|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 3 |

**Дисциплина: Языки программирования для работы с большими данными.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-21М |  |  | В.В. Ерискин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**Задание**

Задача 1. Определить класс Дробь в виде пары (m,n). Класс должен содержать несколько конструкторов. Реализовать методы для сложения, вычитания, умножения и деления дробей. Объявить массив из k дробей, ввести/вывести значения для массива дробей. Создать массив объектов и передать его в метод, который изменяет каждый элемент массива с четным индексом путем добавления следующего за ним элемента массива.

Задача 2.

Создать классы, спецификации которых приведены ниже. Определить конструкторы и методы setТип(), getТип(), toString(). Определить дополнительно методы в классе, создающем массив объектов. Задать критерий выбора данных и вывести эти данные на консоль.

Phone: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Дебет, Кредит, Время городских и междугородных разговоров. Создать массив объектов. Вывести: a) сведения об абонентах, у которых время внутригородских разговоров превышает заданное; b) сведения об абонентах, которые пользовались междугородной связью; c) сведения об абонентах в алфавитном порядке.

Задача 3.

Создать приложение, удовлетворяющее требованиям, приведенным в задании. Аргументировать принадлежность классу каждого создаваемого метода и корректно переопределить для каждого класса методы equals(), hashCode(), toString(). Задание 7. Создать объект класса Дерево, используя классы Лист. Методы: зацвести, опасть листьям, покрыться инеем, пожелтеть листьям.

**Решение**:

Программа 1.

|  |
| --- |
| import java.util.ArrayList; |
|  |  |
|  | class MyClass { |
|  | public static ArrayList<Fraction> array = new ArrayList<Fraction>(); |
|  |  |
|  |  |
|  | } |
|  |  |
|  | public class Fraction { |
|  |  |
|  | int numerator; |
|  | int denominator; |
|  |  |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Constructor |
|  | \* |
|  | \* @param numr |
|  | \* @param denr |
|  | \*/ |
|  | public Fraction(int numr, int denr) { |
|  | numerator = numr; |
|  | denominator = denr; |
|  | reduce(); |
|  | } |
|  |  |
|  | public int getNumerator() { |
|  | return numerator; |
|  | } |
|  |  |
|  | public void setNumerator(int numerator) { |
|  | this.numerator = numerator; |
|  | } |
|  |  |
|  | public int getDenominator() { |
|  | return denominator; |
|  | } |
|  |  |
|  | public void setDenominator(int denominator) { |
|  | this.denominator = denominator; |
|  | } |
|  | /\*\* |
|  | \* Calculates gcd of two numbers |
|  | \* |
|  | \* @param numerator |
|  | \* @param denominator |
|  | \* @return |
|  | \*/ |
|  | public int calculateGCD(int numerator, int denominator) { |
|  | if (numerator % denominator == 0) { |
|  | return denominator; |
|  | } |
|  | return calculateGCD(denominator, numerator % denominator); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Reduce the fraction to lowest form |
|  | \*/ |
|  | void reduce() { |
|  | int gcd = calculateGCD(numerator, denominator); |
|  | numerator /= gcd; |
|  | denominator /= gcd; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Adds two fractions |
|  | \* |
|  | \* @param fractionTwo |
|  | \* @return |
|  | \*/ |
|  | public Fraction add(Fraction fractionTwo) { |
|  | int numer = (numerator \* fractionTwo.getDenominator()) + |
|  | (fractionTwo.getNumerator() \* denominator); |
|  | int denr = denominator \* fractionTwo.getDenominator(); |
|  | return new Fraction(numer, denr); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Subtracts two fractions |
|  | \* |
|  | \* @param fractionTwo |
|  | \* @return |
|  | \*/ |
|  | public Fraction subtract(Fraction fractionTwo) { |
|  | int newNumerator = (numerator \* fractionTwo.denominator) - |
|  | (fractionTwo.numerator \* denominator); |
|  | int newDenominator = denominator \* fractionTwo.denominator; |
|  | Fraction result = new Fraction(newNumerator, newDenominator); |
|  | return result; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Multiples two functions |
|  | \* |
|  | \* @param fractionTwo |
|  | \* @return |
|  | \*/ |
|  | public Fraction multiply(Fraction fractionTwo) { |
|  | int newNumerator = numerator \* fractionTwo.numerator; |
|  | int newDenominator = denominator \* fractionTwo.denominator; |
|  | Fraction result = new Fraction(newNumerator, newDenominator); |
|  | return result; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Divides two fractions |
|  | \* |
|  | \* @param fractionTwo |
|  | \* @return |
|  | \*/ |
|  | public Fraction divide(Fraction fractionTwo) { |
|  | int newNumerator = numerator \* fractionTwo.getDenominator(); |
|  | int newDenominator = denominator \* fractionTwo.numerator; |
|  | Fraction result = new Fraction(newNumerator, newDenominator); |
|  | return result; |
|  | } |
|  | /\*\* |
|  | \* Returns string representation of the fraction |
|  | \*/ |
|  | @Override |
|  | public String toString() { |
|  | return this.numerator + "/" + this.denominator; |
|  | } |
|  |  |
|  |  |
|  | public static void main(String[] args) { |
|  | Fraction f1 = new Fraction(25, 35); |
|  | System.out.println("Первая дробь " + f1.toString()); |
|  | Fraction f2 = new Fraction(2, 5); |
|  | System.out.println("Вторая дробь " + f2.toString()); |
|  | Fraction f3 = f1.add(f2); |
|  | System.out.println("Результат сложения " |
|  | + f1 + " и " + f2 + " is : " + f3); |
|  | f3 = f1.subtract(f2); |
|  | System.out.println("Результат вычитания " |
|  | + f1 + " и " + f2 + " is : " + f3); |
|  | f3 = f1.divide(f2); |
|  | System.out.println("Результат деления " |
|  | + f1 + " и " + f2 + " is : " + f3); |
|  | f3 = f1.multiply(f2); |
|  | System.out.println("Результат умножения " |
|  | + f1 + " и " + f2 + " is : " + f3); |
|  | int count = 1; |
|  |  |
|  | Fraction k = new Fraction(1, 3); |
|  | do { |
|  | Fraction f4; |
|  | f4 = new Fraction(1, (count+1)); |
|  | Fraction c = k.add(f4); |
|  | MyClass.array.add(c); |
|  | count++; |
|  | } while (count < 7); |
|  | System.out.println("Начальный массив из k - дробей " + MyClass.array); |
|  | //System.out.println("Модифиц. массив из k - дробей " + MyClass.array) |
|  | } |
|  | } |

Программа 2.

|  |
| --- |
| import java.util.List; |
|  | import java.util.Collections; |
|  | import java.util.Comparator; |
|  | import java.util.ArrayList; |
|  |  |
|  | public class Phones { |
|  | public static void main(String[] args) { |
|  | PhonesList phoneList = new PhonesList(); |
|  | addPhoneList(phoneList); |
|  |  |
|  |  |
|  | phoneList.ShowCityTimeUnderValue(60); |
|  | phoneList.ShowPhonesUseIntercity(); |
|  | PhonesList sortPhoneList; |
|  | sortPhoneList=phoneList.sortPhoneList(); |
|  | System.out.println("Сведения об абонентах в алфавитном порядке:"); |
|  | sortPhoneList.printPhonelist(); |
|  |  |
|  | } |
|  |  |
|  | public static void addPhoneList(PhonesList phoneList) { |
|  | phoneList.addPhoneList(new Phone("Баранов", "Олег", "Иванович", 150467887863690L, 100, 50, 1000, 100)); |
|  | phoneList.addPhoneList(new Phone("Сидоров", "Иван", "Петрович", 119338934889750L, 200, 100, 100, 20000)); |
|  | phoneList.addPhoneList(new Phone("Бобчинская", "Ольга", "Владимировна", 939908453736346L, 300, 150, 5000, 23444)); |
|  | phoneList.addPhoneList(new Phone("Коновалов", "Олег", "Егорович", 783400770797464L, 400, 200, 7777, 50)); |
|  | phoneList.addPhoneList(new Phone("Белый", "Антон", "Платонович", 983643622083780L, 500, 250, 0, 320)); |
|  | phoneList.addPhoneList(new Phone("Скоробогатько", "Ирина", "Александровна", 845982235567799L, 60, 125, 12345, 7000)); |
|  | phoneList.addPhoneList(new Phone("Морозов", "Антон", "Максимович", 603095181535640L, 600, 125, 900, 0)); |
|  |  |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  | class Phone { |
|  | private static int idCount = 0; |
|  |  |
|  | private int id; |
|  | private String name1; |
|  | private String name2; |
|  | private String name3; |
|  | private long card; |
|  | private double debit; |
|  | private double credit; |
|  | private long oncityTimer; |
|  | private long intercityTimer; |
|  |  |
|  | public Phone(String name1, String name2, String name3, long card, double debit, double credit, long oncityTimer, long intercityTimer) { |
|  | setId(); |
|  | setName1(name1); |
|  | setName2(name2); |
|  | setName3(name3); |
|  | setCard(card); |
|  | setDebit(debit); |
|  | setCredit(credit); |
|  | setOncityTimer(oncityTimer); |
|  | setIntercityTimer(intercityTimer); |
|  | } |
|  |  |
|  | public Phone(String name1, String name2, String name3, long cardNo) { |
|  | setId(); |
|  | setName1(name1); |
|  | setName2(name2); |
|  | setName3(name3); |
|  | setCard(card); |
|  | } |
|  |  |
|  | public Phone(String name1, String name2, String name3, long cardNo, double debit, double credit) { |
|  | setId(); |
|  | setName1(name1); |
|  | setName2(name2); |
|  | setName3(name3); |
|  | setCard(card); |
|  | setDebit(debit); |
|  | setCredit(credit); |
|  | } |
|  |  |
|  | public int getId() { |
|  | return id; |
|  | } |
|  |  |
|  | private void setId() { |
|  | this.id = ++idCount; |
|  | } |
|  |  |
|  | public String getName1() { |
|  | return name1; |
|  | } |
|  |  |
|  | public void setName1(String name1) { |
|  | checkString(name1); |
|  | this.name1 = name1; |
|  | } |
|  |  |
|  | private void checkString(String string) { |
|  | if (string == null || string.equals("")) { |
|  | throw new IllegalArgumentException("Value is empty!!!"); |
|  | } |
|  | } |
|  |  |
|  | public String getName2() { |
|  | return name2; |
|  | } |
|  |  |
|  | public void setName2(String name2) { |
|  | checkString(name2); |
|  | this.name2 = name2; |
|  | } |
|  |  |
|  | public String getName3() { |
|  | return name3; |
|  | } |
|  |  |
|  | public void setName3(String name3) { |
|  | checkString(name3); |
|  | this.name3 = name3; |
|  | } |
|  |  |
|  | public long getCard() { |
|  | return card; |
|  | } |
|  |  |
|  | public void setCard(long card) { |
|  | checkLong(card); |
|  | this.card = card; |
|  | } |
|  |  |
|  | private void checkLong(long number) { |
|  | if (number <= 0) { |
|  | throw new IllegalArgumentException("Wrong Number!!!"); |
|  | } |
|  | } |
|  |  |
|  | private void checkTime(long seconds) { |
|  | if (seconds < 0) { |
|  | throw new IllegalArgumentException("Wrong Number!!!"); |
|  | } |
|  | } |
|  |  |
|  | public double getDebit() { |
|  | return debit; |
|  | } |
|  |  |
|  | public void setDebit(double debit) { |
|  | checkDouble(debit); |
|  | this.debit = debit; |
|  | } |
|  |  |
|  | private void checkDouble(double amount) { |
|  | if (amount < 0) { |
|  | throw new IllegalArgumentException("Wrong amount!!!"); |
|  | } |
|  | } |
|  |  |
|  | public double getCredit() { |
|  | return credit; |
|  | } |
|  |  |
|  | public void setCredit(double credit) { |
|  | checkDouble(credit); |
|  | this.credit = credit; |
|  | } |
|  |  |
|  | public long getOncityTimer() { |
|  | return oncityTimer; |
|  | } |
|  |  |
|  | public void setOncityTimer(long oncityTimer) { |
|  | checkTime(oncityTimer); |
|  | this.oncityTimer = oncityTimer; |
|  | } |
|  |  |
|  | public long getIntercityTimer() { |
|  | return intercityTimer; |
|  | } |
|  |  |
|  | public void setIntercityTimer(long intercityTimer) { |
|  | checkTime(intercityTimer); |
|  | this.intercityTimer = intercityTimer; |
|  | } |
|  |  |
|  | @Override |
|  | public String toString() { |
|  | final StringBuilder sb = new StringBuilder("Phone{"); |
|  | sb.append("id=").append(id); |
|  | sb.append(", name1='").append(name1).append('\''); |
|  | sb.append(", name2='").append(name2).append('\''); |
|  | sb.append(", name3='").append(name3).append('\''); |
|  | sb.append(", card=").append(card); |
|  | sb.append(", debit=").append(debit); |
|  | sb.append(", credit=").append(credit); |
|  | sb.append(", oncityTimer=").append(oncityTimer); |
|  | sb.append(", intercityTimer=").append(intercityTimer); |
|  | sb.append('}'); |
|  | return sb.toString(); |
|  | } |
|  | } |
|  |  |
|  | class PhonesList { |
|  | private List<Phone> phoneList = new ArrayList<>(); |
|  |  |
|  | public void addPhoneList(Phone phone) { |
|  | this.phoneList.add(phone); |
|  | } |
|  |  |
|  | public void printPhonelist() { |
|  | for (Phone phone : phoneList) { |
|  | System.out.println(phone); |
|  | } |
|  | } |
|  |  |
|  | public void ShowCityTimeUnderValue(long second) { |
|  | System.out.println("Абоненты, у которых время внутригородских разговоров превышает > " + second + " sec.:"); |
|  | for (Phone phone : phoneList) { |
|  | if (phone.getOncityTimer() > second) { |
|  | System.out.println(phone); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | public void ShowPhonesUseIntercity() { |
|  | System.out.println("Абоненты, которые пользовались междугородной связью:"); |
|  | for (Phone phone : phoneList) { |
|  | if (phone.getIntercityTimer() > 0) { |
|  | System.out.println(phone); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | public PhonesList sortPhoneList() { |
|  | List<Phone> custlist = new ArrayList<>(phoneList); |
|  | Collections.sort(custlist, new Comparator<Phone>() { |
|  | @Override |
|  | public int compare(Phone cust1, Phone cust2) { |
|  | return cust1.getName1().compareToIgnoreCase(cust2.getName1()); |
|  | } |
|  | }); |
|  | PhonesList sortPhoneList = new PhonesList(); |
|  | for (Phone phone : custlist) { |
|  | sortPhoneList.addPhoneList(phone); |
|  | } |
|  | return sortPhoneList; |
|  | } |
|  |  |
|  | } |

Программа 3.

public class Main {

|  |  |
| --- | --- |
|  | //Equal objects must have equal hash codes |
|  | public static void testLeaf(Leaf leaf1, Leaf leaf2){ |
|  | System.out.println(leaf1 + ", " + leaf2); |
|  | System.out.print("leaf1.equals(leaf2): "); |
|  | System.out.println(leaf1.equals(leaf2)); |
|  | System.out.print("leaf1.hashCode() == leaf2.hashCode(): "); |
|  | System.out.println(leaf1.hashCode() == leaf2.hashCode()); |
|  | System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"); |
|  | } |
|  | //Equal objects must have equal hash codes |
|  | public static void testTree(Tree tree1, Tree tree2){ |
|  | System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"); |
|  | System.out.print("tree1.equals(tree2): "); |
|  | System.out.println(tree1.equals(tree2)); |
|  | System.out.print("tree1.hashCode() == tree2.hashCode(): "); |
|  | System.out.println(tree1.hashCode() == tree2.hashCode()); |
|  | System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"); |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | System.out.println("------------------------------------------"); |
|  | System.out.println("Testing Leaf"); |
|  | System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"); |
|  | Leaf leaf1 = new Leaf(); |
|  | Leaf leaf2 = new Leaf(); |
|  | testLeaf(leaf1,leaf2); |
|  | leaf1.turnYellow(); |
|  | testLeaf(leaf1,leaf2); |
|  |  |
|  | System.out.println("------------------------------------------"); |
|  | System.out.println("Testing Tree"); |
|  | System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"); |
|  | Tree tree1 = new Tree(); |
|  | Tree tree2 = new Tree(); |
|  | System.out.println(tree1 + ", " + tree2); |
|  | testTree(tree1, tree2); |
|  |  |
|  | for (int i = (int) (Math.random()\*5+1); i > 0; i--) { |
|  | tree1.changeState(); |
|  | tree2.changeState(); |
|  | } |
|  | testTree(tree1, tree2); |
|  |  |
|  | } |
|  | } |

|  |
| --- |
| import lombok.Setter; |
|  | import lombok.Getter; |
|  | import java.util.ArrayList; |
|  |  |
|  |  |
|  | public class Tree { |
|  |  |
|  | private State state; |
|  | @Setter @Getter |
|  | private ArrayList<Leaf> leaves = new ArrayList<>(); |
|  |  |
|  | public Tree(){ |
|  | state = State.NEW; |
|  | } |
|  | public void changeState() { state = state.doNext(this); |
|  | } |
|  | @Override |
|  | public boolean equals(Object o) { |
|  | if (o == this) return true; |
|  | if (!(o instanceof Tree)) |
|  | return false; |
|  | Tree anotherTree = (Tree) o; |
|  | return anotherTree.leaves.size() == leaves.size() && anotherTree.state == state; |
|  | } |
|  | @Override |
|  | public String toString(){ |
|  | StringBuilder resultString = new StringBuilder("Tree has " + leaves.size() + " leaves: {"); |
|  | for (int i = 0; i < leaves.size(); i++){ |
|  | resultString.append("[" + leaves.get(i) + "]" + ","); |
|  | } |
|  | resultString.append("}"); |
|  | return resultString.toString(); |
|  | } |
|  | @Override |
|  | public int hashCode(){ |
|  | int result = 17; |
|  | result = 31 \* result + ((state == null) ? 0 : state.hashCode()); |
|  | result = 31 \* result + ((leaves == null) ? 0 : leaves.hashCode()); |
|  | return result; |
|  | } |
|  | } |

|  |
| --- |
| import java.util.ArrayList; |
|  |  |
|  | public enum State { |
|  |  |
|  | NEW { |
|  | @Override |
|  | State doNext(Tree tree) { |
|  | ArrayList<Leaf> leafList = new ArrayList(); |
|  | int leavesAmount= (int) (Math.random()\*9+1); |
|  | while (0 <= leavesAmount--){ |
|  | leafList.add(new Leaf()); |
|  | } |
|  | //tree.setLeaves(leafList); |
|  | System.out.println(this); |
|  | System.out.println(tree); |
|  | return BLOOM; |
|  | } |
|  | }, |
|  | BLOOM { |
|  | @Override |
|  | State doNext(Tree tree) { |
|  | System.out.println(this); |
|  | System.out.println("Tree blooms"); |
|  | return YELLOWING; |
|  | } |
|  | }, |
|  | YELLOWING { |
|  | @Override |
|  | State doNext(Tree tree) { |
|  | System.out.println(this); |
|  | System.out.println(tree); |
|  | return FALLING; |
|  | } |
|  | }, |
|  | FALLING { |
|  | @Override |
|  | State doNext(Tree tree) { |
|  | System.out.println(this); |
|  | System.out.println(tree); |
|  | return FROSTED; |
|  | } |
|  | }, |
|  | FROSTED { |
|  | @Override |
|  | State doNext(Tree tree) { |
|  | System.out.println(this); |
|  | System.out.println("Tree is frosted"); |
|  | return NEW; |
|  | } |
|  | }; |
|  |  |
|  | abstract State doNext(Tree tree); |
|  | } |

|  |
| --- |
| class Leaf { |
|  | private enum LeafColor {GREEN, YELLOW} |
|  | private LeafColor color; |
|  |  |
|  | public Leaf(){ |
|  | color = LeafColor.GREEN; |
|  | } |
|  | void turnYellow(){ |
|  | color = LeafColor.YELLOW; |
|  | } |
|  | @Override |
|  | public boolean equals(Object o){ |
|  | if (o == this) return true; |
|  | if (!(o instanceof Leaf)) |
|  | return false; |
|  | return ((Leaf) o).color == this.color; |
|  | } |
|  | @Override |
|  | public String toString(){ |
|  | return (this.color + " leaf").toLowerCase(); |
|  | } |
|  | @Override |
|  | public int hashCode(){ |
|  | int result = 17; |
|  | result = 31 \* result + ((color == null) ? 0 : color.hashCode()); |
|  | return result; |
|  | } |
|  | } |

**Вывод:**

Задание выполнено, программы компилируются и успешно выполняются.