P. Thiemann, L. Fennell Sommersemester 2017

Programmieren in Java

`http://proglang.informatik.uni-freiburg.de/teaching/java/2017/`

rationals

Rationale Zahlen

Woche 06 Aufgabe 2/3

Herausgabe: 2017-05-29
Abgabe: 2017-06-17

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `rationals`
Package `rationals`
Klassen

Rational
`public Rational(long nominator, long denominator)`
`public Rational add(Rational r)`
`public Rational multiply(Rational r)`
`public Rational invert(Rational r)`
`public double toDouble()`
`public long getNominator()`
`public long getDenominator()`
`@Override public boolean equals(Object r)`

Rationale Zahlen sind die Teilmenge der Reellen Zahle, die sich durch einen Bruch aus zwei ganzen Zahlen darstellen lassen.

`https://de.wikipedia.org/wiki/Rationale_Zahl`

Implementieren Sie die Klasse `Rational`, deren Objekte rationale Zahlen darstellen. Repräsentiert werden diese durch zwei `long`-Werte, den Zähler (eng. nominator) und den Nenner (eng. denominator) der Zahl. Die `Rational`s haben gegenüber dem Fließkommatyp `double` den Vorteil, dass Berechnungen immer präzise ausgeführt werden können, so lange das Ergebnis nicht den Zahlenbereich von `long` verlässt.

Die Parameter der Konstruktors sind Zähler und Nenner des `Rational`s als `long`-Werte. Wenn der Nenner 0 ist, soll eine `IllegalArgumentException` geworfen werden. Eine rationale Zahl ist per se nicht eindeutig durch zwei `long`-Werte bestimmt (z.B. $\frac{1}{2} = \frac{2}{4} = \ldots$) Sorgen Sie deshalb dafür, dass die interne Darstellung der rationalen Zahl vollständig gekürzt und kanonisch ist.

Die Methoden `add`, `multiply` und `invert` ensprechen den gleichnamigen Rechenoperationen. Ist das Ergebnis der Operation nicht präzise als `Rational` darstellbar, oder ist eine Operation nicht definiert, soll eine `ArithmeticException` geworfen werden.

Die Methode `toDouble` gibt den `double`-Wert der Zahl zurück. (Dieser dieser entspricht natürlich unter Umständen nur ungefähr der rationalen Zahl.)

Die Methoden `getNominator` und `getDenominator` geben jeweils Zähler und Nenner der Zahl zurück. Achten Sie darauf, dass diese Methode für gleiche Zahlen auch das gleiche Ergebnis liefern.

Die Methode `equals` gibt `true` zurück genau dann wenn das Argument `other` ein `Rational` Objekt ist und der gleichen Zahl entspricht.

Im Skelett zu dieser Aufgabe finden sie die Klasse `RationalTestExamples` mit einigen Beispieltests.

```
package rationals;

import org.junit.Test;

import static org.junit.Assert.*;

public class RationalTestExamples {

    @Test
    public void testAddingPrecicely() {
        Rational r1 = new Rational(100000000001, 1);
        Rational r2 = new Rational(1, 2);
        assertEquals(new Rational(200000000011, 2), r1.add(r2));
    }

    @Test
    public void testUniqueNominator() {
        Rational r1 = new Rational(1, 2);
        Rational r2 = new Rational(2, 4);
        assertEquals(r1.getNominator(), r2.getNominator());
        assertEquals(r1.getDenominator(), r2.getDenominator());
    }

    @Test(expected = IllegalArgumentException.class)
    public void testThrowingWithDenominatorZero() {
        new Rational(1, 0);
    }
}
```

Das Beispiel demonstriert auch, wie Sie in JUnit auf das Werfen von Exceptions testen können.

Hinweise

- Zum Kürzen empfehlen wir den Algorithmus von Euklid

  `https://de.wikipedia.org/wiki/Euklidischer_Algorithmus#Beschreibung_durch_Pseudocode_2`

Auch nützlich:

`https://de.wikipedia.org/wiki/Kleinstes_gemeinsames_Vielfaches#Zusammenhang_zwischen_kgV_und_dem_gr.C3.B6.C3.9Bten_gemeinsamen_Teiler`

- Die Klassen `java.lang.Math` enthält Funktionen für Rechenoperationen ohne impliziten Überlauf.
- Um Literale vom Typ `long` zu Schreiben muss ihnen ein „l" hintenangestellt werden (z.B. `9999999999999999l`). Bei zu großen Zahlen, die nicht mehr in `int` passen, gibt es sonst eine Fehlermeldung vom Compiler.