Лабораторная работа №5

Выполнил работу
Гапанюк Антон Андреевич
Группа: 6204-010302D

Самара, 2025 г.

Цель работы: расширить возможности классов, связанных с табулированными функциями, переопределив в них методы, унаследованные из класса Object.

Ход работы

Задание №1

Переопределим в классе **FunctionPoint** следующие методы:

- **String toString():** Будет возвращать текстовое описание точки. Например: (1.1; -7.5), где 1.1 и -7.5 – абсцисса и ордината точки соответственно.
- **boolean equals(Object o):** Будет возвращать **true** тогда и только тогда, когда переданный объект также является точкой и его координаты в точности совпадают с координатами объекта, у которого вызывается метод. Не забудем про корректное сравнение через эпсилон
- **int hashCode():** Будет возвращать значение хэш-кода для объекта точки.
- **Object clone():** Будет возвращать объект-копию для объекта точки.

```
// Переопределение метода toString()
    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }

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

        FunctionPoint that = (FunctionPoint) o;

        // Корректное сравнение чисел с плавающей точкой через машинный
эпсилон
        return Math.abs(that.x - x) < 1e-10 && Math.abs(that.y - y) < 1e-10;
    }

    // Переопределение метода hashCode()
    @Override
```

```
    public int hashCode() {
        long xBits = Double.doubleToLongBits(x);
        long yBits = Double.doubleToLongBits(y);

        // Комбинируем сразу long, потом преобразуем в int
        long combined = xBits ^ yBits;

        // Преобразуем long в int, комбинируя старшие и младшие биты
        return (int)(combined ^ (combined >>> 32));
    }

    // Переопределение метода clone()
    @Override
    public Object clone(){
        try {
            return new FunctionPoint(this);
        } catch (Exception e) {
            throw new AssertionError("Клонирование невозможно");
        }
    }
```

Задание №2

Переопределим в классе **ArrayTabulatedFunction** следующие методы:

• **String toString()**: Будет возвращать описание табулированной функции.

• **boolean equals(Object o)**: Будет возвращать **true** тогда и только тогда, когда переданный объект также является табулированной функцией (реализует интерфейс **TabulatedFunction**) и её набор точек в точности совпадает с набором точек функции, у которой вызывается метод. В случае если переданный объект является экземпляром класса **ArrayTabulatedFunction**, время работы метода будет сокращено за счёт прямого обращения к элементам состояния переданного объекта.

• **int hashCode():** Будет возвращать значение хэш-кода для объекта табулированной функции.

• **Object clone():** Будет возвращать объект-копию для объекта табулированной функции. Клонирование будет глубоким.

```
@Override
    public String toString(){
```

```
        StringBuffer curStr = new StringBuffer();
        curStr.append("{");

        for (int i = 0; i < points.length; ++i){
            curStr.append(points[i]);
            if (i < points.length - 1)
                curStr.append(", ");
        }
        curStr.append("}");

        return curStr.toString();
    }

    @Override
    public boolean equals(Object o){
        if (this == o) return true; // Проверка на идентичность ссылок
        if (o == null) return false; // Проверка на null

        // Оптимизированное сравнение для ArrayTabulatedFunction
        if (o instanceof ArrayTabulatedFunction) { // Проверяем принадлежит
ли объект классу
            ArrayTabulatedFunction other = (ArrayTabulatedFunction) o;

            // Быстрая проверка - разное кол-во точек
            if (this.points.length != other.points.length) return false;

            // Прямое сравнение массиво точек (быстрое)
            for (int i = 0; i < points.length; ++i){
                if (!this.points[i].equals(other.points[i])) return false;
            }
            return true;
        }

        // Универсальное сравнение для любой TabulatedFunction
        if (o instanceof TabulatedFunction){
            TabulatedFunction other = (TabulatedFunction) o;

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

```
            // Сравнение через getPoint (медленне, но универсальнее)
            for (int i = 0; i < points.length; ++i){
                FunctionPoint thisPoint = this.points[i];
                FunctionPoint otherPoint = other.getPoint(i);

                if (!thisPoint.equals(otherPoint)) return false;
            }
            return true;
        }
        // Объект несовместимого типа
        return false;
    }

    @Override
    public int hashCode(){
        int result = points.length; // Начинаем с кол-ва точек

        for (FunctionPoint point : points){
            result ^= point.hashCode(); // XOR с хэш-кодом каждой точки
        }
        return result; // Возвращаем итоговый хэш-код
    }

    @Override
    public Object clone(){
        try {
            // Создаем копию массива точек
            FunctionPoint[] clonedPoints = new FunctionPoint[points.length];
            for (int i = 0; i < points.length; ++i){
                clonedPoints[i] = (FunctionPoint) points[i].clone();
            }

            // Создаем новый объект с скопированными точками
            ArrayTabulatedFunction newArray = new
ArrayTabulatedFunction(clonedPoints);
            return newArray;

        } catch (Exception e) {
            throw new AssertionError("Клонирование невозможно");
```

```
        }
    }
```

Задание №3

Аналогично, переопределите методы **toString(), equals(), hashCode()** и **clone()** в классе **LinkedListTabulatedFunction.** Учтем, что:

• Метод **equals()** должен корректно работать при сравнении с любым объектом типа **TabulatedFunction**, а при сравнении с объектом типа **LinkedListTabulatedFunction** время работы метода должно быть сокращено за счёт возможности прямого обращения к полям переданного объекта.

• Клонирование в методе **clone()** тоже будет глубоким. Будем «пересобирать» новый объект списка.

```
@Override
    public String toString(){
        StringBuffer curStr = new StringBuffer();
        curStr.append("{");

        FunctionNode curNode = head.getNext();
        while (curNode != head){ // Пробегаемся по всему списку
            curStr.append(curNode.getPoint().toString());

            if (curNode.getNext() != head) curStr.append(", "); // Добавляем
запятые между пар координат точек
            curNode = curNode.getNext();
        }

        curStr.append("}");
        return curStr.toString();
    }

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

        // Оптимизированное сравнение
        if (o instanceof LinkedListTabulatedFunction){
            LinkedListTabulatedFunction other = (LinkedListTabulatedFunction)
o;
```

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

            // Прямое сравнение узлов списка
            FunctionNode thisNode = this.head.getNext();
            FunctionNode otherNode = other.head.getNext();

            while (thisNode != this.head && otherNode != other.head){
                if (!thisNode.getPoint().equals(otherNode.getPoint())) return
false;

                thisNode.getNext();
                otherNode.getNext();
            }
            return true;
        }

        // Универсальное сравнение для любой TabulatedFunction
        if (o instanceof TabulatedFunction){
            TabulatedFunction other = (TabulatedFunction) o;

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

            FunctionNode curNode = this.head.getNext(); // Для LinkedList
            int index = 0; // Для TabulatedFunction
            while (curNode != this.head){
                FunctionPoint thisPoint = curNode.getPoint();
                FunctionPoint otherPoint = other.getPoint(index);

                if (!thisPoint.equals(otherPoint)) return false;

                curNode = curNode.getNext();
                ++index;
            }
            return true;

        }
        return false;
```

```
    }

    @Override
    public int hashCode(){
        int result = pointsCount; // Начинаем с кол-ва точек

        FunctionNode curNode = head.getNext();
        while (curNode != head){
            result ^= curNode.getPoint().hashCode(); // XOR с хэш-кодом
каждой точки
            curNode = curNode.getNext();
        }
        return result;
    }

    @Override
    public Object clone(){
        try {
            // Создаем массив точек из исходного списка
            FunctionPoint[] pointsArray = new FunctionPoint[pointsCount];

            FunctionNode curNode = head.getNext();
            int index = 0;
            while (curNode != head){
                pointsArray[index] = (FunctionPoint)
curNode.getPoint().clone();
                curNode = curNode.getNext();
                ++index;
            }

            // Создаем новый объект используя конструктор с массивом точек
            return new LinkedListTabulatedFunction(pointsArray);

        } catch (Exception e) {
            throw new AssertionError("Клонирование невозможно");
        }
    }
```

Задание №4

Сделаем так, чтобы все объекты типа TabulatedFunction были клонируемыми с точки зрения JVM и внесем метод clone() в этот интерфейс.

Добавим **Cloneable** и внесем метод **Object clone();**

```
public interface TabulatedFunction extends Function, Cloneable
// Лабораторная работа №5: метод clone() в интерфейс
    Object clone();
```

Задание №5

Проверим работу написанных методов.

```
=== ТЕСТИРОВАНИЕ ВСЕХ МЕТОДОВ ===

--- ТЕСТ toString() ---
ArrayTabulatedFunction: {(0.0, 1.0), (1.0, 3.0), (2.0, 5.0)}
LinkedListTabulatedFunction: {(0.0, 1.0), (1.0, 3.0), (2.0, 5.0)}

--- ТЕСТ equals() ---
arrayFunc1.equals(arrayFunc2): true
arrayFunc1.equals(arrayFunc3): false
arrayFunc1.equals(arrayFunc4): false
linkedFunc1.equals(linkedFunc2): true
arrayFunc1.equals(linkedFunc1): true
arrayFunc1.equals("строка"): false
arrayFunc1.equals(null): false

--- ТЕСТ hashCode() ---
arrayFunc1.hashCode(): 1075576835
arrayFunc2.hashCode(): 1075576835
arrayFunc3.hashCode(): 1071906818
arrayFunc4.hashCode(): 1075314691
linkedFunc1.hashCode(): 1075576835
linkedFunc2.hashCode(): 1075576835

--- СОГЛАСОВАННОСТЬ equals() и hashCode() ---
arrayFunc1.equals(arrayFunc2) && arrayFunc1.hashCode() == arrayFunc2.hashCode(): true
arrayFunc1.equals(arrayFunc3) || arrayFunc1.hashCode() != arrayFunc3.hashCode(): true

--- ТЕСТ ИЗМЕНЕНИЯ ХЭШ-КОДА ---
Исходный hashCode: 1075576835
После изменения Y[1] с 3.0 на 3.001: 161897530
Хэш-код изменился: true

--- ТЕСТ clone() ---
arrayFunc1.equals(arrayClone): true
arrayFunc1 == arrayClone: false
linkedFunc1.equals(linkedClone): true
linkedFunc1 == linkedClone: false

--- ТЕСТ ГЛУБОКОГО КЛОНИРОВАНИЯ ---
После изменения клонов:
arrayFunc1.getPointY(0): 1.0 (оригинал не изменился)
arrayClone.getPointY(0): 999.0 (клон изменился)
linkedFunc1.getPointY(0): 1.0 (оригинал не изменился)
linkedClone.getPointY(0): 888.0 (клон изменился)
arrayFunc1.getPoint(0) == arrayClone.getPoint(0): false (разные объекты точек)
linkedFunc1.getPoint(0) == linkedClone.getPoint(0): false (разные объекты точек)

--- ТЕСТ РАБОТЫ С TabulatedFunction ИНТЕРФЕЙСОМ ---
TabulatedFunction интерфейс работает: true
```

Примечание – Код main

```java
public class Main {
    public static void main(String[] args){
        testAllMethods();
    }
```

```
    public static void testAllMethods() {
        System.out.println("=== ТЕСТИРОВАНИЕ ВСЕХ МЕТОДОВ ===");

        // Создаем тестовые данные
        FunctionPoint[] points1 = {
            new FunctionPoint(0.0, 1.0),
            new FunctionPoint(1.0, 3.0),
            new FunctionPoint(2.0, 5.0)
        };

        FunctionPoint[] points2 = {
            new FunctionPoint(0.0, 1.0),
            new FunctionPoint(1.0, 3.0),
            new FunctionPoint(2.0, 5.0)
        };

        FunctionPoint[] points3 = {
            new FunctionPoint(0.0, 1.0),
            new FunctionPoint(2.0, 5.0)  // Меньше точек
        };

        FunctionPoint[] points4 = {
            new FunctionPoint(0.0, 1.0),
            new FunctionPoint(1.0, 3.5), // Измененное значение Y
            new FunctionPoint(2.0, 5.0)
        };

        // Создаем объекты для тестирования
        ArrayTabulatedFunction arrayFunc1 = new
ArrayTabulatedFunction(points1);
        ArrayTabulatedFunction arrayFunc2 = new
ArrayTabulatedFunction(points2);
        ArrayTabulatedFunction arrayFunc3 = new
ArrayTabulatedFunction(points3);
        ArrayTabulatedFunction arrayFunc4 = new
ArrayTabulatedFunction(points4);

        LinkedListTabulatedFunction linkedFunc1 = new
LinkedListTabulatedFunction(points1);
```

```
        LinkedListTabulatedFunction linkedFunc2 = new
LinkedListTabulatedFunction(points2);
        LinkedListTabulatedFunction linkedFunc3 = new
LinkedListTabulatedFunction(points3);

        System.out.println("\n--- ТЕСТ toString() ---");
        System.out.println("ArrayTabulatedFunction: " +
arrayFunc1.toString());
        System.out.println("LinkedListTabulatedFunction: " +
linkedFunc1.toString());

        System.out.println("\n--- ТЕСТ equals() ---");
        System.out.println("arrayFunc1.equals(arrayFunc2): " +
arrayFunc1.equals(arrayFunc2)); // true
        System.out.println("arrayFunc1.equals(arrayFunc3): " +
arrayFunc1.equals(arrayFunc3)); // false
        System.out.println("arrayFunc1.equals(arrayFunc4): " +
arrayFunc1.equals(arrayFunc4)); // false
        System.out.println("linkedFunc1.equals(linkedFunc2): " +
linkedFunc1.equals(linkedFunc2)); // true
        System.out.println("arrayFunc1.equals(linkedFunc1): " +
arrayFunc1.equals(linkedFunc1)); // true
        System.out.println("arrayFunc1.equals(\"строка\"): " +
arrayFunc1.equals("строка")); // false
        System.out.println("arrayFunc1.equals(null): " +
arrayFunc1.equals(null)); // false

        System.out.println("\n--- ТЕСТ 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("arrayFunc4.hashCode(): " +
arrayFunc4.hashCode());
        System.out.println("linkedFunc1.hashCode(): " +
linkedFunc1.hashCode());
        System.out.println("linkedFunc2.hashCode(): " +
linkedFunc2.hashCode());
```

```
        // Проверка согласованности equals() и hashCode()
        System.out.println("\n--- СОГЛАСОВАННОСТЬ equals() и hashCode() ---
");
        System.out.println("arrayFunc1.equals(arrayFunc2) &&
arrayFunc1.hashCode() == arrayFunc2.hashCode(): " +
                           (arrayFunc1.equals(arrayFunc2) &&
arrayFunc1.hashCode() == arrayFunc2.hashCode()));
        System.out.println("arrayFunc1.equals(arrayFunc3) ||
arrayFunc1.hashCode() != arrayFunc3.hashCode(): " +
                           (!arrayFunc1.equals(arrayFunc3) ||
arrayFunc1.hashCode() != arrayFunc3.hashCode()));

        System.out.println("\n--- ТЕСТ ИЗМЕНЕНИЯ ХЭШ-КОДА ---");
        int originalHash = arrayFunc1.hashCode();
        System.out.println("Исходный hashCode: " + originalHash);

        // Незначительно изменяем объект
        arrayFunc1.setPointY(1, 3.001); // Изменяем на несколько тысячных
        int modifiedHash = arrayFunc1.hashCode();
        System.out.println("После изменения Y[1] с 3.0 на 3.001: " +
modifiedHash);
        System.out.println("Хэш-код изменился: " + (originalHash !=
modifiedHash));

        // Восстанавливаем исходное значение
        arrayFunc1.setPointY(1, 3.0);

        System.out.println("\n--- ТЕСТ clone() ---");

        // Тест ArrayTabulatedFunction
        ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction)
arrayFunc1.clone();
        System.out.println("arrayFunc1.equals(arrayClone): " +
arrayFunc1.equals(arrayClone));
        System.out.println("arrayFunc1 == arrayClone: " + (arrayFunc1 ==
arrayClone));

        // Тест LinkedListTabulatedFunction
```

```
        LinkedListTabulatedFunction linkedClone =
(LinkedListTabulatedFunction) linkedFunc1.clone();
        System.out.println("linkedFunc1.equals(linkedClone): " +
linkedFunc1.equals(linkedClone));
        System.out.println("linkedFunc1 == linkedClone: " + (linkedFunc1 ==
linkedClone));

        System.out.println("\n--- ТЕСТ ГЛУБОКОГО КЛОНИРОВАНИЯ ---");

        // Сохраняем исходные значения
        double originalArrayY = arrayFunc1.getPointY(0);
        double originalLinkedY = linkedFunc1.getPointY(0);

        // Изменяем клоны
        arrayClone.setPointY(0, 999.0);
        linkedClone.setPointY(0, 888.0);

        System.out.println("После изменения клонов:");
        System.out.println("arrayFunc1.getPointY(0): " +
arrayFunc1.getPointY(0) + " (оригинал не изменился)");
        System.out.println("arrayClone.getPointY(0): " +
arrayClone.getPointY(0) + " (клон изменился)");
        System.out.println("linkedFunc1.getPointY(0): " +
linkedFunc1.getPointY(0) + " (оригинал не изменился)");
        System.out.println("linkedClone.getPointY(0): " +
linkedClone.getPointY(0) + " (клон изменился)");

        // Проверка глубокого копирования точек
        System.out.println("arrayFunc1.getPoint(0) == arrayClone.getPoint(0):
" +
                           (arrayFunc1.getPoint(0) == arrayClone.getPoint(0)) +
" (разные объекты точек)");
        System.out.println("linkedFunc1.getPoint(0) ==
linkedClone.getPoint(0): " +
                           (linkedFunc1.getPoint(0) == linkedClone.getPoint(0))
+ " (разные объекты точек)");

        System.out.println("\n--- ТЕСТ РАБОТЫ С TabulatedFunction ИНТЕРФЕЙСОМ
---");
        TabulatedFunction tabulatedFunc = arrayFunc1;
```

```
        TabulatedFunction tabulatedClone = (TabulatedFunction)
tabulatedFunc.clone();
        System.out.println("TabulatedFunction интерфейс работает: " +
tabulatedFunc.equals(tabulatedClone));
    }
}
```