ЛАБОРАТОРНАЯ РАБОТА №5

Курс: Объектно-ориентированное программирование.

Студент: Кузнецов Эрик Витальевич
Группа:6204-010302D
Преподаватель: Борисов Дмитрий Сергеевич

Задание 1

Ход выполнения работы:

1)Переопределен метод toString(): возвращает текстовое описание точки в формате (x; y), где x и y - координаты точки.

2)Переопределен метод equals(Object o):

Проверяет, что переданный объект является точкой

Сравнивает координаты с учетом точности чисел с плавающей точкой

Возвращает true только при точном совпадении координат

3)Переопределен метод hashCode():

Использован метод Double.doubleToLongBits() для преобразования double в long

Выполнено побитовое XOR между старшими и младшими 4 байтами каждого double

Результаты для x и y объединены через XOR

4)Переопределен метод clone():

Реализовано простое клонирование (без глубокого, так как точка не содержит ссылок на другие объекты)

Возвращает новый объект FunctionPoint с теми же координатами

```java
// 1 (переопределение методов object)

@Override
public String toString() {
    return "(" + x + "; " + y + ")";
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    FunctionPoint that = (FunctionPoint) o;
    return Double.compare(that.x, x) == 0 && Double.compare(that.y, y) == 0;//для дабл чисел compare
}

@Override
public int hashCode() {
    long xBits = Double.doubleToLongBits(x);//преобразует double в 64-битное целое
    long yBits = Double.doubleToLongBits(y);

    int xHash = (int)(xBits ^ (xBits >>> 32));//делаем XOR для исходного и сдвинутого значений
    int yHash = (int)(yBits ^ (yBits >>> 32));

    return xHash ^ yHash;
}
//4
@Override
public Object clone() {
    return new FunctionPoint(this);
}
```

Рисунок 1-изображение с конечным результатом для задания 1.

Задание 2.

Ход выполнения работы:

1)Переопределен метод toString(): возвращает строку в формате {(x1; y1), (x2; y2), ...}, содержащую все точки функции.

2)Переопределен метод equals(Object o):

Проверяет, что переданный объект реализует TabulatedFunction

При сравнении с ArrayTabulatedFunction обращается напрямую к элементам массива для оптимизации

Сравнивает количество точек и координаты каждой точки с учетом точности чисел с плавающей точкой

3)Переопределен метод hashCode():

Рассчитывается как побитовое XOR хэш-кодов всех точек функции

Дополнительно включает количество точек для различия функций с разным числом точек

4)Переопределен метод clone():

Реализовано глубокое клонирование

Создает новый массив точек и клонирует каждую точку отдельно

```java
// 2(переопределение методов)
//StringBuilder - это класс в Java для эффективного построения строк.
@Override
public String toString() {//StringBuilder-массив char[],который изменяется массив в динамике
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    for (int i = 0; i < pointsCount; i++) {
        sb.append(points[i].toString());
        if (i < pointsCount - 1) {
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;//instanceof - проверяет, является ли объект 'o' экземпляром класса TabulatedFunction

    TabulatedFunction that = (TabulatedFunction) o;

    if (this.getPointsCount() != that.getPointsCount()) return false;

    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction arrayThat = (ArrayTabulatedFunction) o;
        for (int i = 0; i < pointsCount; i++) {
            if (!this.points[i].equals(arrayThat.points[i])) {
                return false;
            }
        }
    } else {
        for (int i = 0; i < pointsCount; i++) {
            if (!this.getPoint(i).equals(that.getPoint(i))) {
                return false;
            }
        }
    }

    return true;// Если все проверки пройдены - true
}
```

```java
@Override
public int hashCode() {
    int hash = pointsCount;
    for (int i = 0; i < pointsCount; i++) {
        hash ^= points[i].hashCode();
    }
    return hash;
}

@Override
public Object clone() {
    FunctionPoint[] clonedPoints = new FunctionPoint[points.length];
    for (int i = 0; i < pointsCount; i++) {
        clonedPoints[i] = (FunctionPoint) points[i].clone();
    }

    ArrayTabulatedFunction cloned = new ArrayTabulatedFunction();
    cloned.points = clonedPoints;
    cloned.pointsCount = this.pointsCount;
    return cloned;
}
```

Рисунок 2,3-Снимок Экрана с конечным результатом для задания 2

Задание 3.

Ход выполнения работы:

1)Переопределен метод toString(): возвращает строковое представление всех точек связного списка в формате {(x1; y1), (x2; y2), ...}.

2)Переопределен метод equals(Object o):

Корректно работает с любым объектом типа TabulatedFunction

При сравнении с LinkedListTabulatedFunction использует прямое обращение к узлам списка для оптимизации

3)Переопределен метод hashCode():

Аналогично массиву, вычисляется как XOR хэш-кодов всех точек

Учитывает количество точек в списке

4)Переопределен метод clone():

Реализовано глубокое клонирование через "пересборку" нового списка.

Создаются новые узлы и клонируются точки без использования методов добавления для оптимизации производительности

```java
// 3(переопределение методов)

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    FunctionNode current = head.next;
    while (current != head) {
        sb.append(current.point.toString());
        if (current.next != head) {
            sb.append(", ");
        }
        current = current.next;
    }
    sb.append("}");
    return sb.toString();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction that = (TabulatedFunction) o;

    if (this.getPointsCount() != that.getPointsCount()) return false;

    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction listThat = (LinkedListTabulatedFunction) o;
        FunctionNode currentThis = this.head.next;
        FunctionNode currentThat = listThat.head.next;

        while (currentThis != this.head && currentThat != listThat.head) {
            if (!currentThis.point.equals(currentThat.point)) {
                return false;
            }
            currentThis = currentThis.next;
            currentThat = currentThat.next;
        }
```

```
        } else {// Общий случай для других реализаций TabulatedFunction
            for (int i = 0; i < getPointsCount(); i++) {
                if (!this.getPoint(i).equals(that.getPoint(i))) {
                    return false;
                }
            }
        }

        return true;
    }

    @Override
    public int hashCode() {
        int hash = getPointsCount();
        FunctionNode current = head.next;
        while (current != head) {
            hash ^= current.point.hashCode();
            current = current.next;
        }
        return hash;
    }

    @Override
    public Object clone() {
        LinkedListTabulatedFunction cloned = new LinkedListTabulatedFunction();

        FunctionNode current = head.next;
        while (current != head) {
            FunctionNode newNode = cloned.addNodeToTail();
            newNode.point = (FunctionPoint) current.point.clone();
            current = current.next;
        }

        return cloned;
    }
}
```

Рисунок 4,5-Снимок Экрана с конечным результатом для задания 3

Задание 4

Ход выполнения работы:

1)Добавлен метод clone() в интерфейс TabulatedFunction

2)Убедились, что все реализации интерфейса (ArrayTabulatedFunction, LinkedListTabulatedFunction) корректно реализуют метод clone()

3)Обеспечили совместимость с механизмом клонирования JVM

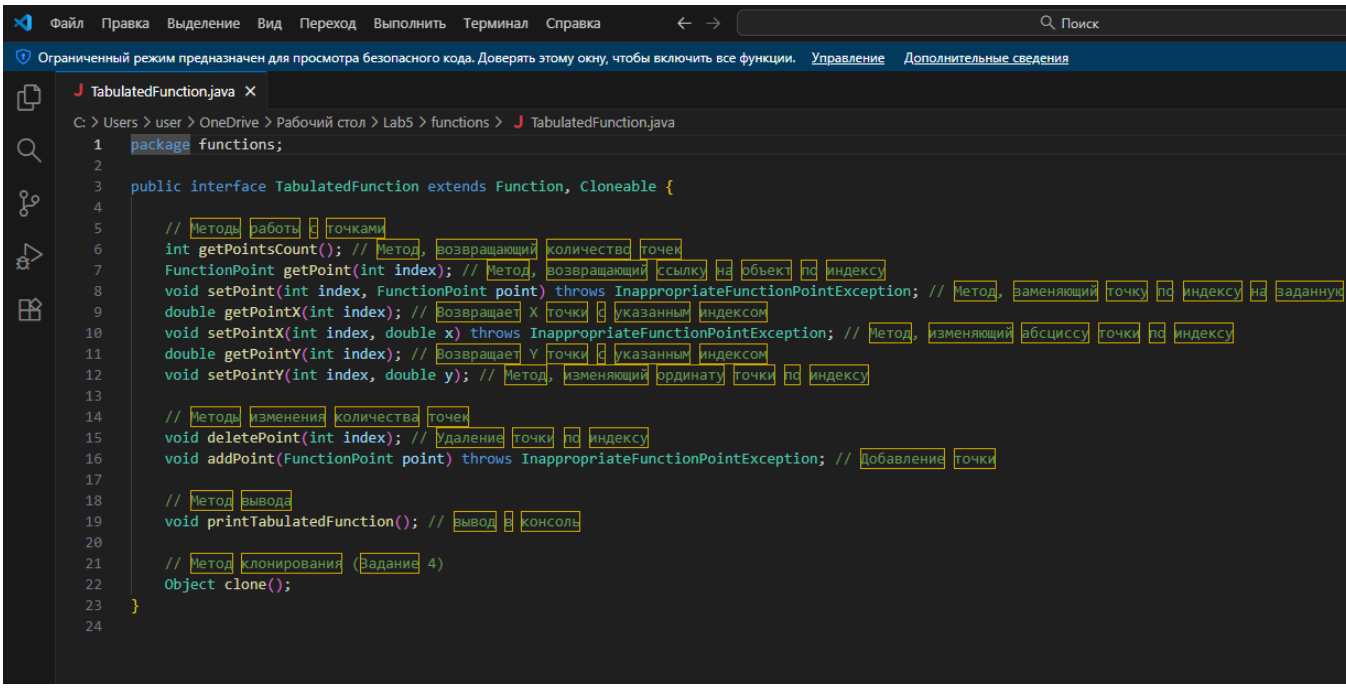


Рисунок 6-Снимок Экрана с конечным результатом для задания 4.

Задание 5

Ход выполнения работы:

1)Проверен метод toString():

Выведены строковые представления объектов ArrayTabulatedFunction и LinkedListTabulatedFunction в консоль

Убедились в корректности формата вывода

2)Проверен метод equals():

Вызван для одинаковых объектов одного класса - возвращает true

Вызван для разных объектов с одинаковыми данными - возвращает true

Вызван для объектов разных классов с одинаковыми данными - возращает true

Вызван для различных объектов - возвращает false

3)Проверен метод hashCode():

Выведены значения хэш-кодов для всех тестовых объектов

Убедились, что одинаковые объекты имеют одинаковые хэш-коды

Проверили согласованность equals() и hashCode()

Изменили координаты точки на 0.001 и убедились в изменении хэш-кода

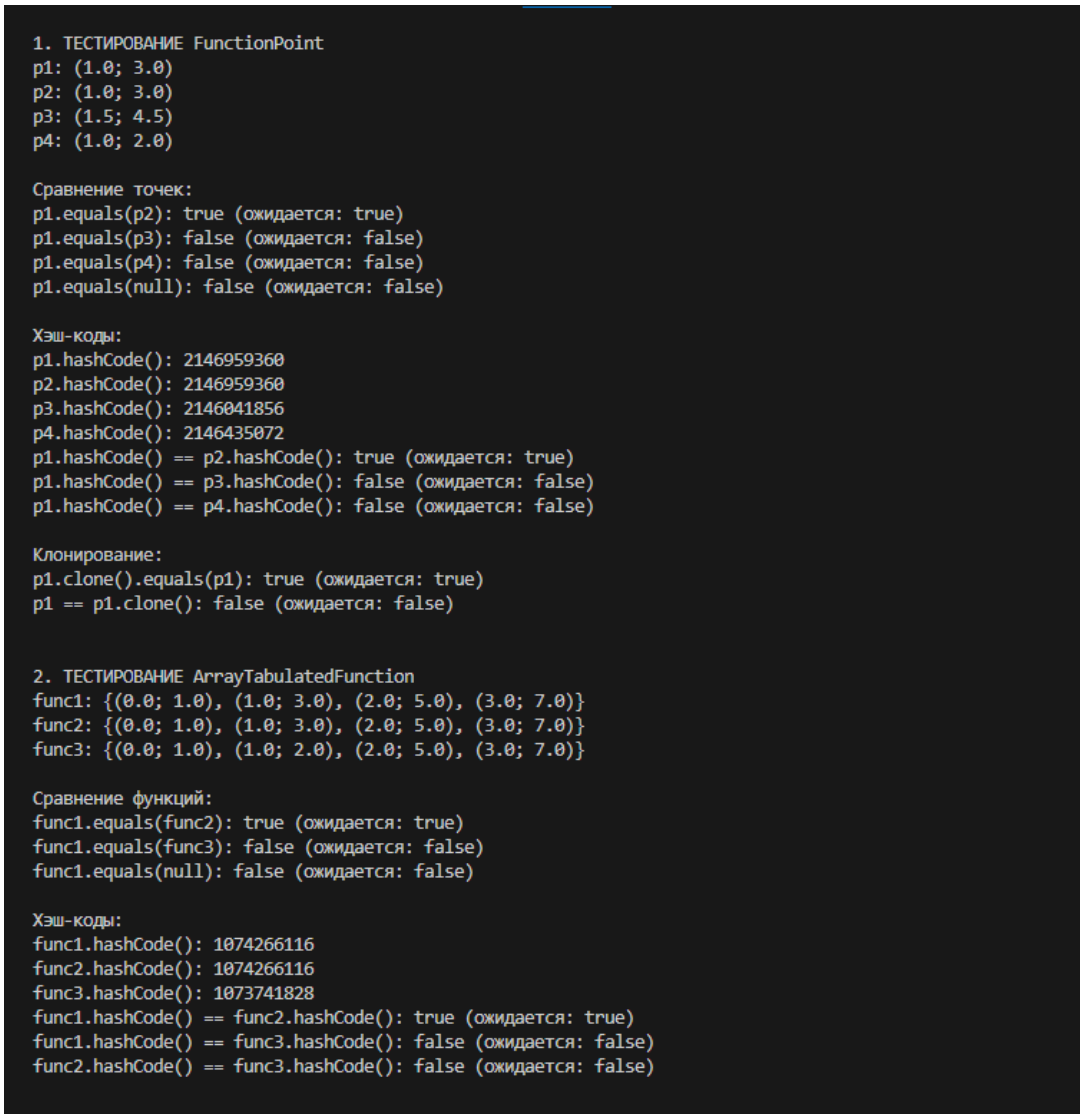
4)Проверен метод clone():

Выполнено клонирование объектов обоих классов

Изменены исходные объекты после клонирования

Убедились, что клонированные объекты не изменились (глубокое клонирование)

Проверили независимость исходных и клонированных объектов



```
3. ТЕСТИРОВАНИЕ LinkedListTabulatedFunction
listFunc1: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
listFunc2: {(0.0; 1.0), (1.0; 3.0), (2.0; 5.0), (3.0; 7.0)}
listFunc3: {(0.0; 1.0), (1.0; 2.0), (2.0; 5.0), (3.0; 7.0)}

Сравнение функций:
listFunc1.equals(listFunc2): true (ожидается: true)
listFunc1.equals(listFunc3): false (ожидается: false)
listFunc1.equals(func1): true (ожидается: true - одинаковые точки)

Хэш-коды:
listFunc1.hashCode(): 1074266116
listFunc2.hashCode(): 1074266116
listFunc3.hashCode(): 1073741828
listFunc1.hashCode() == listFunc2.hashCode(): true (ожидается: true)
listFunc1.hashCode() == listFunc3.hashCode(): false (ожидается: false)
listFunc2.hashCode() == listFunc3.hashCode(): false (ожидается: false)

Клонирование:
listFunc1.clone().equals(listFunc1): true (ожидается: true)
listFunc1 == listFunc1.clone(): false (ожидается: false)

Проверка глубокого клонирования (ArrayTabulatedFunction):
После изменения func1:
func1.getPointY(1): 999.0 (ожидается: 999.0)
func1Clone.getPointY(1): 3.0 (ожидается: 1.0)
Клон не изменился: true (ожидается: true)

Проверка глубокого клонирования (LinkedListTabulatedFunction):
После изменения listFunc1:
listFunc1.getPointY(1): 888.0 (ожидается: 888.0)
listFunc1Clone.getPointY(1): 3.0 (ожидается: 1.0)
Клон не изменился: true (ожидается: true)


4. ПРОВЕРКА СОГЛАСОВАННОСТИ equals() И hashCode()
testPoint1.equals(testPoint2): true
testPoint1.hashCode() == testPoint2.hashCode(): true
testPoint1.equals(testPoint3): false
testPoint1.hashCode() == testPoint3.hashCode(): false
Изменение хэш-кода при небольшом изменении координат: true


5. ТЕСТИРОВАНИЕ С РАЗНЫМ КОЛИЧЕСТВОМ ТОЧЕК
func1 (4 точки): {(0.0; 1.0), (1.0; 999.0), (2.0; 5.0), (3.0; 7.0)}
func4 (3 точки): {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0)}
func1.equals(func4): false (ожидается: false)
func1.hashCode() == func4.hashCode(): false (ожидается: false)
PS C:\Users\user\OneDrive\Рабочий стол\Lab5>
```

Рисунок 7,8-Снимок Экрана с конечным результатом для задания 5.