction class we can use it to construct a larger class.
- We will define a class for working with matrix of fractions. In particular a 3x3 matrix where each entity is a fraction.

Important Instructions

- Implement each functionality, test it out with a driver class and then the next functionality with its driver, so on and so forth.

  Your mantra should be: One functionality at a time

- Try to reuse the existing code to the best possible extent. Don't reinvent the wheel. You already have class that implements methods to manipulate fractions. Use them.

```
------------MatrixFraction.java-------------

public class MatrixFraction {
  private Fraction[][] matrixFrac;

  // Constructor section

  MatrixFraction() {  // default constructor - initializes to identity
matrix
      matrixFrac = new Fraction[3][3]; // First create array of 3x3 size
      matrixFrac[0][0] = new Fraction(1,1); // Then populate the matrix
      matrixFrac[0][1] = new Fraction(0,1);
      matrixFrac[0][2] = new Fraction(0,1);
      matrixFrac[1][0] = new Fraction(0,1);
      matrixFrac[1][1] = new Fraction(1,1);
      matrixFrac[1][2] = new Fraction(0,1);
      matrixFrac[2][0] = new Fraction(0,1);
      matrixFrac[2][1] = new Fraction(0,1);
      matrixFrac[2][2] = new Fraction(1,1);
  }

  MatrixFraction(Fraction f00, Fraction f01, Fraction f02,
                 Fraction f10, Fraction f11, Fraction f12,
                 Fraction f20, Fraction f21, Fraction f22) {
      matrixFrac = new Fraction[3][3];
      matrixFrac[0][0] = f00;
      matrixFrac[0][1] = f01;
      matrixFrac[0][2] = f02;
      matrixFrac[1][0] = f10;
      matrixFrac[1][1] = f11;
      matrixFrac[1][2] = f12;
      matrixFrac[2][0] = f20;
      matrixFrac[2][1] = f21;
      matrixFrac[2][2] = f22;
  }

  /** To set the values of all the elements of the matrix */
```

```
public void setMatrix(Fraction f00, Fraction f01, Fraction f02,
                      Fraction f10, Fraction f11, Fraction f12,
                      Fraction f20, Fraction f21, Fraction f22) {
     // Your code here
}

/** To set the value of a particular element of the matrix */
public void setElement(Fraction frac, int row, int column) {
     // Your code here
}

/** To get the value of a particular element of the matrix */
public Fraction getElement(int row, int column) {
     // Your code here
}

/** To compute determinant */
public int determinant() {
     // Your code here
}

/** To compute transpose (non-mutable) */
public MatrixFraction transpose() {
     // Your code here
}

/** Check if this is an identity matrix */
public boolean isIdentity() {
    // Your code here
}

/** Check if this is an upper triangular matrix */
public boolean isUpperTriangular() {
    // Your code here
}

/** Check if this is a lower triangular matrix */
public boolean isLowerTriangular() {
    // Your code here
}
```

```
  /** Check if two matrices are same */
  public boolean isSame(MatrixFraction mFrac) {
      // Your code here
  }

  /** To compute the sum of two matrices */
  public MatrixFraction add(MatrixFraction mFrac) {
      // Your code here
  }

  /** To compute the difference between two matrices */
  public MatrixFraction subtract(MatrixFraction mFrac) {
      // Your code here
  }

  /** To compute the product of two matrices */
  public MatrixFraction multiply(MatrixFraction mFrac) {
      // Your code here
  }

  /** To compute a matrix with a scalar */
  public MatrixFraction scalarMultiply(int scalar) {
      // Your code here
  }

  /** To compute the inverse of a matrix */
  public MatrixFraction scalarMultiply() {
      // Your code here
  }
}
---------------------------------------------
```

- Implement some more driver classes
  - to show commutativity property does not hold
  - to show associative property holds
  - multiplication of a matrix with its inverse results in identity matrix
  - Any other interesting properties about matrices
- Generate the documentation using javadoc