Лабораторная работа № 5

Выполнила: Оленина Арина Игоревна
группа 6204-010302D

Оглавление

Задание 1

Я переопределила в классе FunctionPoint следующие методы:
• **String toString():** возвращает текстовое описание точки. Например: (1.1; -7.5), где 1.1 и -7.5 – абсцисса и ордината точки соответственно.
• **boolean equals(Object o):** возвращает true тогда и только тогда, когда переданный объект также является точкой и его координаты в точности совпадают с координатами объекта, у которого вызывается метод.
• **int hashCode():** возвращает значение хэш-кода для объекта точки. Я воспользовалась простейшей реализацией, основанной на применении операции исключающего ИЛИ (XOR). Рассчитала хэш-код как побитовое XOR для набора значений типа int. Этот набор включает в себя всю информацию, описывающую состояние объекта, т.е. два значения координат. Поскольку они имеют тип double, я привела эту информацию к типу int, представив одно значение типа double (8 байт) как два значения типа int (4 байта и 4 байта). Сделала это с помощью метода Double.doubleToLongBits(), оператора побитового И (&) (для выделения младших четырёх байтов) и оператора битового логического сдвига (>>) (для выделения старших четырёх байтов).
• **Object clone():** возвращает объект-копию для объекта точки. Достаточно простого клонирования, так как точка не имеет ссылок на другие объекты.
Результат:

```
@Override
public String toString() { // Возвращаем точку в виде (x; y)
    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 && // Используем
сравнение с учетом погрешности для double
            Double.compare(that.y, y) == 0;
}

@Override
public int hashCode() {
    // Преобразуем double в long битовое представление
    long xBits = Double.doubleToLongBits(x);
    long yBits = Double.doubleToLongBits(y);

    // Разбиваем каждый long на два int (старшие и младшие 4
байта)
    int x1 = (int)(xBits & 0xFFFFFFFF);       // Младшие 4 байта
x
```

```
    int x2 = (int)(xBits >>> 32);            // Старшие 4 байта x
    int y1 = (int)(yBits & 0xFFFFFFFF);       // Младшие 4 байта
y
    int y2 = (int)(yBits >>> 32);            // Старшие 4 байта y

    // Комбинируем с помощью XOR
    return x1 ^ x2 ^ y1 ^ y2;
}

@Override
public Object clone() {
    return new FunctionPoint(this); // Возвращает объект-копию
для объекта точки
}
```

Задание 2

Я переопределила в классе ArrayTabulatedFunction следующие методы:
• **String toString():** возвращает описание табулированной функции. Например: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}, где в круглых скобках указываются координаты точек.
• **boolean equals(Object o):** возвращает 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, я сократила время работы метода за счёт прямого обращения к элементам состояния переданного объекта.
• **int hashCode():** возвращает значение хэш-кода для объекта табулированной функции. Я снова воспользовалась простейшей реализацией, основанной на применении операции исключающего ИЛИ (XOR). В этом случае хэш-код рассчитывается как побитовое XOR для набора значений типа int. В данный набор входят хэш-коды всех точек табулированной функции, а также количество точек в функции. Последнее нужно для того, чтобы значения хэш-кода были различны для функций, отличающихся наличием нулевой точки (например, {(-1; 1), (0; 0), (1, 1)} и {(-1; 1), (1, 1)}).
• **Object clone():** возвращает объект-копию для объекта табулированной функции. Поскольку табулированная функция ссылается на другие объекты, клонирование глубокое

```
@Override
public String toString() { // Вывод табулированной функции в
сроку
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    for (int i = 0; i < pointsCount; i++) {
        sb.append("(").append(points[i].getX())
                .append(";
").append(points[i].getY()).append(")");
```

```
        if (i < pointsCount - 1) {
            sb.append(", ");
        }
    }
    sb.append("}");
    return sb.toString();
}
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null) return false;

    // Если объект - ArrayTabulatedFunction, используется
оптимизированное сравнение
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction other = (ArrayTabulatedFunction)
o;
        if (this.pointsCount != other.pointsCount) return false;

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

    // Если объект - другой реализации TabulatedFunction,
используется общий подход
    if (o instanceof TabulatedFunction) {
        TabulatedFunction other = (TabulatedFunction) o;
        if (this.getPointsCount() != other.getPointsCount())
return false;

        // Сравниваем через getPoint()
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = this.getPoint(i);
            FunctionPoint otherPoint = other.getPoint(i);
            if (!thisPoint.equals(otherPoint)) {
                return false;
            }
        }
        return true;
    }

    return false;
}
@Override
public int hashCode() {
    int hash = pointsCount; // Начало с количества точек
    // Комбинация хэш-кодов всех точек через XOR
    for (int i = 0; i < pointsCount; i++) {
```

```
            hash ^= points[i].hashCode();
        }
        return hash;
    }

    @Override
    public Object clone() { // Глубокое копирование
        // Создание массива
        FunctionPoint[] realPoints = new
    FunctionPoint[this.pointsCount];
        for (int i = 0; i < this.pointsCount; i++) {
            realPoints[i] = new FunctionPoint(this.points[i]);
        }

        // Временный объект через конструктор
        ArrayTabulatedFunction cloned = new
    ArrayTabulatedFunction(realPoints);

        return cloned;
    }
```

Задание 3

Аналогично, переопределила
методы toString(), equals(), hashCode() и clone() в
классе LinkedListTabulatedFunction. При написании методов учла следующие особенности.

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

• Клонирование в методе clone() тоже глубокое, однако классическое глубокое клонирование в данном случае не совсем разумно. Если бы я сделала объекты класса FunctionNode клонируемыми, после их клонирования значения полей ссылок пришлось бы изменить (т.к. они будут ссылаться на объекты из исходного списка), и значение ссылающегося на объект точки поля тоже пришлось бы изменить (т.к. его нужно будет заменить клоном объекта точки). Поэтому проще оказалось «пересобрать» новый объект списка, причём сделала это без использования методов добавления в список, т.к. это привело бы к выполнению большого количества нерезультативных операций и заметно сказалось бы на скорости выполнения программы.

Результат:

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    FunctionNode current = head.next;
```

```
    int count = 0;
    while (current != head) {
        sb.append("(").append(current.point.getX())
                .append(";
").append(current.point.getY()).append(")");
        if (count < pointsCount - 1) {
            sb.append(", ");
        }
        current = current.next;
        count++;
    }
    sb.append("}");
    return sb.toString();
}
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction other = (TabulatedFunction) o;

    // Проверка количества точек
    if (this.getPointsCount() != other.getPointsCount()) return
false;

    // Оптимизация для LinkedListTabulatedFunction
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction otherList =
(LinkedListTabulatedFunction) o;

        // Прямое сравнение узлов двух списков
        FunctionNode currentThis = this.head.next;
        FunctionNode currentOther = otherList.head.next;

        while (currentThis != head && currentOther !=
otherList.head) {
            if (!currentThis.point.equals(currentOther.point)) {
                return false;
            }
            currentThis = currentThis.next;
            currentOther = currentOther.next;
        }
        return true;
    }

    // Общий случай для любой TabulatedFunction
    for (int i = 0; i < this.getPointsCount(); i++) {
        if (!this.getPoint(i).equals(other.getPoint(i))) {
            return false;
        }
    }
    return true;
}
```

```java
@Override
public int hashCode() {
    int hash = pointsCount; // Начало с количества точек

    // Комбинирование хэш-кодов всех точек через XOR
    FunctionNode current = head.next;
    while (current != head) {
        hash ^= current.point.hashCode();
        current = current.next;
    }

    return hash;
}
@Override
public Object clone() {
    // Временный объект
    LinkedListTabulatedFunction cloned = new
LinkedListTabulatedFunction();
    // Пустой список
    cloned.head = new FunctionNode(null);
    cloned.head.next = cloned.head;
    cloned.head.prev = cloned.head;
    cloned.pointsCount = 0;

    // Если исходный список пуст, возврат пустого клона
    if (this.pointsCount == 0) {
        return cloned;
    }

    // Пересборка списка вручную без методов добавления
    FunctionNode currentOriginal = this.head.next;
    FunctionNode lastClonedNode = cloned.head;

    while (currentOriginal != this.head) {
        // Новая точка
        FunctionPoint newPoint = new FunctionPoint(
                currentOriginal.point.getX(),
                currentOriginal.point.getY()
        );

        // Новый узел с новой точкой
        FunctionNode newNode = new FunctionNode(newPoint);

        // Прямое связывание узлов без методов добавления
        newNode.prev = lastClonedNode;
        newNode.next = cloned.head;
        lastClonedNode.next = newNode;
        cloned.head.prev = newNode;

        // Переход к следующему узлу
        lastClonedNode = newNode;
        currentOriginal = currentOriginal.next;
        cloned.pointsCount++;
```

```
    }

    return cloned;
}
```

Задание 4

Сделала так, чтобы все объекты типа TabulatedFunction были клонируемыми с точки зрения JVM и внесла метод clone() в этот интерфейс.
Добавила:

```
public interface TabulatedFunction extends Function, Cloneable {
      Object clone();
}
```

Задание 5

Проверила работу написанных методов.

• Проверила работу метода toString() для объектов типов ArrayTabulatedFunction и LinkedListTabulatedFunction, выведя строковое представление объектов в консоль.
• Проверила работу метода equals(), вызывая его для одинаковых и различающихся объектов одинаковых и различающихся классов.
• Проверила работу метода hashCode(), выведя в консоль его значения для всех использованных объектов. Убедилась в согласованности работы методов equals() и hashCode(). Также попробовала незначительно изменить один из объектов (изменила одну из координат точки с 1.0 до 1.001) и проверила, как изменилось значение хэш-кода объекта.
• Проверила работу метода clone() для объектов обоих классов табулированных функций. Убедилась, что произведено именно глубокое клонирование: для этого после клонирования изменила исходные объекты и проверила, что объекты-клоны не изменились.
Результат:

```
import functions.*;
import functions.basic.*;
import functions.meta.*;
import java.io.*;

public class Main {
    public static void main(String[] args) {

        // Создание тестовых данных
        FunctionPoint[] points1 = {
                new FunctionPoint(0.0, 0.0),
                new FunctionPoint(1.0, 1.0),
                new FunctionPoint(2.0, 8.0)
        };
```

```
        FunctionPoint[] points2 = {
                new FunctionPoint(0.0, 0.0),
                new FunctionPoint(1.0, 1.0),
                new FunctionPoint(2.0, 8.0)
        };

        FunctionPoint[] points3 = {
                new FunctionPoint(0.0, 0.0),
                new FunctionPoint(2.0, 8.0)  // Пропущена точка
        };

        FunctionPoint[] points4 = {
                new FunctionPoint(0.0, 0.0),
                new FunctionPoint(1.0, 1.001), // Незначительное
изменение
                new FunctionPoint(2.0, 8.0)
        };

        // Создание объектов 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("Тест toString()");
        testToString(arrayFunc1, linkedFunc1);

        System.out.println("\nТест equals()");
        testEquals(arrayFunc1, arrayFunc2, arrayFunc3,
linkedFunc1, linkedFunc2, linkedFunc3);

        System.out.println("\nТест hashCode()");
        testHashCode(arrayFunc1, arrayFunc2, arrayFunc4,
linkedFunc1, linkedFunc2);

        System.out.println("\nТест clone()");
        testClone(arrayFunc1, linkedFunc1);
    }

    // Тестирование метода toString()
    private static void testToString(ArrayTabulatedFunction
```

```
arrayFunc,
                                    LinkedListTabulatedFunction
linkedFunc) {
        System.out.println("ArrayTabulatedFunction toString(): "
+ arrayFunc.toString());
        System.out.println("LinkedListTabulatedFunction
toString(): " + linkedFunc.toString());
    }

    // Тестирование метода equals()
    private static void testEquals(ArrayTabulatedFunction
array1, ArrayTabulatedFunction array2,
                                   ArrayTabulatedFunction
array3, LinkedListTabulatedFunction linked1,
                                   LinkedListTabulatedFunction
linked2, LinkedListTabulatedFunction linked3) {

        System.out.println("Сравнение одинаковых объектов");
        System.out.println("array1.equals(array2): " +
array1.equals(array2)); // true
        System.out.println("linked1.equals(linked2): " +
linked1.equals(linked2)); // true

        System.out.println("\nСравнение разных объектов");
        System.out.println("array1.equals(array3): " +
array1.equals(array3)); // false
        System.out.println("linked1.equals(linked3): " +
linked1.equals(linked3)); // false

        System.out.println("\nСравнение объектов разных
классов");
        System.out.println("array1.equals(linked1): " +
array1.equals(linked1)); // true
        System.out.println("linked1.equals(array1): " +
linked1.equals(array1)); // true

        System.out.println("\nСравнение с null");
        System.out.println("array1.equals(null): " +
array1.equals(null)); // false

        System.out.println("\nСравнение с самим собой");
        System.out.println("array1.equals(array1): " +
array1.equals(array1)); // true
    }

    // Тестирование метода hashCode()
    private static void testHashCode(ArrayTabulatedFunction
array1, ArrayTabulatedFunction array2,
                                     ArrayTabulatedFunction
array4, LinkedListTabulatedFunction linked1,
                                     LinkedListTabulatedFunction
linked2) {
```

```
        System.out.println("Хэш-коды одинаковых объектов");
        int arrayHash1 = array1.hashCode();
        int arrayHash2 = array2.hashCode();
        int linkedHash1 = linked1.hashCode();
        int linkedHash2 = linked2.hashCode();

        System.out.println("Array1 hashCode: " + arrayHash1);
        System.out.println("Array2 hashCode: " + arrayHash2);
        System.out.println("Linked1 hashCode: " + linkedHash1);
        System.out.println("Linked2 hashCode: " + linkedHash2);

        System.out.println("array1.hashCode == array2.hashCode:
" + (arrayHash1 == arrayHash2)); // true
        System.out.println("linked1.hashCode ==
linked2.hashCode: " + (linkedHash1 == linkedHash2)); // true

        System.out.println("\nПроверка согласованности
equals/hashCode");
        System.out.println("Если equals=true, то hashCode равны:
" +
                (array1.equals(array2) == (arrayHash1 ==
arrayHash2))); // true

        System.out.println("\nХэш-код после незначительного
изменения");
        int arrayHash4 = array4.hashCode();
        System.out.println("Array4 (с изменением) hashCode: " +
arrayHash4);
        System.out.println("Array1.hashCode == Array4.hashCode:
" + (arrayHash1 == arrayHash4)); // false
        System.out.println("Изменение хэш-кода: " +
Math.abs(arrayHash1 - arrayHash4));
    }

    // Тестирование метода clone()
    private static void testClone(ArrayTabulatedFunction
arrayFunc,
                                  LinkedListTabulatedFunction
linkedFunc) {

        System.out.println("Тестирование ArrayTabulatedFunction
clone()");
        ArrayTabulatedFunction arrayClone =
(ArrayTabulatedFunction) arrayFunc.clone();

        System.out.println("Исходный Array: " + arrayFunc);
        System.out.println("Клон Array: " + arrayClone);
        System.out.println("Равенство до изменения: " +
arrayFunc.equals(arrayClone)); // true

        // Замена оригинала
        arrayFunc.setPointY(0, 999.0);
        System.out.println("После изменения оригинала:");
```

```
        System.out.println("Исходный Array Y[0]: " +
arrayFunc.getPointY(0)); // 999.0
        System.out.println("Клон Array Y[0]: " +
arrayClone.getPointY(0)); // 0.0
        System.out.println("Равенство после изменения: " +
arrayFunc.equals(arrayClone)); // false
        System.out.println("Разные объекты в памяти: " +
(arrayFunc != arrayClone)); // true

        System.out.println("\nТестирование
LinkedListTabulatedFunction clone()");
        LinkedListTabulatedFunction linkedClone =
(LinkedListTabulatedFunction) linkedFunc.clone();

        System.out.println("Исходный Linked: " + linkedFunc);
        System.out.println("Клон Linked: " + linkedClone);
        System.out.println("Равенство до изменения: " +
linkedFunc.equals(linkedClone)); // true

        // Замена оригинала
        linkedFunc.setPointY(1, 888.0);
        System.out.println("После изменения оригинала:");
        System.out.println("Исходный Linked Y[1]: " +
linkedFunc.getPointY(1)); // 888.0
        System.out.println("Клон Linked Y[1]: " +
linkedClone.getPointY(1)); // 1.0
        System.out.println("Равенство после изменения: " +
linkedFunc.equals(linkedClone)); // false
        System.out.println("Разные объекты в памяти: " +
(linkedFunc != linkedClone)); // true

        System.out.println("\nПроверка глубокого копирования");
        System.out.println("Array: изменение оригинала не
затронуло клон - " +
                (arrayFunc.getPointY(0) !=
arrayClone.getPointY(0))); // true
        System.out.println("Linked: изменение оригинала не
затронуло клон - " +
                (linkedFunc.getPointY(1) !=
linkedClone.getPointY(1))); // true
    }
}
```

Вывод в консоль:

```
Тест toString()
ArrayTabulatedFunction toString(): {(0.0; 0.0), (1.0; 1.0), (2.0; 8.0)}
LinkedListTabulatedFunction toString(): {(0.0; 0.0), (1.0; 1.0), (2.0; 8.0)}

Тест equals()
Сравнение одинаковых объектов
array1.equals(array2): true
```

```
linked1.equals(linked2): true

Сравнение разных объектов
array1.equals(array3): false
linked1.equals(linked3): false

Сравнение объектов разных классов
array1.equals(linked1): true
linked1.equals(array1): true

Сравнение с null
array1.equals(null): false

Сравнение с самим собой
array1.equals(array1): true

Тест hashCode()
Хэш-коды одинаковых объектов
Array1 hashCode: 2097155
Array2 hashCode: 2097155
Linked1 hashCode: 2097155
Linked2 hashCode: 2097155
array1.hashCode == array2.hashCode: true
linked1.hashCode == linked2.hashCode: true

Проверка согласованности equals/hashCode
Если equals=true, то hashCode равны: true

Хэш-код после незначительного изменения
Array4 (с изменением) hashCode: -1823164303
Array1.hashCode == Array4.hashCode: false
Изменение хэш-кода: 1825261458

Тест clone()
Тестирование ArrayTabulatedFunction clone()
Исходный Array: {(0.0; 0.0), (1.0; 1.0), (2.0; 8.0)}
Клон Array: {(0.0; 0.0), (1.0; 1.0), (2.0; 8.0)}
Равенство до изменения: true
После изменения оригинала:
Исходный Array Y[0]: 999.0
Клон Array Y[0]: 0.0
Равенство после изменения: false
Разные объекты в памяти: true

Тестирование LinkedListTabulatedFunction clone()
```

```
Исходный Linked: {(0.0; 0.0), (1.0; 1.0), (2.0; 8.0)}
Клон Linked: {(0.0; 0.0), (1.0; 1.0), (2.0; 8.0)}
Равенство до изменения: true
После изменения оригинала:
Исходный Linked Y[1]: 888.0
Клон Linked Y[1]: 1.0
Равенство после изменения: false
Разные объекты в памяти: true

Проверка глубокого копирования
Array: изменение оригинала не затронуло клон - true
Linked: изменение оригинала не затронуло клон - true
```