Programming and Data Structures
Assignment 5: Generics

Objectives of the assignment

Students should demonstrate the following abilities:

1. Create a generic interface Arithmetic<E> and a generic class GenericMatrix<E>
2. Create a generic method printMatrixOperation()
3. Use Java generic class **`ArrayList`**
4. Test the generic interface, class and method using generic instantiation in a test program

---

1. Create the generic interface **`Arithmetic<E>`** as described in the UML diagram below:

"interface" Arithmetic<E>
+add(E item): E
+subtract(E item): E
+multiply(E item): E
+divide(E item): E

2. Create the class **`Rational`** to represent fractions. The class implements the interface **`Arithmetic`** for the type **`Rational`**. The class is described in the UML diagram below:

Rational	**Description**
-numerator: int	Value of the fraction numerator
-denominator: int	Value of the fraction denominator
+Rational()	Initializes numerator to 0 and denominator to 1
+Rational(int n, int d)	Initializes numerator to n and denominator to d
+Rational(String s)	Extracts the value of numerator and denominator from a string in the format "\\d+/\\d+"
+add(Rational f): Rational	Adds f to this fraction and returns the result as a reduced fraction
+subtract(Rational f): Rational	Subtracts f from this fraction and returns the

	result as a reduced fraction
+multiply(Rational f): Rational	Multiplies this fraction by f and returns the result as a reduced fraction
+divide(Rational f): Rational	Divides this fraction by f and returns the result as a reduced fraction
toString(): String	Returns the string "numerator/denominator". If the denominator is equal to 1, it should return only the numerator as a string, and if the numerator is 0, it should return "0"
-reduce(): void	Reduces the fraction by dividing the numerator and the denominator by their GCD (Greatest Common Divisor)
+gcd(int m, int n): int	Static recursive method that calculates the greatest common divisor of m and n using Euclid's algorithm.

3. Create the class **Complex** to represent complex numbers. The class implements the interface **Arithmetic** for the type **Complex**. The class is described in the UML below:

Complex	**Description**
-real: int	Value of the real part
-imaginary: int	Value of the imaginary part
+Complex()	Initializes real and imaginary to 0
+Complex(int r, int im)	Initializes real to r and imaginary to im
+add(Complex c): Complex	Adds c to this complex number and returns the result as a complex number
+subtract(Complex c): Complex	Subtracts c from this complex number and returns the result as a complex number
+multiply(Complex c): Complex	Multiplies this complex number by c and returns the result as a complex number

+divide(Complex c): Complex	Divides this complex number by c and returns the result as a complex number
toString(): String	Returns the string **("real + imaginary i").** If real is equal to 0, it should return **"(imaginary i)".** If imaginary is 0, it should return only **"(real)"** and if both are zero, it returns "0". If imaginary is 1, it should return **"(real + i)".**

4. Create a class **GenericMatrix<E extends Arithmetic>** to hold a two-dimensional array **matrix** of type **E** using **ArrayLists**. The class is described in the UML diagram below:

GenericMatrix<E>	**Description**
-matrix : ArrayList<ArrayList<E>>	Two-dimensional array list to hold the elements of the matrix
+GenericMatrix(ArrayList<E[][] input)	Creates the array list **matrix** and makes a deep copy of the array **input** to **matrix**
+GenericMatrix(ArrayList<ArrayList<E>> input)	Creates the array list **matrix** and makes a deep copy of the array list **input** to **matrix**
+get(int r, int c): E	Return the element at row **r** and column **c** in **matrix**
+set(int r, int c, E value): E	Sets the element at row **r** and column **c** to **value**
+rows(): int	Returns the number of rows in **matrix**
+columns(): int	Returns the number of **columns** in matrix
+add(GenericMatrix<E> gm): GenericMatrix<E>	Adds the matrices **matrix** and **gm.matrix** and returns the sum as a **matrix**
+subtract(GenericMatrix<E> gm): GenericMatrix<E>	Subtracts **gm.matrix** from **matrix** and returns the result a matrix

+multiply(GenericMatrix<E> gm): GenericMatrix<E>	Multiplies **matrix** and **gm.matrix** and returns the result as a matrix

5. The test program file, **`Test.java`**, is provided at the end of this document to test your classes. However, you need to define the method **`printMatrixOperation()`** to print the matrix operation in the following order:

   `first_matrix operation second_matrix = result_matrix`

   and as shown in the program output below.

```
Rational Matrices
Addition
|1/2 1/2 1/2 |   |1/2 1/2 1/2 |   |1 1 1 |
|1/2 1/2 1/2 | + |1/2 1/2 1/2 | = |1 1 1 |
|1/2 1/2 1/2 |   |1/2 1/2 1/2 |   |1 1 1 |
Subtraction
|1/2 1/2 1/2 |   |1/2 1/2 1/2 |   |0 0 0 |
|1/2 1/2 1/2 | - |1/2 1/2 1/2 | = |0 0 0 |
|1/2 1/2 1/2 |   |1/2 1/2 1/2 |   |0 0 0 |
Multiplication
|1/2 1/2 1/2 |   |1/2 1/2 1/2 |   |3/4 3/4 3/4 |
|1/2 1/2 1/2 | * |1/2 1/2 1/2 | = |3/4 3/4 3/4 |
|1/2 1/2 1/2 |   |1/2 1/2 1/2 |   |3/4 3/4 3/4 |
Complex Matrices
Addition
|1+2i 1+2i 1+2i |   |1+2i 1+2i 1+2i |   |2+4i 2+4i 2+4i |
|1+2i 1+2i 1+2i | + |1+2i 1+2i 1+2i | = |2+4i 2+4i 2+4i |
|1+2i 1+2i 1+2i |   |1+2i 1+2i 1+2i |   |2+4i 2+4i 2+4i |
Subtraction
|1+2i 1+2i 1+2i |   |1+2i 1+2i 1+2i |   |0 0 0 |
|1+2i 1+2i 1+2i | - |1+2i 1+2i 1+2i | = |0 0 0 |
|1+2i 1+2i 1+2i |   |1+2i 1+2i 1+2i |   |0 0 0 |
Multiplication
|1+2i 1+2i 1+2i |   |1+2i 1+2i 1+2i |   |-9+12i -9+12i -9+12i |
|1+2i 1+2i 1+2i | * |1+2i 1+2i 1+2i | = |-9+12i -9+12i -9+12i |
|1+2i 1+2i 1+2i |   |1+2i 1+2i 1+2i |   |-9+12i -9+12i -9+12i |
```

```
public class Test {
  public static void main(String[] args) {
      Rational[][] A = {
         {new Rational("1/2"),new Rational("1/2"),new Rational("1/2")},
         {new Rational("1/2"),new Rational("1/2"),new Rational("1/2")},
         {new Rational("1/2"),new Rational("1/2"),new Rational("1/2")}
      };

      // Creating rational matrices
      GenericMatrix<Rational> rationalMatrix1 = new GenericMatrix<>(A);
      GenericMatrix<Rational> rationalMatrix2 = new GenericMatrix<>(A);
      GenericMatrix<Rational> rationalMatrix3;

      // Operations on rational matrices
      System.out.println("Rational Matrices");
      rationalMatrix3 = rationalMatrix1.add(rationalMatrix2);
      System.out.println("Addition");
      printMatrixOperation(rationalMatrix1, rationalMatrix2,
                                      '+', rationalMatrix3);

      rationalMatrix3 = rationalMatrix1.subtract(rationalMatrix2);
      System.out.println("Subtraction");
      printMatrixOperation(rationalMatrix1, rationalMatrix2,
                                      '-', rationalMatrix3);

      rationalMatrix3 = rationalMatrix1.multiply(rationalMatrix2);
      System.out.println("Multiplication");
      printMatrixOperation(rationalMatrix1, rationalMatrix2,
                                      '*', rationalMatrix3);


      Complex[][] AC = {
       {new Complex(1,2), new Complex(1,2), new Complex(1,2)},
       {new Complex(1,2), new Complex(1,2), new Complex(1,2)},
       {new Complex(1,2), new Complex(1,2), new Complex(1,2)}
       };

      // Creating complex matrices
      GenericMatrix<Complex> complexMatrix1 = new GenericMatrix<>(AC);
      GenericMatrix<Complex> complexMatrix2 = new GenericMatrix<>(AC);
      GenericMatrix<Complex> complexMatrix3;

       // Operations on complex matrices
      System.out.println("Complex Matrices");
      complexMatrix3 = complexMatrix1.add(complexMatrix2);
      System.out.println("Addition");
      printMatrixOperation(complexMatrix1, complexMatrix2,
                                     '+', complexMatrix3);
```

```
        complexMatrix3 = complexMatrix1.subtract(complexMatrix2);
        System.out.println("Subtraction");
        printMatrixOperation(complexMatrix1, complexMatrix2,
                                        '-', complexMatrix3);

        complexMatrix3 = complexMatrix1.multiply(complexMatrix2);
        System.out.println("Multiplication");
        printMatrixOperation(complexMatrix1, complexMatrix2,
                                         '*', complexMatrix3);

    }
    // Generic method to print matrix operation result
    public static <E extends Arithmetic<E>>
                 void printMatrixOperation(GenericMatrix<E> m1,
                                           GenericMatrix<E> m2,
                                           char operation,
                                           GenericMatrix<E> result) {

            ---------  Write the body of the method ----------
    }
}
```

Submit the following Java files on courseSite:

- **Arithmetic.java**
- **GenericMatrix.java**,
- **Rational.java** ,
- **Complex.java**, and
- updated **Test.java**