Лабораторная работа № 5

Титульный лист

Дисциплина: Объектно-ориентированное программирование
Выполнил: Мостовщиков Владимир Витальевич
Группа: 6204-010302D
Преподаватель: Борисов Дмитрий Сергеевич
Год: 2025

Содержание:

Задание 1. Переопределение методов в классе FunctionPoint

В классе FunctionPoint я переопределил методы toString(), equals(Object o), hashCode() и clone(). Для метода toString() я создал удобное текстовое

представление точки в формате "(x; y)", что позволяет легко читать и выводить информацию о точках в консоли и отчетах.

Для метода equals() я реализовал сравнение точек по координатам. Две точки считаются равными, если совпадают обе координаты x и y. Я использовал точное сравнение чисел через Double.compare() без допусков погрешности, чтобы гарантировать математически корректное поведение - если A=B и B=C, то A=C.

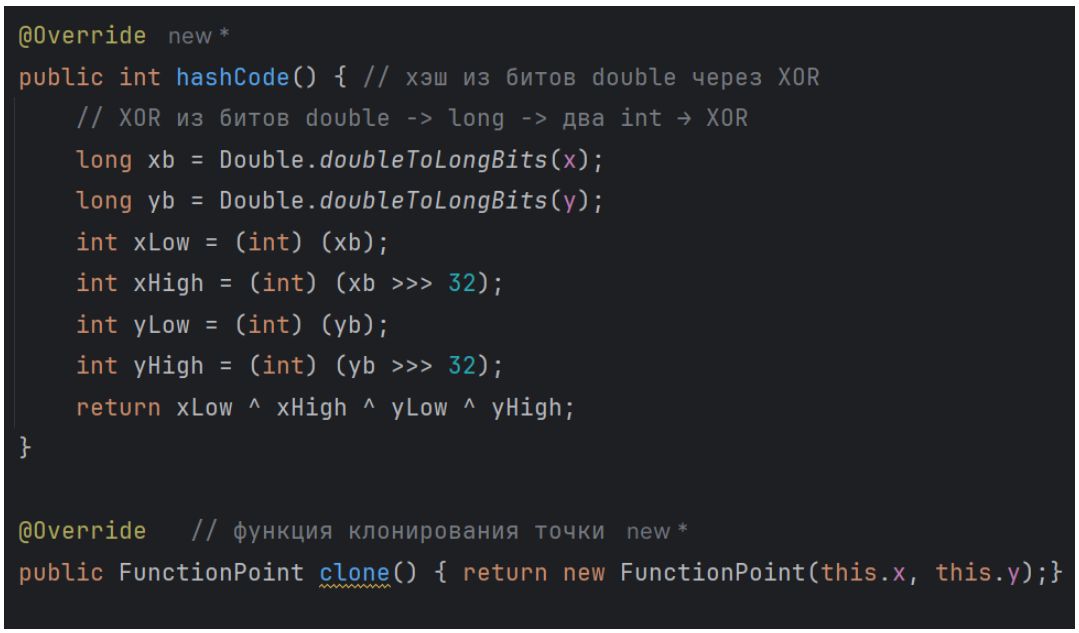
Для метода hashCode() было реализовано точное битовое представление double: для каждой координаты беру Double.doubleToLongBits, делю на старшие/младшие 32 бита и использую XOR.

Для метода clone() я сделал возможность создания точных копий точек. Создаем новый объект с теми же координатами.

```
@Override  new *
public String toString() { // человекочитаемый вывод
    // Требуемый формат: (x; y)
    return "(" + x + "; " + y + ")";
}

@Override  new *
public boolean equals(Object o) { // сравниваем точки по координатно
    if (this == o) return true;
    if (o == null || o.getClass() != FunctionPoint.class) return false;
    FunctionPoint that = (FunctionPoint) o;
    // сравнение чисел типа double
    return Double.compare(this.x, that.x) == 0 &&
            Double.compare(this.y, that.y) == 0;
}
```

Рисунок 1 – переопределение методов toString(), equals(Object o)



```
@Override  new *
public int hashCode() { // хэш из битов double через XOR
    // XOR из битов double -> long -> два int → XOR
    long xb = Double.doubleToLongBits(x);
    long yb = Double.doubleToLongBits(y);
    int xLow = (int) (xb);
    int xHigh = (int) (xb >>> 32);
    int yLow = (int) (yb);
    int yHigh = (int) (yb >>> 32);
    return xLow ^ xHigh ^ yLow ^ yHigh;
}

@Override   // функция клонирования точки  new *
public FunctionPoint clone() { return new FunctionPoint(this.x, this.y);}
```

Рисунок 2 – переопределение методов hashCode() и clone().

Задание 2. Переопределение методов в ArrayTabulatedFunction

Для выполнения 2-го задания, в классе ArrayTabulatedFunction я переопределил методы toString(), equals(Object o), hashCode() и clone().

Для метода toString**()** я сформировал читаемое представление всей функции в виде последовательности точек: {(x0; y0), (x1; y1), ...}. Точки выводятся в порядке возрастания координаты x, что соответствует их фактическому хранению в массиве.

Для метода equals() я реализовал два способа сравнения. Быстрый способ используется, когда сравниваем две функции на массивах, сначала проверяем количество точек, затем сравниваем каждую точку в массиве по координатам. Универсальный способ работает с любой реализацией TabulatedFunction, используем стандартные методы интерфейса для поэлементного сравнения.

Для метода hashCode() я создал хеш-код на основе всех точек функции, добавив информацию о количестве точек. Это позволяет различать функции с одинаковыми точками, но разной длиной.

Для метода clone() я реализовал полное независимое копирование, создается новый массив и новые объекты точек. Это гарантирует, что после копирования изменения в оригинально массиве не влияют на копию.

```java
@Override  new *
public String toString() { // человекочитаемый вывод в формате из задания
    StringBuilder sb = new StringBuilder();
    sb.append('{');
    for (int i = 0; i < size; ++i) {
        if (i > 0) sb.append(", ");
        FunctionPoint p = points[i];
        sb.append('(').append(p.getX()).append("; ").append(p.getY()).append(')');
    }
    sb.append('}');
    return sb.toString();
}
```

Рисунок 3 – Реализация метода toString() в ArrayTabulatedFunction

```java
@Override  new *
public boolean equals(Object o) { // Равенство, если длина и все (x,y) совпадают
    if (this == o) return true;
    // Быстрый путь
    if (o != null && o.getClass() == ArrayTabulatedFunction.class) {
        ArrayTabulatedFunction that = (ArrayTabulatedFunction) o;
        if (this.size != that.size) return false;
        for (int i = 0; i < this.size; ++i) {
            FunctionPoint a = this.points[i];
            FunctionPoint b = that.points[i];
            if (Double.compare(a.getX(), b.getX()) != 0) return false; // строгое сравнение double
            if (Double.compare(a.getY(), b.getY()) != 0) return false;
        }
        return true;
    }
    // Универсальный путь - любой TabulatedFunction сверяем через интерфейс
    if (o instanceof TabulatedFunction) {
        TabulatedFunction tf = (TabulatedFunction) o;
        if (this.size != tf.getPointsCount()) return false;
        for (int i = 0; i < this.size; ++i) {
            FunctionPoint a = this.points[i];
            FunctionPoint b = tf.getPoint(i);
```

```
                if (Double.compare(a.getX(), b.getX()) != 0) return false; // строгое сравнение double
                if (Double.compare(a.getY(), b.getY()) != 0) return false;
            }
            return true;
        }
        // Универсальный путь - любой TabulatedFunction сверяем через интерфейс
        if (o instanceof TabulatedFunction) {
            TabulatedFunction tf = (TabulatedFunction) o;
            if (this.size != tf.getPointsCount()) return false;
            for (int i = 0; i < this.size; ++i) {
                FunctionPoint a = this.points[i];
                FunctionPoint b = tf.getPoint(i);
                if (Double.compare(a.getX(), b.getX()) != 0) return false;
                if (Double.compare(a.getY(), b.getY()) != 0) return false;
            }
            return true;
        }
        return false;
    }
```

Рисунок 4,5 – реализация метода equals(Object o) в ArrayTabulatedFunction

```
@Override  new *
public int hashCode() { // XOR всех точек
    int h = size; // учитываем размер, чтобы {(-1,1),(0,0),(1,1)} != {(-1,1),(1,1)}
    for (int i = 0; i < size; ++i) {
        FunctionPoint p = points[i];
        long xb = Double.doubleToLongBits(p.getX()); // точные биты double
        long yb = Double.doubleToLongBits(p.getY());
        int xLow  = (int) xb;
        int xHigh = (int) (xb >>> 32);
        int yLow  = (int) yb;
        int yHigh = (int) (yb >>> 32);
        int ph = xLow ^ xHigh ^ yLow ^ yHigh;  // хеш одной точки
        h ^= (ph << 1) | (ph >>> 31);          // ротация влево на 1 бит, а потом применяется XOR
    }
    return h;
}
```

Рисунок 6 – реализация метода hashCode() в ArrayTabulatedFunction

```java
@Override new *
public ArrayTabulatedFunction clone() { // клонирование - новый массив и новые точки
    FunctionPoint[] copy = new FunctionPoint[this.size];
    for (int i = 0; i < this.size; ++i) {
        FunctionPoint p = this.points[i];
        copy[i] = new FunctionPoint(p.getX(), p.getY()); // независимый объект точки
    }
    return new ArrayTabulatedFunction(copy); // используем конструктор из массива точек
}
```

Рисунок 6 – реализация метода clone() в ArrayTabulatedFunction

Задание 3: Переопределение методов в LinkedListTabulatedFunction

Для выполнения 3-го задания, в классе LinkedListTabulatedFunction я переопределил методы toString(), equals(Object o), hashCode() и clone().

Для метода toString() я, аналогично массиву, сформировал текстовое представление всех точек в формате {(x0; y0), (x1; y1), ...}, последовательно проходя по всем элементам структуры данных.

Для метода equals() я также реализовал два способа сравнения. Быстрый способ для двух функций на связных структурах синхронно проходит по обоим наборам данных и сравнивает соответствующие точки. Универсальный способ использует интерфейсные методы для поэлементного сравнения с любой реализацией TabulatedFunction.

Для метода hashCode() я вычислил хэш-код на основе всех точек, аналогично массиву, учитывая количество точек для обеспечения уникальности.

Для метода clone() вместо сложного копирования внутренней структуры, я создаю новый пустой объект и последовательно добавляю в него копии всех точек из исходного объекта.

```java
@Override  new *
public String toString() {
    // Формируем строковое представление функции в формате: {(x1; y1), (x2; y2), ...}
    StringBuilder sb = new StringBuilder();
    sb.append('{');
    FunctionNode cur = head.next; // Начинаем с первой реальной точки, после головы

    for (int i = 0; i < size; i++) {
        if (i > 0) sb.append(", "); // Разделитель между точками
        sb.append('(')
          .append(cur.point.getX())
          .append("; ")
          .append(cur.point.getY())
          .append(')');
        cur = cur.next; // Переходим к следующей точке
    }
    sb.append('}');
    return sb.toString();
}
```

Рисунок 7 – реализация метода toString() в LinkedListTabulatedFunction

```java
@Override  new *
public boolean equals(Object o) {
    // Быстрая проверка: если это тот же объект, возвращаем true
    if (this == o) return true;

    // Оптимизированная проверка для объектов того же класса
    if (o != null && o.getClass() == LinkedListTabulatedFunction.class) {
        LinkedListTabulatedFunction that = (LinkedListTabulatedFunction) o;

        // Если разное количество точек делаем вывод, что функции не равны
        if (this.size != that.size) return false;

        // Последовательно сравниваем все точки
        FunctionNode thisNode = this.head.next;
        FunctionNode thatNode = that.head.next;

        for (int i = 0; i < size; i++) {
            // Точное сравнение координат для соблюдения equals
            if (Double.compare(thisNode.point.getX(), thatNode.point.getX()) != 0) return false;
            if (Double.compare(thisNode.point.getY(), thatNode.point.getY()) != 0) return false;
```

```
        thisNode = thisNode.next;
        thatNode = thatNode.next;}
    return true;
}
// Универсальная проверка для любых реализаций TabulatedFunction
if (o instanceof TabulatedFunction) {
    TabulatedFunction otherFunction = (TabulatedFunction) o;
    if (this.size != otherFunction.getPointsCount()) return false;
    // Сравниваем точки через интерфейсные методы
    FunctionNode thisNode = this.head.next;
    for (int i = 0; i < size; ++i) {
        FunctionPoint otherPoint = otherFunction.getPoint(i);
        if (Double.compare(thisNode.point.getX(), otherPoint.getX()) != 0) return false;
        if (Double.compare(thisNode.point.getY(), otherPoint.getY()) != 0) return false;
        thisNode = thisNode.next;}
    return true;
}
return false; // Объект другого типа
```

Рисунок 8,9 – реализация метода equals(Object o) в LinkedListTabulatedFunction

```
@Override  new *
public int hashCode() {     // Вычисляем хеш код на основе всех точек функции
    int hash = size; // Начинаем с размера
    FunctionNode current = head.next;
    for (int i = 0; i < size; ++i) {
        // Преобразуем double в long для получения точного битового представления
        long xBits = Double.doubleToLongBits(current.point.getX());
        long yBits = Double.doubleToLongBits(current.point.getY());
        // Разбиваем 64-битные значения на две 32-битные части
        int xLow = (int) xBits;
        int xHigh = (int) (xBits >>> 32);
        int yLow = (int) yBits;
        int yHigh = (int) (yBits >>> 32);
        // Комбинируем хеши координат точки
        int pointHash = xLow ^ xHigh ^ yLow ^ yHigh;
        // Вращаем биты и добавляем к общему хешу для лучшего распределения
        hash ^= (pointHash << 1) | (pointHash >>> 31);

        current = current.next;
    }
    return hash;
```

Рисунок 10 – реализация метода hashCode() в LinkedListTabulatedFunction

```java
@Override  new *
public LinkedListTabulatedFunction clone() {
    // Создаем новый независимый объект
    LinkedListTabulatedFunction clonedFunction = new LinkedListTabulatedFunction();

    // Копируем все точки, создавая новые объекты FunctionPoint
    FunctionNode current = this.head.next;
    for (int i = 0; i < this.size; i++) {
        // Создаем новую точку с теми же координатами
        FunctionPoint newPoint = new FunctionPoint(current.point.getX(), current.point.getY());
        clonedFunction.addNodeToTail(newPoint);
        current = current.next;
    }
    return clonedFunction;
}
```

Рисунок 11 – реализация метода clone() в LinkedListTabulatedFunction

Задание 4. Клонирование на уровне интерфейса TabulatedFunction

Интерфейс TabulatedFunction теперь расширяет Cloneable и **имеет метод** clone(). Значит, любая реализация обязана его предоставить, а мы можем одинаково копировать любые функции, не разбираясь в их внутренних деталях.

```java
package functions;


public interface TabulatedFunction extends Function, Cloneable { // Все реализации клонируемые
    TabulatedFunction clone(); // Создает и возвращает копию табулированной функции  2 implementa
```

Рисунок 12 –добавляем поддержку клонирования в TabulatedFunction

Задание 5. Проверка работы методов

Проверим работу методов в Main:
1) Печать строковых представлений (toString())
Создаём табулированные функции на массивах и на последовательностях из одинаковых точек и выводим их строковый вид. Это демонстрирует формат "{(x; y), ...}" и корректность переопределённого toString() в обеих реализациях.

```
-- toString() --
Array реализация: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}
List  реализация: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}
```

Рисунок 13 - Вывод toString() для Array/Linked реализаций

```
-- equals(): основные тестовые случаи --
arr1 == arr2 (один класс, одинаковые данные): true
list1 == list2 (один класс, одинаковые данные): true
arr1 == list1 (разные классы, одинаковые данные): true
list1 == arr1 (разные классы, одинаковые данные): true
arr1 == arr3 (один класс, разные данные): false
list1 == list3 (один класс, разные данные): false
arr1 == list3 (разные классы, разные данные): false
list1 == arr3 (разные классы, разные данные): false
arr1 == arrShort (разная длина): false
list1 == listShort (разная длина): false
arr1 == null: false
arr1 == "не функция": false
```

Рисунок 14 - Базовые сценарии equals()

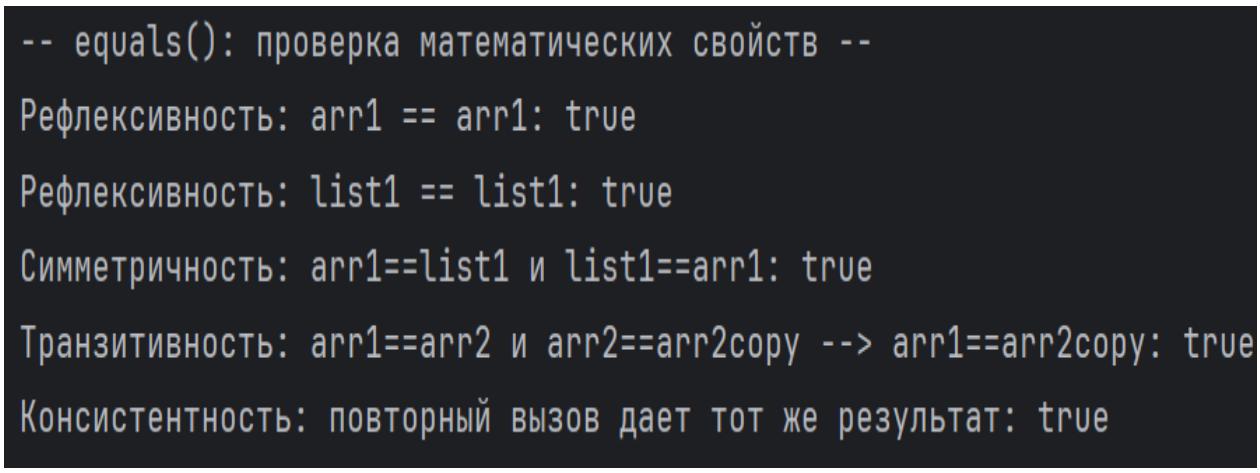


Рисунок 15 - Проверка рефлексивности/симметричности/транзитивности/консистентности

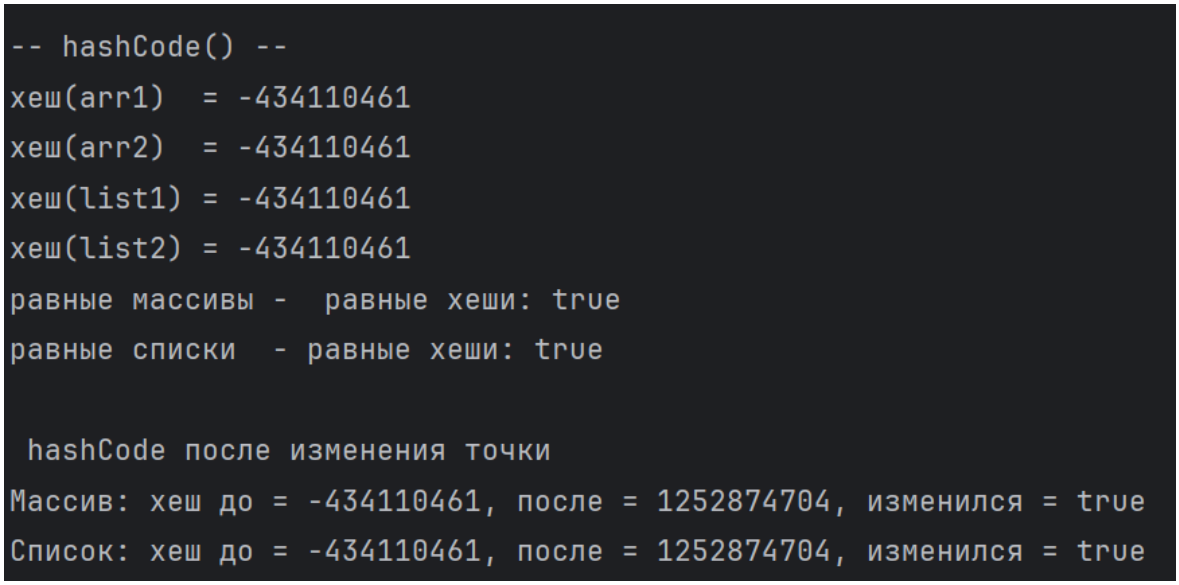


Рисунок 16,17 - Совпадение hashCode() у равных объектов, hashCode до/после изменения точки

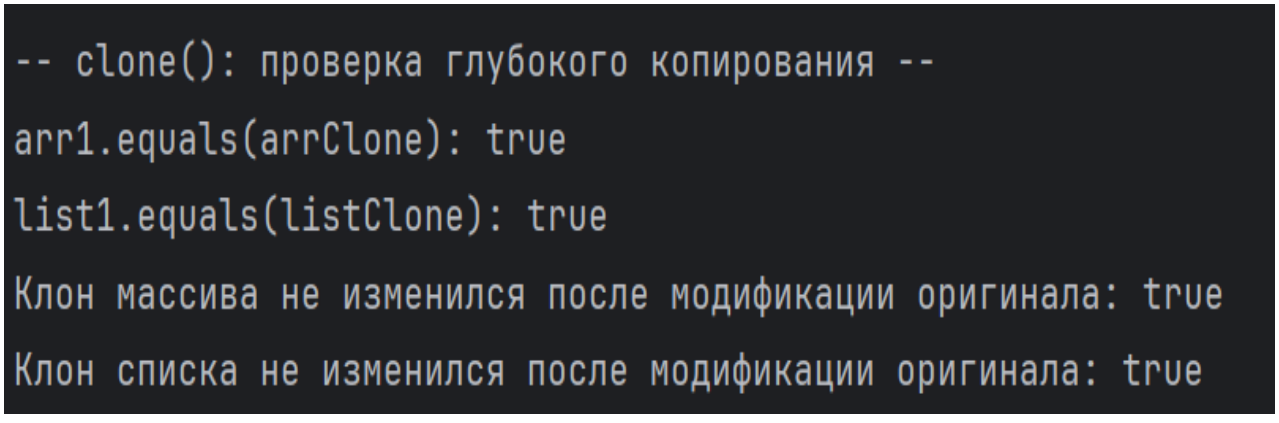


Рисунок 18 - Независимость клонов от оригиналов

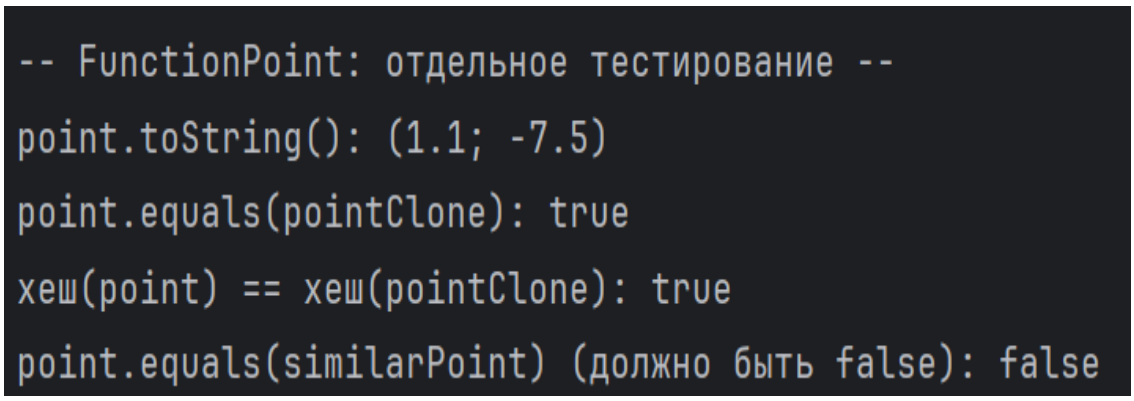


Рисунок 19 - Тесты FunctionPoint: toString/equals/hashCode/clone