```
//This is the fraction class, which contains
//   methods for simple fraction math
//Author: Caspian Peavyhouse
//CS101-02
//11/19/2014

/*
     Data Table for Fraction

Variable or Constant      Purpose
_____________________     ___________________
args                      parameter for Fraction
numerator                 Stores the value of the numerator of the fraction as an int

denominator               Stores the value of the denominator of the fraction as an int

undefined                 Stores the boolean value of true if a fraction is undefined

DEFAULT_NUMERATOR         The constant int numerator value of the default fraction
DEFAULT_DENOMINATOR       The constant int denominator value of the default fraction

*/

public class Fraction
{
   private int numerator;
   private int denominator;
   private boolean undefined;
   public static final int DEFAULT_NUMERATOR = 1;
   public static final int DEFAULT_DENOMINATOR = 1;

   /*
    *Fraction constructor initializes numerator and denominator
    *    reduces fraction, checks sign, and checks if undefined
    *
    *The second constructor makes a fraction with a default value of 1 / 1
    *
    *algorithm:
    *int divisor <--Fraction.greatestCommonDivisor(numerator, denominator)
    *this.numerator <-- (numerator / divisor)
    *this.denominator <-- (denominator / divisor)
    *if (this.denominator < 0)
    *{
    *    this.denominator *= -1
    *    this.numerator *= -1
    *}
    *if (denominator == 0)
    *    this.undefined <-- true
    *else
    *    this.undefined <-- false
   */

   public Fraction(int numerator, int denominator)
   {
      int divisor = Fraction.greatestCommonDivisor(numerator, denominator);
      this.numerator = (numerator / divisor);
      this.denominator = (denominator / divisor);
      if (this.denominator < 0)
      {
         this.denominator *= -1;
         this.numerator *= -1;
      }
      if (denominator == 0)
         this.undefined = true;
      else
         this.undefined = false;

   }//Fraction(1)
```

```
public Fraction()
{
   this.numerator = DEFAULT_NUMERATOR;
   this.denominator = DEFAULT_DENOMINATOR;
   this.undefined = false;
}//Fraction(2)


/*
 *add returns a fraction from the addition of two fractions
 *
 *algorithm:
 *int newDenominator <-- (this.getDenominator() * addend.getDenominator())
 *int convertedNumerator1 <-- (this.getNumerator() * addend.getDenominator())
 *int convertedNumerator2 <-- (addend.getNumerator() * this.getDenominator())
 *int newNumerator <-- convertedNumerator1 + convertedNumerator2
 *Fraction newFraction <--new Fraction(newNumerator, newDenominator)
 *return newFraction
*/
/*
  Data Table for add

   Variable or Constant      Purpose
   ____________________      ___________________
   args                      parameter for Fraction
   addend                    Fraction object being added
   newDenominator            Stores the value of the denominator of the new fraction

   convertedNumerator1       Stores the value of the first modified numerator
   convertedNumerator1       Stores the value of the second modified numerator

   newNumerator              Stores the value of the sum of the new numerators

*/

public Fraction add(Fraction addend)
{
   int newDenominator = (this.getDenominator() * addend.getDenominator());
   int convertedNumerator1 = (this.getNumerator() * addend.getDenominator());
   int convertedNumerator2 = (addend.getNumerator() * this.getDenominator());
   int newNumerator = convertedNumerator1 + convertedNumerator2;

   Fraction newFraction = new Fraction(newNumerator, newDenominator);
   return newFraction;
}//add


/*
 *subtract returns a fraction from the subtraction of two fractions
 *
 *algorithm:
 *int newDenominator <-- (this.getDenominator() * subtrahend.getDenominator())
 *int convertedNumerator1 <-- (this.getNumerator() * subtrahend.getDenominator())
 *int convertedNumerator2 <-- (subtrahend.getNumerator() * this.getDenominator())
 *int newNumerator <-- convertedNumerator1 - convertedNumerator2
 *Fraction newFraction <--new Fraction(newNumerator, newDenominator)
 *return newFraction
*/
/*
  Data Table for subtract

   Variable or Constant      Purpose
   ____________________      ___________________
   args                      parameter for Fraction
   subtrahend                Fraction object being subtracted
   newDenominator            Stores the value of the denominator of the new fraction
```

```
   convertedNumerator1       Stores the value of the first modified numerator
   convertedNumerator1       Stores the value of the second modified numerator

   newNumerator              Stores the value of the difference of the new numerators

*/

public Fraction subtract(Fraction subtrahend)
{
   int newDenominator = (this.getDenominator() * subtrahend.getDenominator());
   int convertedNumerator1 = (this.getNumerator() * subtrahend.getDenominator());
   int convertedNumerator2 = (subtrahend.getNumerator() * this.getDenominator());
   int newNumerator = convertedNumerator1 - convertedNumerator2;

   Fraction newFraction =new Fraction(newNumerator, newDenominator);
   return newFraction;
}//subtract


/*
 *multiply returns a fraction from the multiplication of two fractions
 *
 *algorithm:
 *int newDenominator <-- (this.getDenominator() * multiplier.getDenominator())
 *int newNumerator <-- (this.getNumerator() * multiplier.getNumerator())
 *Fraction newFraction <--new Fraction(newNumerator, newDenominator)
 *return newFraction
*/
/*
  Data Table for multiply

   Variable or Constant      Purpose
   ____________________      ___________________
   args                      parameter for Fraction
   multiplier                Fraction object being multiplied
   newDenominator            Stores the value of the denominator of the new fraction

   newNumerator              Stores the value of the product of the two numerators

*/
public Fraction multiply(Fraction multiplier)
{
   int newDenominator = (this.getDenominator() * multiplier.getDenominator());
   int newNumerator = (this.getNumerator() * multiplier.getNumerator());

   Fraction newFraction = new Fraction(newNumerator, newDenominator);
   return newFraction;
}//multiply


/*
 *divide returns a fraction from the division of two fractions
 *
 *algorithm:
 *flipDivisor <-- divisor.reciprocal()
 *Fraction newFraction <-- new Fraction()
 *newFraction <-- this.multiply(flipDivisor)
 *return newFraction
*/
/*
  Data Table for divide

   Variable or Constant      Purpose
   ____________________      ___________________
   args                      parameter for Fraction
   divisor                   Fraction object being divided by
   newDenominator            Stores the value of the denominator of the new fraction
```

```
   newNumerator              Stores the value of the numerator of the new fraction
*/
public Fraction divide(Fraction divisor)
{
   Fraction flipDivisor = divisor.reciprocal();
   Fraction newFraction = new Fraction();
   newFraction = this.multiply(flipDivisor);
   return newFraction;
}//divide


/*
 *reciprocal returns the value of the inverted fraction
 *
 *algorithm:
 *int newNumerator <-- this.getDenominator()
 *int newDenominator <-- this.getNumerator()
 *Fraction newFraction <-- new Fraction(newNumerator, newDenominator)
 *return newFraction
*/
/*
  Data Table for reciprocal

   Variable or Constant      Purpose
   _____________________     ___________________
   args                      parameter for Fraction
   newDenominator            Stores the value of the denominator of the new fraction

   newNumerator              Stores the value of the numerator of the new fraction
*/
public Fraction reciprocal()
{
   int newNumerator = this.getDenominator();
   int newDenominator = this.getNumerator();
   Fraction newFraction = new Fraction(newNumerator, newDenominator);
   return newFraction;
}//reciprocal


/*
 *greatestCommonDivisor returns the largest int that can be divide into
 *    two inputs evenly
 *
 *algorithm:
 *inputOne <-- Math.abs(inputOne)
 *inputTwo <-- Math.abs(inputTwo)

 *int smaller <--  Math.min(inputOne, inputTwo)
 *int greatestCommonDivisor <--

 *for (int index = 0; index <= smaller; index++)

 *return greatestCommonDivisor
 *inputOne <-- Math.abs(inputOne)
 *inputTwo <-- Math.abs(inputTwo)

 *int smaller <--  Math.min(inputOne, inputTwo)
 *int greatestCommonDivisor <-- 1

 *for (int index = 0; index <= smaller; index++)
 *   if ((inputOne % index) == 0 && (inputTwo % index) == 0)
 *      greatestCommonDivisor <-- index
 *return greatestCommonDivisor
*/
/*
  Data Table for multiply

   Variable or Constant      Purpose
```

```
   ____________________      ___________________
   args                      parameter for Fraction
   inputOne                  The first input value
   inputTwo                  The second input value
   smaller                   The smaller of the two inputs
*/
public static int greatestCommonDivisor(int inputOne, int inputTwo)
{
   inputOne = Math.abs(inputOne);
   inputTwo = Math.abs(inputTwo);

   int smaller =  Math.min(inputOne, inputTwo);
   int greatestCommonDivisor = 1;

   for (int index = 1; index <= smaller; index++)
      if ((inputOne % index) == 0 && (inputTwo % index) == 0)
         greatestCommonDivisor = index;
   return greatestCommonDivisor;
}//greatestCommonDivisor


/*
 *getNumerator returns the value of the numerator
 *
 *algorithm:
 *return this.numerator
*/

public int getNumerator()
{
   return this.numerator;
}//getNumerator


/*
 *getDenominator returns the value of the denominator
*/

public int getDenominator()
{
   return this.denominator;
}//getDenominator


/*
 *toString returns a string displaying the fraction
 *
 *algorithm:
 *String output

 *if (this.undefined == true)
 *    output <-- "Undefined"
 *else
 *    output <-- "" + this.getNumerator() + " / " + this.getDenominator()
 *return output

*/

public String toString()
{
   String output;

   if (this.undefined == true)
      output = "Undefined";
   else
      output = "" + this.getNumerator() + " / " + this.getDenominator();

   return output;
```

```
   }//toString

}//class Fraction
```