##### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

##### «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

##### КАФЕДРА ВТ

##### ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

##### «Разработка класса для нестандартного типа данных» по дисциплине «Технологии разработки программного обеспечения»

##### Выполнили: студенты гр. АММ2-24 Атласюк Игорь Романович

##### Ириков Евгений Алексеевич

##### Проверил: к.т.н., доцент Кафедры ВТ Токарев Вадим Геннадьевич

##### Новосибирск 2024

## Содержание

[Постановка задачи 3](#_bookmark0)

[Выполнение работы 4](#_bookmark1)

[Результат работы 12](#_bookmark2)

[Приложение 13](#_bookmark3)

# Постановка задачи

# Разработать класс для представления нестандартного типа данных. Реализовать внутреннее представление данных, конструкторы, деструктор, методы ввода/вывода (сохранения в текстовый и двоичный поток), изменения содержимого отдельных элементов, операции, свойственные данному типу (например, сложение, умножение матриц, вычисление длины и угла вектора).

# Вариант 2 - Правильная дробь, представленная целой частью, числителем и знаменателем.

# Выполнение работы

# Для работы с дробью был создан отдельный класс Fraction.java, в котором описывается логика работы с ней.

# Класс содержит поля для хранения целой части, числителя и знаменателя, а также методы для выполнения математических операций с дробями (сложение, вычитание, умножение, деление).

# 

# Рисунок 1. Конструктор

# На рисунке 1 изображен конструктор класса Fraction, который создает объект дроби с указанными целой частью, числителем и знаменателем. Также показан метод-фабрика для создания экземпляра дроби. Прежде чем создать дробь, проверяет её корректность с помощью метода isValid(). Если данные некорректные, вызывает исключение IllegalArgumentException.

# 

# Рисунок 2. Сложение и вычитание

# На рисунке 2 изображены два метода addition и subtraction.

# addition - складывает текущую дробь с другой дробью. Приводит дроби к общему знаменателю, складывает числители и возвращает результат в виде новой дроби.

# subtraction - выполняет вычитание одной дроби из другой, использует сложение с дробью, у которой числитель предварительно меняется на отрицательный.

# 

# Рисунок 3. Умножение и деление

# На рисунке 3 изображены два метода multiplication и division.

# multiplication - умножает текущую дробь на другую дробь. Для этого получает новую дробь с целой частью, которая равна произведению числителей и знаменателей.

# division - делит текущую дробь на другую дробь. Деление выполняется путем умножения текущей дроби на обратную дробь.

# 

# Рисунок 4. Геттеры и сеттеры

# На рисунке 4 изображены геттеры и сеттеры.

# Геттеры используются для получения значений целой части, числителя и знаменателя (Чтение данных из private класса).

# Сеттеры используются для изменения значений полей. В каждом сеттере выполняется проверка корректности данных через isValid(), и если данные некорректные, вызывается исключение с сообщением.

# 

# Рисунок 5. Запись в текстовый файл и бинарный файл

# На рисунке 5 изображены методы записи дробей в текстовый файл и бинарный файл.

# 

# Рисунок 6. Преобразование в строку

# На рисунке 6 изображен метод, который преобразует дробь в строку, для удобного отображения.

# Также ниже показаны вспомогательные методы.

# 

# Рисунок 7. Преобразование в строку

# На рисунке 7 изображены вспомогательные методы:

# Метод getFraction() - приводит дроби к виду с целой частью 0 (правильные дроби) и возвращает дробь, которая получается при умножении дробей.

# Метод getNOD() - находит наибольший общий делитель (НОД) двух чисел, используя алгоритм Евклида.

# Метод getCorrectNumerator() - преобразует дробь с целой частью в дробь без целой части (неправильную дробь), чтобы было проще выполнять арифметические операции

# isValid() - проверяет корректность дроби. Проверка заключается в том, что числитель по модулю должен быть меньше знаменателя, знаменатель больше 0, если целая часть не равна 0, числитель должен быть неотрицательным

# Далее идет описание класса Main.

# 

# Рисунок 8. Класс Main

# На рисунке 8 показан класс main в котором выполняется ввод дробей, далее происходит обработка (сложение, вычитание, умножение, деление, запись в текстовый и бинарный файлы), и вывод в консоль результатов операций.

# Результат работы

# Пример использования программы.

# Введем две дроби 5 целых 3/8 и 2 целых 9/12, ниже показаны результат работы программы:

# рисунок 9 – математические операции

# рисунок 10 – запись в текстовом виде

# рисунок 11 – запись в двоичном коде.

# 

# Рисунок 9. Результат работы программы

# 

# Рисунок 10. Результат работы программы

# 

# Рисунок 11. Результат работы программы

# Приложение

# Class Main.java

# package main.java;

# import java.io.IOException;

# import java.util.Scanner;

# public class Main {

# public static void main(String[] args) throws IOException {

# var firstFraction = getFraction();

# var secondFraction = getFraction();

# System.out.println("Результат сложения : " + firstFraction.addition(secondFraction));

# System.out.println("Результат умножения : " + firstFraction.multiplication(secondFraction));

# System.out.println("Результат вычитания : " + firstFraction.subtraction(secondFraction));

# System.out.println("Результат деления : " + firstFraction.division(secondFraction));

# //firstFraction.setNumerator(-1);

# firstFraction.writeToFileChar();

# firstFraction.writeToFileBinary();

# }

# public static Fraction getFraction() {

# Scanner scanner = new Scanner(System.in);

# System.out.println("Введите целую часть");

# int wholePart = scanner.nextInt();

# System.out.println("Введите числитель дроби");

# int numerator = scanner.nextInt();

# System.out.println("Введите знаменатель дроби");

# int denominator = scanner.nextInt();

# return Fraction.of(wholePart, numerator, denominator);

# }

# }

# Class Fraction.java

# package main.java;

# import java.io.DataOutputStream;

# import java.io.IOException;

# import java.nio.file.Files;

# import java.nio.file.Path;

# public class Fraction {

# private int wholePart;

# private int numerator;

# private int denominator;

# private Fraction(int wholePart, int numerator, int denominator) {

# this.wholePart = wholePart;

# this.numerator = numerator;

# this.denominator = denominator;

# }

# public static Fraction of(int wholePart, int numerator, int denominator) {

# if (isValid(wholePart, numerator, denominator)) {

# return new Fraction(wholePart, numerator, denominator);

# } else {

# throw new IllegalArgumentException("Введены не корректные данные");

# }

# }

# public Fraction addition(Fraction fraction) {

# int currentFractionNumerator = getCorrectNumerator(this);

# int additionalFractionNumerator = getCorrectNumerator(fraction);

# int minDenominator = this.getNOD(this.denominator, fraction.denominator);

# minDenominator = (this.denominator \* fraction.denominator) / minDenominator;

# int commonNumerator = currentFractionNumerator \* minDenominator / this.denominator

# + additionalFractionNumerator \* minDenominator / fraction.denominator;

# int commonNODNumeratorDenominator = this.getNOD(commonNumerator, minDenominator);

# if (commonNODNumeratorDenominator != 1) {

# commonNumerator /= commonNODNumeratorDenominator;

# minDenominator /= commonNODNumeratorDenominator;

# }

# int currentWholePart = commonNumerator / minDenominator;

# commonNumerator -= currentWholePart \* minDenominator;

# if (commonNumerator < 0) {

# commonNumerator \*= -1;

# }

# return Fraction.of(currentWholePart, commonNumerator, minDenominator);

# }

# public Fraction subtraction(Fraction fraction) {

# return addition(new Fraction(

# 0,

# -getCorrectNumerator(fraction),

# fraction.denominator

# ));

# }

# public Fraction multiplication(Fraction fraction) {

# var tmpFraction = getFraction(fraction);

# int currentWholePart = tmpFraction.numerator / tmpFraction.denominator;

# int currentNumerator = tmpFraction.numerator - currentWholePart \* tmpFraction.denominator;

# int nod = getNOD(currentNumerator, tmpFraction.denominator);

# if (currentWholePart < 0 && currentNumerator < 0) {

# currentNumerator \*= -1;

# }

# return Fraction.of(currentWholePart, currentNumerator / nod, tmpFraction.denominator / nod);

# }

# public Fraction division(Fraction fraction) {

# var tmpNumerator = getCorrectNumerator(fraction);

# var tmpDenominator = fraction.denominator;

# if (tmpNumerator < 0) {

# tmpNumerator \*= -1;

# tmpDenominator \*= -1;

# }

# return multiplication(new Fraction(

# 0,

# tmpDenominator,

# tmpNumerator

# ));

# }

# public int getWholePart() {

# return wholePart;

# }

# public void setWholePart(int wholePart) {

# try {

# if (!isValid(wholePart, numerator, denominator)) {

# throw new IllegalArgumentException(

# "Предоставлен не корректные данные");

# }

# this.wholePart = wholePart;

# } catch (IllegalArgumentException ex) {

# System.out.println(ex.getLocalizedMessage());

# }

# }

# public int getNumerator() {

# return numerator;

# }

# public void setNumerator(int numerator) {

# try {

# if (!isValid(wholePart, numerator, denominator)) {

# throw new IllegalArgumentException(

# "Предоставлен не корректный числитель "

# + "который больше либо равен знаменателю");

# }

# this.numerator = numerator;

# } catch (IllegalArgumentException ex) {

# System.out.println(ex.getLocalizedMessage());

# }

# }

# public int getDenominator() {

# return denominator;

# }

# public void setDenominator(int denominator) {

# try {

# if (!isValid(wholePart, numerator, denominator)) {

# throw new IllegalArgumentException(

# "Предоставлен не корректный знаменатель "

# + "который меньше либо равен числителю");

# }

# this.denominator = denominator;

# } catch (IllegalArgumentException ex) {

# System.out.println(ex.getLocalizedMessage());

# }

# }

# public void writeToFileChar() throws IOException {

# var path = Path.of(System.getProperty("user.dir"), "/char.txt");

# Files.deleteIfExists(path);

# Files.createFile(path);

# try (var bf = Files.newBufferedWriter(path)) {

# bf.write(this.toString());

# }

# }

# public void writeToFileBinary() throws IOException {

# var path = Path.of(System.getProperty("user.dir"), "/byte.txt");

# Files.deleteIfExists(path);

# Files.createFile(path);

# try (var dos = new DataOutputStream(Files.newOutputStream(path))) {

# dos.write(wholePart);

# dos.write(numerator);

# dos.write(denominator);

# }

# }

# @Override

# public String toString() {

# String str = wholePart + " " + numerator + "/" + denominator;

# if (numerator == 0) {

# str = String.valueOf(wholePart);

# }

# if (wholePart == 0 && numerator != 0) {

# str = numerator + "/" + denominator;

# }

# return str;

# }

# private Fraction getFraction(Fraction fraction) {

# int currentFractionNumerator = getCorrectNumerator(this);

# int additionalFractionNumerator = getCorrectNumerator(fraction);

# var tmpFraction = new Fraction(

# 0,

# additionalFractionNumerator,

# fraction.denominator

# );

# var currentFraction = new Fraction(

# 0,

# currentFractionNumerator,

# denominator

# );

# tmpFraction = new Fraction(

# 0,

# tmpFraction.numerator \* currentFraction.numerator,

# tmpFraction.denominator \* currentFraction.denominator

# );

# return tmpFraction;

# }

# private int getNOD(int a, int b) {

# while (b != 0) {

# int temp = b;

# b = a % b;

# a = temp;

# }

# return Math.abs(a);

# }

# private int getCorrectNumerator(Fraction fraction) {

# int fractionNumerator = fraction.numerator;

# if (fraction.wholePart > 0) {

# fractionNumerator += fraction.denominator \* fraction.wholePart;

# }

# if (fraction.wholePart < 0) {

# fractionNumerator += fraction.denominator \* (-fraction.wholePart);

# fractionNumerator \*= -1;

# }

# return fractionNumerator;

# }

# public static boolean isValid(int wholePart, int numerator, int denominator) {

# boolean isCorrectNumerator = true;

# if (wholePart != 0) {

# isCorrectNumerator = numerator >= 0;

# }

# var isValidNumerator = (Math.abs(numerator) < Math.abs(denominator)) && isCorrectNumerator;

# var isNonZeroDenominator = denominator > 0;

# return isValidNumerator && isNonZeroDenominator;

# }

# }