```
/*-------------------------------------------------------------------------------------------------
 * Author:      Dan Cassidy
 * Date:        2015-07-14
 * Assignment:  HW2-1
 * Source File: Rational.java
 * Language:    Java
 * Course:      CSCI-C 490, Android Programming, MoWe 08:00
-------------------------------------------------------------------------------------------------*/

/**
 * Small class that represents rational numbers in accordance with homework assignment 2-1. By
 * default, the numerator is set to 0 and the denominator is set to 1.
 *
 * @author Dan Cassidy
 */
public class Rational
{
    private int numerator = 0;
    private int denominator = 1;

    /**
     * Default constructor.
     */
    public Rational()
    {
        // Do nothing, numerator and denominator are already set to 0 and 1 respectively.
    }

    /**
     * 1-parameter constructor. Takes a whole number and creates a new Rational object based upon
     * that. This means that the denominator will be 1.
     *
     * @param wholeNumber The number that will be stored in the numerator.
     */
    public Rational(int wholeNumber)
    {
        this.setNumerator(wholeNumber);
        // Don't have to set denominator because it's 1 by default.
    }

    /**
     * 2-parameter constructor. Takes two arguments that will be the numerator and denominator of
     * the resultant Rational object.
     *
     * @param numerator The number that will be the numerator.
     * @param denominator The number that will be the denominator.
     */
    public Rational(int numerator, int denominator)
    {
        this.setNumerator(numerator);
        this.setDenominator(denominator);
    }

    // BEGIN GETTERS AND SETTERS -->
    public int getDenominator()
    {
        return this.denominator;
    }

    /**
```

```
61          * @param denominator The number to set the denominator to.
62          * @throws IllegalArgumentException if <b>denominator</b> is 0.
63          */
64         public void setDenominator(int denominator)
65         {
66             if (denominator == 0)
67                 throw new IllegalArgumentException("Cannot have a denominator of 0.");
68 
69             this.denominator = denominator;
70             this.simplify();
71         }
72 
73         public int getNumerator()
74         {
75             return this.numerator;
76         }
77 
78         /**
79          * @param numerator The number to set the numerator to.
80          */
81         public void setNumerator(int numerator)
82         {
83             this.numerator = numerator;
84             this.simplify();
85         }
86         // <-- END GETTERS AND SETTERS
87 
88         /**
89          * Addition method that takes two Rational objects and adds them together.
90          *
91          * @param r1 The first of two Rational objects that will be added together.
92          * @param r2 The second of two Rational objects that will be added together.
93          * @return Rational object, containing the result of the operation.
94          * @throws NullPointerException if <b>r1</b> or <b>r2</b> is null.
95          */
96         public static Rational add(Rational r1, Rational r2)
97         {
98             if (r1 == null || r2 == null)
99                 throw new NullPointerException();
100 
101             if (r1.denominator == r2.denominator)
102                 return new Rational(r1.numerator + r2.numerator, r1.denominator);
103             else
104                 return new Rational(r1.numerator * r2.denominator + r2.numerator * r1.denominator,
105                         r1.denominator * r2.denominator);
106         }
107 
108         /**
109          * Subtraction method that takes two Rational objects and subtracts the second from the first.
110          *
111          * @param r1 The first of two Rational objects. Will be subtracted from by the second.
112          * @param r2 The second of two Rational objects. Will subtract from the first.
113          * @return Rational object, containing the result of the operation.
114          * @throws NullPointerException if <b>r1</b> or <b>r2</b> is null.
115          */
116         public static Rational subtract(Rational r1, Rational r2)
117         {
118             if (r1 == null || r2 == null)
119                 throw new NullPointerException();
120 
```

```
121             if (r1.denominator == r2.denominator)
122                 return new Rational(r1.numerator - r2.numerator, r1.denominator);
123             else
124                 return new Rational(r1.numerator * r2.denominator - r2.numerator * r1.denominator,
125                         r1.denominator * r2.denominator);
126         }
127 
128         /**
129          * Multiplication method that takes two Rational objects and multiplies them together.
130          *
131          * @param r1 The first of two Rational objects that will be multiplied together.
132          * @param r2 The second of two Rational objects that will be multiplied together.
133          * @return Rational object, containing the result of the operation.
134          * @throws NullPointerException if <b>r1</b> or <b>r2</b> is null.
135          */
136         public static Rational multiply(Rational r1, Rational r2)
137         {
138             if (r1 == null || r2 == null)
139                 throw new NullPointerException();
140 
141             return new Rational(r1.numerator * r2.numerator, r1.denominator * r2.denominator);
142         }
143 
144         /**
145          * Division method that takes two Rational objects and divides the first by the second.
146          *
147          * @param r1 The first of two Rational objects. Will be divided by the second.
148          * @param r2 The second of two Rational objects. Will divide the first.
149          * @return Rational object, containing the result of the operation.
150          * @throws NullPointerException if <b>r1</b> or <b>r2</b> is null.
151          */
152         public static Rational divide(Rational r1, Rational r2)
153         {
154             if (r1 == null || r2 == null)
155                 throw new NullPointerException();
156 
157             return new Rational(r1.numerator * r2.denominator, r1.denominator * r2.numerator);
158         }
159 
160         /**
161          * Equals method that compares this object to another Rational object.
162          * @param other A Rational object that will be compared against to determine equality.
163          * @return boolean, representing whether this object is equal to the other object or not.
164          */
165         public boolean equals(Rational other)
166         {
167             if (other == null)
168                 return false;
169 
170             return (this.numerator * other.denominator == other.numerator * this.denominator);
171         }
172 
173         /**
174          * Override of the toString() method. Returns a String representation of the object.
175          * @return String containing either the numerator and denominator separated by a forward slash
176          * if denominator is not equal to 1, or just the numerator if the denominator is equal to 1.
177          */
178         public String toString()
179         {
180             if (this.denominator == 1)
```

```
181             return "" + this.numerator;
182
183         return this.numerator + "/" + this.denominator;
184     }
185
186     /**
187      * Simplifies the current Rational object (this) by putting any negative signs in the numerator
188      * and dividing both the numerator and the denominator by their greatest common divisor.
189      */
190     private void simplify()
191     {
192         if (this.denominator < 0)
193         {
194             this.numerator *= -1;
195             this.denominator *= -1;
196         }
197
198         if (this.denominator != 1)
199         {
200             int gcd = gcd(this.numerator < 0 ? this.numerator * -1 : this.numerator,
201                     this.denominator);
202             this.numerator /= gcd;
203             this.denominator /= gcd;
204         }
205     }
206
207     /**
208      * Iterative binary greatest common divisor (GCD) algorithm (Stein's algorithm) from
209      * <a href="https://en.wikipedia.org/wiki/Binary_GCD_algorithm">Wikipedia</a>.
210      *
211      * @param u The first of two non-negative numbers to find the GCD of.
212      * @param v The second of two non-negative numbers to find the GCD of.
213      * @return An integer representing the greatest common divisor of <b>u</b> and <b>v</b>.
214      * @throws IllegalArgumentException if either <b>u</b> or <b>v</b> are negative.
215      */
216     private int gcd(int u, int v)
217     {
218         if (u < 0 || v < 0)
219             throw new IllegalArgumentException("Arguments must be non-negative.");
220
221         // If either argument is 0, return the other argument.
222         if (u == 0)
223             return v;
224         if (v == 0)
225             return u;
226
227         // Find the greatest power of 2 dividing both u and v.
228         int shift;
229         for (shift = 0; ((u | v) & 1) == 0; ++shift)
230         {
231             u >>= 1;
232             v >>= 1;
233         }
234
235         // Make u odd.
236         while ((u & 1) == 0)
237             u >>= 1;
238
239         do
240         {
```

```
241             // Remove all factors of 2 in v.
242             while ((v & 1) == 0)
243                 v >>= 1;
244 
245             // Now u and v are both odd. Swap if necessary so u <= v, then set v = v - u.
246             if (u > v)
247             {
248                 int t = v;
249                 v = u;
250                 u = t;
251             }
252             v = v - u;
253         } while (v != 0);
254 
255         // Restore common factors of 2.
256         return u << shift;
257     }
258 }
259 
```