Отчет по лабораторной работе №5

“Переопределение методов, унаследованные от класса Object"

Выполнила: Качкуркина Арина Валерьевна

Группа: 6204-010302D

Год: 2025

Задание 1

Метод toString(): я реализовала метод, который возвращает текстовое представление точки в формате "(x; y)". Это удобно для отладки и вывода информации.

Метод equals(): реализация метода equals()мс сравнением чисел с плавающей точкой. Использовала константу EPSILON для учёта погрешности вычислений.

Метод hashCode(): для вычисления хэш-кода я использовала метод Double.doubleToLongBits() для преобразования double в битовое представление, затем применила операцию XOR.

Метод clone(): реализовала простое клонирование, так как класс FunctionPoint не содержит ссылок на другие изменяемые объекты.

```
//метод возвращает текстовое предст. точки в формате (x; y)
@Override
public String toString() {
    return "(" + x + "; " + y + ")";
}
    //метод сравнивает тек. точку с другой точкой на р-во коорд.
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof FunctionPoint)) return false;
    FunctionPoint that = (FunctionPoint) o;
    return Math.abs(this.x - that.x) < EPSILON &&
           Math.abs(this.y - that.y) < EPSILON;
}
    //метод преобразует double в long биты и комбинирует их через XOR
@Override
public int hashCode() {
    long xBits = Double.doubleToLongBits(x);
    long yBits = Double.doubleToLongBits(y);
    return (int)(xBits ^ (xBits >>> 32)) ^ (int)(yBits ^ (yBits >>> 32));
}
    //метод создает и возвращает копию текущей точки
@Override
public Object clone() {
    return new FunctionPoint(this.x, this.y);
}
```

Задание 2

Метод toString(): формирует строку в формате "{(x1; y1), (x2; y2), ...}", перебирая все точки массива.

Метод equals():добавила оптимизацию: если сравниваемый объект тоже является ArrayTabulatedFunction, то происходит прямое сравнение массивов точек. В противном случае используется общий механизм через интерфейс TabulatedFunction.

Метод hashCode(): Хэш-код вычисляется как комбинация хэш-кодов всех точек и количества точек. Это гарантирует, что функции с разным количеством точек будут иметь разные хэш-коды.

Метод clone(): реализовала глубокое клонирование, создавая новый массив и клонируя каждую точку отдельно.

```
//метод сравнивает текущую функцию с другой функцией на равенство точек (оптимизирован через прямой доступ к узлам)
@Override
public boolean equals(Object o) {
    //если тот же объект в памяти
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;//проверка типа

    //оптимизация(если сравниваем с LinkedListTabulatedFunction)
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction other = (LinkedListTabulatedFunction) o;

        //проверка:разное кол-во точек -> не равны
        if (this.pointsCount != other.pointsCount) return false;

        //оптим. сравнение: прямой обход узлов обоих списков
        FunctionNode currentThis = this.head.next;    //начинаем с первого узла этого списка
        FunctionNode currentOther = other.head.next;  //начинаем с первого узла другого списка

        for (int i = 0; i < pointsCount; i++) {
            //сравниваем точки из узлов
            if (!currentThis.point.equals(currentOther.point)) return false;

            //переходим к следующим узлам в обоих списках
            currentThis = currentThis.next;
            currentOther = currentOther.next;
        }
        return true;
    }
    else {
        TabulatedFunction other = (TabulatedFunction) o;

        //проверка кол-ва точек
        if (this.getPointsCount() != other.getPointsCount()) return false;

        //сравниваем через getPoint()
        for (int i = 0; i < pointsCount; i++) {
            if (!this.getPoint(i).equals(other.getPoint(i))) return false;//getPoint() создает копию точки
        }
        return true;
    }
}

//метод вычисляет хэш-код функции на основе хэш-кодов всех точек и их количества
//включает количество точек чтобы различать функции с одинаковыми точками но разным количеством
@Override
public int hashCode() {
    int result = pointsCount;  //начинаем с количества точек
    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        result ^= current.point.hashCode();  //комбинируем хэш точки через XOR
        current = current.next;
    }
    return result;
}

//метод создает глубокую копию тек. ф-ии через "пересборку" списка
@Override
public Object clone() {
    if (pointsCount == 0) return new LinkedListTabulatedFunction();  //если список пуст - возвращаем пустой

    //создаем массив клон точек из текущего списка
    FunctionPoint[] pointsArray = new FunctionPoint[pointsCount];
    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        pointsArray[i] = (FunctionPoint) current.point.clone();  //клонируем каждую точку
        current = current.next;
    }

    //создаем новую ф-ию используя конструктор с массивом точек
    return new LinkedListTabulatedFunction(pointsArray);
}
```

Задание 3

Метод toString(): аналогично массиву, но с обходом связного списка.

Метод equals(): добавила оптимизацию для случая, когда оба объекта являются LinkedListTabulatedFunction - происходит прямое сравнение узлов списка.

Метод hashCode(): вычисляется обходом связного списка и комбинацией хэш-кодов точек.

Метод clone(): вместо я использовала подход "пересборки" - создала массив точек из исходного списка и передала его в конструктор.

Задание 4

Я добавила метод clone() в интерфейс TabulatedFunction, чтобы все реализации были клонируемыми,

Задание 5

Для тестирования я создала класс Main, в котором:

-проверила toString() (убедилась, что вывод соответствует ожидаемому формату);

-протестировала equals()(проверила сравнение одинаковых и различных объектов);

-проверила глубокое клонирование (изменила оригинальный объект и убедилась, что клон не изменился).

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Тестирование методов toString(), equals(), hashCode(), clone()\n ");
        System.out.println();

        //создаем тестовые точки
        FunctionPoint point1 = new FunctionPoint(1.0, 2.0);
        FunctionPoint point2 = new FunctionPoint(2.0, 4.0);
        FunctionPoint point3 = new FunctionPoint(3.0, 6.0);

        //создаем массивы точек для функций
        FunctionPoint[] points1 = {point1, point2, point3};
        FunctionPoint[] points2 = {point1, point2, point3}; //одинак. точки
        FunctionPoint[] points3 = {new FunctionPoint(1.0, 2.0), new FunctionPoint(2.0, 5.0), new FunctionPoint(3.0, 6.0)};

        //создаем объект ф-ии
        ArrayTabulatedFunction arrayFunc1 = new ArrayTabulatedFunction(points1);
        ArrayTabulatedFunction arrayFunc2 = new ArrayTabulatedFunction(points2);
        ArrayTabulatedFunction arrayFunc3 = new ArrayTabulatedFunction(points3);

        LinkedListTabulatedFunction listFunc1 = new LinkedListTabulatedFunction(points1);
        LinkedListTabulatedFunction listFunc2 = new LinkedListTabulatedFunction(points2);

        //тестирование метода toString()
        System.out.println("1. Метод toString():");
        System.out.println("ArrayTabulatedFunction: " + arrayFunc1.toString());
        System.out.println("LinkedListTabulatedFunction: " + listFunc1.toString());
        System.out.println();

        //тестирование метода equals()
        System.out.println("2. Метод equals():");
        System.out.println("arrayFunc1.equals(arrayFunc2): " + arrayFunc1.equals(arrayFunc2));
        System.out.println("arrayFunc1.equals(arrayFunc3): " + arrayFunc1.equals(arrayFunc3));
        System.out.println("arrayFunc1.equals(listFunc1): " + arrayFunc1.equals(listFunc1));
        System.out.println("listFunc1.equals(listFunc2): " + listFunc1.equals(listFunc2));
        System.out.println();

        //тестирование метода hashCode()
        System.out.println("3. Метод hashCode():");
        System.out.println("arrayFunc1 hashCode: " + arrayFunc1.hashCode());
        System.out.println("arrayFunc2 hashCode: " + arrayFunc2.hashCode());
        System.out.println("arrayFunc3 hashCode: " + arrayFunc3.hashCode());
        System.out.println("listFunc1 hashCode: " + listFunc1.hashCode());
        System.out.println("listFunc2 hashCode: " + listFunc2.hashCode());
        System.out.println();

        //проверка изм. хэш при изм. объекта
        System.out.println("Проверка изменения хэш-кода:");
        System.out.println("point1 hashCode до изменения: " + point1.hashCode());
        point1.setY(2.1);
        System.out.println("point1 hashCode после изменения: " + point1.hashCode());
        System.out.println("Хэш-код изменился: " + (point1.hashCode() != new FunctionPoint(1.0, 2.0).hashCode()));
        System.out.println();

        //тестирование метода clone()
        System.out.println("4. Метод clone():");

        //клонируем ф-ию
        ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc1.clone();
        LinkedListTabulatedFunction listClone = (LinkedListTabulatedFunction) listFunc1.clone();

        System.out.println("arrayClone.equals(arrayFunc1): " + arrayClone.equals(arrayFunc1));
        System.out.println("listClone.equals(listFunc1): " + listClone.equals(listFunc1));
        System.out.println();

        //проверка глубокого копирования
        System.out.println("Проверка глубокого клонирования:");
        double originalValue = arrayFunc1.getPointY(0); //сохраняем исх.знач.
        System.out.println("Исходное значение arrayFunc1: " + originalValue);

        arrayFunc1.setPointY(0, 100.0); //изменяем ориг. ф-ию
        System.out.println("arrayFunc1 после изменения: " + arrayFunc1.getPointY(0));
        System.out.println("arrayClone после изменения оригинала: " + arrayClone.getPointY(0));
        System.out.println("Клон не изменился: " + (arrayClone.getPointY(0) == originalValue));

        System.out.println();
```

Все методы работают корректно. Реализованные оптимизации позволяют эффективно сравнивать объекты одного типа.

Результат работы программы:

1. Метод toString():

ArrayTabulatedFunction: {(1.0; 2.0), (2.0; 4.0)}

LinkedListTabulatedFunction: {(1.0; 2.0), (2.0; 4.0)}

2. Метод equals():

Сравнение Array и List (одинаковые точки): true

Сравнение Array и Array2: true

Сравнение List и List2: true

Сравнение Array и Different (разные точки): false

3. Метод hashCode():

Хэш Array: 2145386498

Хэш List: 2145386498

Хэш Array2: 2145386498

Хэш List2: 2145386498

Хэш Different: 2

Array и Array2 имеют одинаковый хэш: true

4. Проверка согласованности hashCode() и equals():

Исходный хэш point1: 2146435072

Хэш point1 после изменения Y: 910842937

5. Метод clone():

arrayClone равен arrayFunc: true

listClone равен listFunc: true

6. Проверка глубокого клонирования:

Исходное значение arrayFunc: 2.0

arrayFunc после изменения: 999.0

arrayClone после изменения оригинала: 2.0

Клон не изменился: true