Recitation 8

November 4, 2021

Topics to Recap

- Exceptions
- JUnit Tests
- HashMaps

Examples for Recap

- Writing a JUnit test suite for our good friend `PositiveInteger.java`.

Recitation Problem Set

Background

Please pull this week's recitation assignment into your IDE from this GitHub repo.

In the project, you will see the file `Fraction.java`. This declares the `Fraction` class, which (once completed) can be used to represent a fraction whose numerator and denominator are integers. The fraction can be proper or improper, and does not need to be reduced.

`Fraction` has the following methods (DO NOT COMPLETE THESE YET):

```
public Fraction(int a, int b)
```

- This is the constructor of the `Fraction` class. It takes the arguments `a` and `b` as the new fraction's numerator and denominator, respectively.

```
public int getNumerator()
public int getDenominator()
```

- These getter methods should return the `Fraction` object's numerator and denominator, respectively.

```
public void add(Fraction other)
```

- This method should add the fraction represented by `other` to the `Fraction` on which this is called, without any reduction (i.e. the denominator of the sum should be the least common multiple of the two original denominators).
- Examples:
  - If `frac1` is `2/3` and `frac2` is `1/2`, then `frac1` should be `7/6` after `frac1.add(frac2)`.

  - If `frac1` is `1/5` and `frac2` is `2/6`, then `frac1` should be `16/30` after `frac1.add(frac2)`.

```
public void reduceToLowestTerms()
```

- This method should reduce the `Fraction` on which it is called to its lowest terms, i.e. the greatest common factor of its numerator and denominator afterwards should be 1.
- Examples
  - If `frac` is `12/14`, then after `frac.reduceToLowestTerms()` it should be `6/7`.
  - If `frac` is `21/7`, then after `frac.reduceToLowestTerms()` it should be `3/1`.
  - If `frac` is `7/11`, then after `frac.reduceToLowestTerms()` it should be ... still `7/11`!

```
public double toDouble()
```

- This method should return the value of the `Fraction` in its decimal form.

Part 1

As we all know, we should not try dividing by zero, ESPECIALLY when working with computers.

Modify the constructor's declaration so that it can throw a `BadFractionException` if someone tries to create a `Fraction` with a denominator of `0`. *You will need to create this custom exception class!*

You may not modify the declarations of the other methods.

Part 2

Create a JUnit test class called `FractionTest`.

Based on the description of the `Fraction` class above, write a suite of unit tests using the following steps:

1. Write a new unit test that checks for a specific expected behavior of the class.
2. Run the test class (the new test should fail, because the method isn't implemented yet).
3. Write *just enough* code to make the test pass.
4. Run the tests again, and modify/add code until the new test passes.
5. Repeat until you have a fully functioning class, with complete testing coverage.

Through this process, you will be completing `Fraction` and `FractionTest` together.

This is a common practice in industry called Test Driven Development (TDD). Done properly, this approach to programming can help make your code more concise and efficient, and keep you from introducing sneaky bugs that would be more difficult to detect and fix later.

When coming up with your unit tests, try to think of a variety of scenarios in which the `Fraction` may be used. You do yourself a bigger service writing specific, meaningful tests, rather than just aiming for complete coverage.

A good place to start would be testing that you cannot create a `Fraction` that divides by zero!

Submission

Please submit your completed `Fraction.java` and `FractionTest.java` files to Gradescope.

NEW GROUPS:

Group #	Member 1	Member 2	Member 3	Member 4
Group 1	Li, Yunhe	Lee, Jaeyoung	Gallagher, John Manus	Biscaro, Denise
Group 2	Ng, Wai Chung	Wang, An-Jie	Bales, Elijah	Kallas Jatene, Rafael
Group 3	Guo, Zhaosen	Bernat, Kevin Bruno	Sha, Yumeng	Yu, Qingyu
Group 4	Kong, Rachel	Chen, Zheyi	Williams, Levester Randall	Cho, Suebin Grace
Group 5	Jiang, Yao	Wang, Liujia	Zhang, Miaoyan	Sheng, Xinyue
Group 6	Zhang, Minzheng	Rigas, Andrew	Liu, Jiayun	Shah, Rushabh
Group 7	Graham, Alexander Richard	Zhang, Zhihui	Choi, Jae Ho	He, Donglun
Group 8	Huang, Wenyi	Nguyen, Tai D	Pinheiro, Benjamin B	Wang, Yuanqi
Group 9	Cruz, Marye I	He, Ziyi	Zhang, Yihong	Xiao, Zijian
Group 10	Zhang, Han	Qiu, Chengzhuo	Sabri, Rita	Wu, Jeng-Ru
Group 11	Liu, Xinyue	Kim, Yunchae	Thenappan, Bala Sundar	Hu, Yuxin
Group 12	Tims, George	Ye, Huifang	Patel, Rishi	Cheema, Sardar Asfandy
Group 13	Wang, Kehan	Pizzico, Tyler R	Cai, Jialin	Chheda, Shagun Pritesh
Group 14	Ren, Yue	Xue, Mingxin	Mammadov, Elmar	Patel, Siddharth Bhagwanji
Group 15	Richmond, Christian	Hu, Lucy Qian	Liu, Shufan	Qiu, Xi

Group #	Member 1	Member 2	Member 3	Member 4
Group 16	Yiu, Hon-Cheung	Zhang, Yang	Schnall, Aaron Hewitt	Arguello-Gonzalez, Marcos Abraham
Group 17	Nojoomi, Radin	Chou, Randy	Kung, Ling-Hsin	Chen, Xiyue
Group 18	Lai, Qimei	Carnation, Kayla Rae	Lim, Xi Zhen	Pace, Benjamin Michael