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

##### «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

##### КАФЕДРА ВТ

##### ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

##### «Разработка интерфейса и фабрики для произвольного 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 15](#_bookmark2)

[Приложение 17](#_bookmark3)

# Постановка задачи

# Использовать интерфейс для произвольного типа данных, включающий в себя базовые операции:

# public interface Comparator {

# public int compare(Object o1, Object o2);

# }

# public interface UserType{

# public String typeName();

# // Имя типа

# public Object create();

# // Создает объект ИЛИ

# public Object clone();

# // Клонирует текущий

# public Object readValue(InputStreamReader in); // Создает и читает объект

# public Object parseValue(String ss);

# // Создает и парсит содержимое из строки

# public Comparator getTypeComparator;

# // Возвращает компаратор для сравнения

# }

# Присоединить интерфейс к разработанному в л.р. 1 типу данных, а также создать класс с аналогичным интерфейсом для базового типа данных (int,String,double)

# Для класса структуры данных в хранимом элементе заменить int на интерфейс UserType

# В main написать тестовую последовательность создания объектов и выполнения над ними операций с трассировкой результатов для объектов класса л.р.2

# Выполнение работы

# Для начала был создан интерфейс Comparator, который определяет метод для сравнения двух объектов (рис.1).

# 

# Рисунок 1. Интерфейс Comparator

# Далее был создан интерфейс UserType, который описывает шаблон (generic) для работы с типами данных. Он предоставляет методы для создания, клонирования, чтения, парсинга объектов, а также для получения компаратора (рис.2).

# 

# Рисунок 2. Интерфейс UserType

# Далее идет изменения класса Fraction (лабораторная работа №1(правильная дрочь)), где реализуется интерфейс UserType и используется Comparator.

# Рисунок 3. Интерфейс UserType

# Hа рисунке 3 показано что класс Fraction реализует интерфейс UserType с параметром типа Fraction. Класс Fraction обязуется предоставить реализацию всех методов, определенных в интерфейсе UserType.

# Ниже будет показан код реализации методов интерфейса, без основного, так как реализация правильной дроби показана в лабораторной работе №1.

# 

# Рисунок 4. Реализация методов

# На рисунке 4 показана реализация следующих методов:

# typeName(): Возвращает имя типа.

# create(): Создает новый объект Fraction с начальными значениями.

# clone(): Создает копию текущего объекта Fraction.

# 

# Рисунок 5. Реализация методов

# На рисунке 5 показана реализация следующих методов:

# readValue(InputStreamReader in): Считывает значение из потока ввода и преобразует его в объект Fraction.

# parseValue(String ss): Преобразует строку в объект Fraction, если строка соответствует ожидаемому формату.

# 

# Рисунок 6. Метод getTypeComparator

# На рисунке 6 изображен метод getTypeComparator(): Возвращает компаратор для сравнения объектов Fraction с учетом дробных частей.

# Далее был создан класс StringUserTypeImpl, который является реализацией интерфейса UserType. Этот класс используется для работы со строками.

# 

# Рисунок 7. Класс StringUserTypeImpl

# Класс StringUserTypeImpl — это реализация интерфейса UserType для работы с типом данных, который инкапсулирует строку. Этот класс предоставляет методы для:

# Создания нового объекта,

# Клонирования объекта,

# Получения строки из объекта,

# Чтения строки из входного потока,

# Сравнения строк.

# Далее для классов LinkedList и ListNode (лабораторная работа №2 (односвязный список)) была заменена структура хранимого элемента с типа int на интерфейс UserType.

# 

# Рисунок 8 Метод getTypeComparator

# В классе LinkedList метод get, который ранее возвращал значение типа Integer, теперь возвращает объект типа UserType

# 

# Рисунок 9. Изменения

# 

# Рисунок 10. Изменения

# 

# Рисунок 11 Изменения

# На рисунках, показанных выше, вместо integer теперь UserType.

# 

# Рисунок 12. Изменения

# Для сравнения вызывается метод compare из объекта Comparator, полученного через getTypeComparator элемента UserType.

# Далее идет описание класса Main.

# 

# Рисунок 13. Класс Main

# На рисунке 13 показаны два метода, первый метод main запускает две функции: getTraceResultForFraction() и getTraceResultForStringImpl(). Второй Метод getTraceResultForFraction() демонстрирует работу с классом LinkedList и объектами типа Fraction. Процесс выполняется в несколько этапов:

# Создание объектов Fraction и добавление их в связанный список

# Вывод всех элементов в списке

# Получение и вывод третьего элемента в списке

# Вставка нового элемента в список

# Удаление элемента из списка

# Добавление элемента в конец списка

# Сортировка списка

# 

# Рисунок 14. Класс Main

# На рисунке 14 показан метод getTraceResultForStringImpl() этот метод демонстрирует работу с классом StringUserTypeImpl, который является пользовательским типом данных для работы со строками.

# Создание объектов типа StringUserTypeImpl и вывод их типа

# Создание нового объекта с помощью метода create()

# Клонирование объекта

# Парсинг строки и создание нового объекта

# Сравнение объектов с помощью компаратора: Получается компаратор с помощью метода getTypeComparator() и используется для сравнения двух объектов типа StringUserTypeImpl. Если первый объект больше второго, выводится значение первой строки, если они равны — выводится "equal", если второй объект больше — выводится значение второй строки.

# Результат работы

# Пример использования метода getTraceResultForFraction().

# Введем элементы списка [1 2/3, -1 2/3, 9/14, 5 3/8, -8 5/6];

# Выведем третий элемент списка (второй по индексу)

# 

# Добавим 1 2/3 на третье место и выведем его

# 

# Удалим третий элемент (1 2/3) списка и выведем новый

# 

# Добавим в конец число 1 4/5 и выведем последний элемент списка

# 

# Отсортируем список

# 

# Пример использования метода getTraceResultForStringImpl()

# 

# StringUserTypeImpl - результат вызова метода typeName()

# [ru.atlasyk.base.usertype.StringUserTypeImpl@6e2c634b](mailto:ru.atlasyk.base.usertype.StringUserTypeImpl@6e2c634b) -это результат вызова метода toString() для объекта stringUserType (который создается через someStr1.create()) (выводится стандартное представление объекта Java)

# cloneStr value: abg someStr1 value: abg - Этот вывод появляется после клонирования объекта someStr1 с помощью метода clone()

# ex str - Это результат работы метода parseValue()

# qual - Этот вывод возникает в результате сравнения двух объектов

# Приложение

# Class Main

# package ru.atlasyk;

# import java.net.CacheResponse;

# import ru.atlasyk.base.usertype.StringUserTypeImpl;

# import ru.atlasyk.fraction.Fraction;

# import ru.atlasyk.interfaces.Comparator;

# import ru.atlasyk.list.LinkedList;

# public class Main {

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

# getTraceResultForFraction();

# getTraceResultForStringImpl();

# }

# public static void getTraceResultForFraction() {

# var linkedList = new LinkedList();

# var fraction1 = Fraction.of(1, 2, 3);

# var fraction2 = Fraction.of(-1, 2, 3);

# var fraction3 = Fraction.of(0, 9, 14);

# var fraction4 = Fraction.of(5, 3, 8);

# var fraction5 = Fraction.of(-8, 5, 6);

# linkedList.add(fraction1);

# linkedList.add(fraction2);

# linkedList.add(fraction3);

# linkedList.add(fraction4);

# linkedList.add(fraction5);

# linkedList.forEach(x -> System.out.print(x + ", "));

# System.out.println();

# System.out.println("Third element: " + linkedList.get(2));

# linkedList.insert(2, fraction1);

# System.out.println("After insert: ");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# System.out.println("Third element: " + linkedList.get(2));

# linkedList.delete(2);

# System.out.println("After delete: ");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# System.out.println("Third element: " + linkedList.get(2));

# System.out.println("After adding to end: ");

# linkedList.add(Fraction.of(1, 4, 5));

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# System.out.println("Last element: " + linkedList.get(linkedList.size() - 1));

# System.out.println("Before sort: ");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# linkedList.sort();

# System.out.println("After sort:");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# }

# public static void getTraceResultForStringImpl() {

# var someStr = new StringUserTypeImpl("some str");

# var someStr1 = new StringUserTypeImpl("abg");

# System.out.println(someStr.typeName());

# var stringUserType = someStr1.create();

# System.out.println(stringUserType.toString());

# var cloneStr = someStr1.clone();

# System.out.println("cloneStr value: " + cloneStr.getStringValue()

# + " someStr1 value: " + someStr1.getStringValue());

# var exampleStr = "ex str";

# var stringUserType1 = someStr1.parseValue(exampleStr);

# System.out.println(stringUserType1.getStringValue());

# var typeComparator = someStr1.getTypeComparator();

# var compareValue = typeComparator.compare(someStr1, someStr1);

# if (compareValue > 0) {

# System.out.println(someStr1.getStringValue());

# } else if (compareValue == 0) {

# System.out.println("equal");

# } else {

# System.out.println(someStr.getStringValue());

# }

# }

# }

# Class StringUserTypeImpl

# package ru.atlasyk.base.usertype;

# import java.io.BufferedReader;

# import java.io.IOException;

# import java.io.InputStreamReader;

# import ru.atlasyk.fraction.Fraction;

# import ru.atlasyk.interfaces.Comparator;

# import ru.atlasyk.interfaces.UserType;

# public class StringUserTypeImpl implements UserType<StringUserTypeImpl> {

# private String stringValue;

# public StringUserTypeImpl(String stringValue) {

# this.stringValue = stringValue;

# }

# private StringUserTypeImpl() {}

# ;

# @Override

# public String typeName() {

# return this.getClass().getSimpleName();

# }

# @Override

# public StringUserTypeImpl create() {

# return new StringUserTypeImpl();

# }

# @Override

# public StringUserTypeImpl clone() {

# return new StringUserTypeImpl(stringValue);

# }

# public String getStringValue() {

# return stringValue;

# }

# @Override

# public StringUserTypeImpl readValue(InputStreamReader in) {

# try (var br = new BufferedReader(in)) {

# var input = br.readLine();

# return parseValue(input);

# } catch (IOException e) {

# throw new RuntimeException("some trouble in input stream");

# }

# }

# @Override

# public StringUserTypeImpl parseValue(String ss) {

# if (ss != null) {

# return new StringUserTypeImpl(ss);

# } else {

# throw new IllegalArgumentException("input param is null");

# }

# }

# @Override

# public Comparator getTypeComparator() {

# return ((o1, o2) -> {

# if (!(o1 instanceof StringUserTypeImpl str1 && o2 instanceof StringUserTypeImpl str2)) {

# throw new RuntimeException("Некорректные типы для сравнения");

# }

# return str1.stringValue.compareTo(str2.stringValue);

# });

# }

# }

# Class LinkedList

# package ru.atlasyk.list;

# import java.util.Objects;

# import java.util.function.Consumer;

# import ru.atlasyk.interfaces.Comparator;

# import ru.atlasyk.interfaces.UserType;

# public class LinkedList {

# private ListNode head;

# private int size;

# // Получить элемент по индексу

# public UserType get(int index) {

# ListNode current = head;

# int count = 0;

# while (current != null) {

# if (count == index) {

# return current.getData();

# }

# count++;

# current = current.getNext();

# }

# throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + count);

# }

# // Вставить элемент по индексу

# public void insert(int index, UserType data) {

# if (index < 0) {

# throw new IndexOutOfBoundsException("Index cannot be negative: " + index);

# }

# ListNode newNode = new ListNode(data);

# if (index == 0) {

# newNode.setNext(head);

# head = newNode;

# return;

# }

# ListNode current = head;

# for (int i = 0; i < index - 1; i++) {

# if (current == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# current = current.getNext();

# }

# newNode.setNext(current.getNext());

# current.setNext(newNode);

# size++;

# }

# // Удалить элемент по индексу

# public void delete(int index) {

# if (index < 0 || head == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# if (index == 0) {

# head = head.getNext();

# return;

# }

# ListNode current = head;

# for (int i = 0; i < index - 1; i++) {

# if (current == null || current.getNext() == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# current = current.getNext();

# }

# if (current.getNext() == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# current.setNext(current.getNext().getNext());

# size--;

# }

# // Добавить элемент в конец списка

# public void add(UserType data) {

# ListNode newNode = new ListNode(data);

# if (head == null) {

# head = newNode;

# size++;

# return;

# }

# ListNode current = head;

# while (current.getNext() != null) {

# current = current.getNext();

# }

# current.setNext(newNode);

# size++;

# }

# // Применить функцию к каждому элементу списка

# public void forEach(Consumer<? super UserType> func) {

# Objects.requireNonNull(func);

# ListNode current = head;

# while (current != null) {

# func.accept(current.getData());

# current = current.getNext();

# }

# }

# // Сортировка списка (метод пузырька)

# public void sort() {

# head = mergeSort(head);

# }

# public int size() {

# return size;

# }

# // Вспомогательный метод для сортировки

# private ListNode mergeSort(ListNode node) {

# if (node == null || node.getNext() == null) {

# return node; // База рекурсии

# }

# // Разделяем список на две половины

# ListNode middle = getMiddle(node);

# ListNode nextOfMiddle = middle.getNext();

# middle.setNext(null); // Разделяем на две части

# // Рекурсивно сортируем обе половины

# ListNode left = mergeSort(node);

# ListNode right = mergeSort(nextOfMiddle);

# // Объединяем отсортированные половины

# return sortedMerge(left, right);

# }

# // Метод для получения середины списка

# private ListNode getMiddle(ListNode node) {

# if (node == null) {

# return null;

# }

# ListNode slow = node;

# ListNode fast = node.getNext();

# // Используем два указателя для нахождения середины

# while (fast != null) {

# fast = fast.getNext();

# if (fast != null) {

# slow = slow.getNext();

# fast = fast.getNext();

# }

# }

# return slow;

# }

# // Метод для слияния двух отсортированных списков

# private ListNode sortedMerge(ListNode left, ListNode right) {

# ListNode result;

# // База рекурсии

# if (left == null) {

# return right;

# } else if (right == null) {

# return left;

# }

# // Сравниваем данные и сливаем

# Comparator typeComparator = left.getData().getTypeComparator();

# if (typeComparator.compare(right.getData(), left.getData()) < 0) {

# result = left;

# result.setNext(sortedMerge(left.getNext(), right));

# } else {

# result = right;

# result.setNext(sortedMerge(left, right.getNext()));

# }

# return result;

# }

# }

# class ListNode {

# UserType data;

# ListNode next;

# ListNode(UserType data, ListNode next) {

# this.data = data;

# this.next = next;

# }

# ListNode(UserType data) {

# this.data = data;

# }

# UserType getData() {

# return data;

# }

# void setData(UserType data) {

# this.data = data;

# }

# ListNode getNext() {

# return next;

# }

# void setNext(ListNode next) {

# this.next = next;

# }

# }

# Class Fraction

# package ru.atlasyk.fraction;

# import java.io.BufferedReader;

# import java.io.DataOutputStream;

# import java.io.IOException;

# import java.io.InputStreamReader;

# import java.nio.file.Files;

# import java.nio.file.Path;

# import java.util.ArrayList;

# import java.util.List;

# import java.util.StringTokenizer;

# import ru.atlasyk.interfaces.Comparator;

# import ru.atlasyk.interfaces.UserType;

# public class Fraction implements UserType<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;

# }

# @Override

# public String typeName() {

# return this.getClass().getSimpleName();

# }

# @Override

# public Fraction create() {

# return new Fraction(0, 0, 1);

# }

# @Override

# public Fraction clone() {

# return Fraction.of(this.getWholePart(), this.getNumerator(), this.getDenominator());

# }

# @Override

# public Fraction readValue(InputStreamReader in) {

# try (var br = new BufferedReader(in)) {

# var input = br.readLine();

# return parseValue(input);

# } catch (IOException e) {

# throw new RuntimeException("some trouble in input stream");

# }

# }

# @Override

# public Fraction parseValue(String ss) {

# if (!ss.matches("-?\\d\\s-?\\d\*\\/[1-9]\\d\*")) {

# throw new IllegalArgumentException("некорректные данные");

# }

# var stringTokenizer = new StringTokenizer(ss, " /");

# List<Integer> integers = new ArrayList<>();

# while (stringTokenizer.hasMoreElements()) {

# integers.add(Integer.valueOf(stringTokenizer.nextToken()));

# }

# return Fraction.of(integers.get(0), integers.get(1), integers.get(2));

# }

# @Override

# public Comparator getTypeComparator() {

# return (o1, o2) -> {

# if (!(o1 instanceof Fraction thisFraction && o2 instanceof Fraction otherFraction)) {

# throw new RuntimeException("Некорректные типы для сравнения");

# }

# if (thisFraction.wholePart != otherFraction.wholePart) {

# return Integer.compare(otherFraction.wholePart, thisFraction.wholePart);

# }

# int thisNumerator = thisFraction.wholePart < 0 ? -Math.abs(thisFraction.numerator)

# : thisFraction.numerator;

# int otherNumerator = otherFraction.wholePart < 0 ? -Math.abs(otherFraction.numerator)

# : otherFraction.numerator;

# int lcm = lcm(thisFraction.denominator, otherFraction.denominator); // НОК для знаменателей

# int thisAdjustedNumerator = thisNumerator \* (lcm / thisFraction.denominator);

# int otherAdjustedNumerator = otherNumerator \* (lcm / otherFraction.denominator);

# return Integer.compare(otherAdjustedNumerator, thisAdjustedNumerator);

# };

# }

# //NOK

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

# return Math.abs(a \* b) / getNOD(a, b);

# }

# }

# Интерфейс UserType

# package ru.atlasyk.interfaces;

# import java.io.InputStreamReader;

# public interface UserType<T> {

# String typeName();

# // Имя типа

# T create();

# // Создает объект ИЛИ

# T clone();

# // Клонирует текущий

# T readValue(InputStreamReader in); // Создает и читает объект

# T parseValue(String ss);

# // Создает и парсит содержимое из строки

# Comparator getTypeComparator();

# // Возвращает компаратор для сравнения

# }

# Интерфейс Comparator

# package ru.atlasyk.interfaces;

# public interface Comparator {

# int compare(Object o1, Object o2);

# }