

Лабораторная работа № 5

Титульный лист

Дисциплина: Объектно-ориентированное программирование

Выполнил: Мостовщиков Владимир Витальевич

Группа: 6204-010302D

Преподаватель: Борисов Дмитрий Сергеевич

Год: 2025

Содержание:

1. Задание 1. Переопределение методов в классе `FunctionPoint`
2. Задание 2. Переопределение методов в `ArrayTabulatedFunction`
3. Задание 3. Переопределение методов в `LinkedListTabulatedFunction`
4. Задание 4. Клонирование на уровне интерфейса `TabulatedFunction`
5. Задание 5. Проверка работы методов

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

@Override @Metilka
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 Math.abs(this.x - that.x) <= EPSILON
        && Math.abs(this.y - that.y) <= EPSILON;
}
```

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

```

@Override  @Metilka *
public int hashCode() { // хэш из битов double через XOR
    // округляем до ближайшего кратного EPSILON
    double qx = Math rint(x / EPSILON) * EPSILON;
    double qy = Math rint(y / EPSILON) * EPSILON;
    // Нормализуем нули, чтобы +0.0 и -0.0 давали одинаковый хеш
    if (qx == 0.0) qx = 0.0;
    if (qy == 0.0) qy = 0.0;
    // Преобразуем double в long для получения точного битового представления
    long xb = Double.doubleToLongBits(qx);
    long yb = Double.doubleToLongBits(qy);
    // Разбиваем 64-битные значения на две 32-битные части
    int xLow = (int) (xb);
    int xHigh = (int) (xb >>> 32);
    int yLow = (int) (yb);
    int yHigh = (int) (yb >>> 32);
    int h = 31;
    h ^= xLow ^ xHigh;
    h = Integer.rotateLeft(h, distance: 5);
    h ^= yLow ^ yHigh;
    return h;
}

```

```

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

```

Рисунок 2,3 – 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 оригинальном массиве не влияют на копию.

```
@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 4 — Реализация метода toString() в ArrayTabulatedFunction

```

@Override  @Metilka
public boolean equals(Object o) { // Равенство, если длина и все (x,y) совпадают
    if (this == o) return true;
    // Быстрый путь
    if (o instanceof ArrayTabulatedFunction other) {
        // Если разное количество точек - функции не равны
        if (this.size != other.size) return false;
        for (int i = 0; i < this.size; ++i) {
            // Используем встроенный метод equals точек, который учитывает погрешность EPSILON
            if (!this.points[i].equals(other.points[i])) return false;
        }
        return true;
    }
    // Универсальный путь: сравнение с любой реализацией TabulatedFunction.
    if (o instanceof TabulatedFunction tf) {
        if (this.size != tf.getPointsCount()) return false; // Проверяем совпадение количества точек
        // Сравниваем точки через интерфейсные методы
        for (int i = 0; i < this.size; ++i) {
            FunctionPoint otherPoint = new FunctionPoint(tf.getPointX(i), tf.getPointY(i));
            if (!this.points[i].equals(otherPoint)) return false;
        }
    }
}

```

```

// Универсальный путь: сравнение с любой реализацией TabulatedFunction.
if (o instanceof TabulatedFunction tf) {
    if (this.size != tf.getPointsCount()) return false; // Проверяем совпадение количества точек
    // Сравниваем точки через интерфейсные методы
    for (int i = 0; i < this.size; ++i) {
        FunctionPoint otherPoint = new FunctionPoint(tf.getPointX(i), tf.getPointY(i));
        if (!this.points[i].equals(otherPoint)) return false;
    }
    return true;
}
return false;
}

```

Рисунок 5,6 – реализация метода equals(Object o) в ArrayTabulatedFunction

```

@Override  @Metilka
public int hashCode() {
    int h = size; // учитываем размер, чтобы {(-1,1),(0,0),(1,1)} != {(-1,1),(1,1)}
    for (int i = 0; i < size; ++i) {
        int ph = points[i].hashCode(); // берём хеш точки
        h ^= Integer.rotateLeft(ph, (i & 15));
    }
    return h;
}

```

Рисунок 7 – реализация метода hashCode() в ArrayTabulatedFunction

```

@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 8 – 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 связанных 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.

```

@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 9 – реализация метода toString() в LinkedListTabulatedFunction

```

@Override @ Metilka
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 instanceof LinkedListTabulatedFunction other) {
        int n = this.getPointsCount();
        if (n != other.getPointsCount()) return false;
        for (int i = 0; i < n; ++i) {
            if (!this.getPoint(i).equals(other.getPoint(i))) return false;
        }
        return true;
    }

    // Универсальный путь: с любым TabulatedFunction.
    if (o instanceof TabulatedFunction tf) {
        int n = this.getPointsCount();
        if (n != tf.getPointsCount()) return false;
        for (int i = 0; i < n; ++i) {
            FunctionPoint otherPoint = new FunctionPoint(tf.getPointX(i), tf.getPointY(i));
            if (!this.getPoint(i).equals(otherPoint)) return false;
        }
    }
}

```

```

// Оптимизированная проверка для объектов того же класса
if (o instanceof LinkedListTabulatedFunction other) {
    int n = this.getPointsCount();
    if (n != other.getPointsCount()) return false;
    for (int i = 0; i < n; ++i) {
        if (!this.getPoint(i).equals(other.getPoint(i))) return false;
    }
    return true;
}

// Универсальный путь: с любым TabulatedFunction.
if (o instanceof TabulatedFunction tf) {
    int n = this.getPointsCount();
    if (n != tf.getPointsCount()) return false;
    for (int i = 0; i < n; ++i) {
        FunctionPoint otherPoint = new FunctionPoint(tf.getPointX(i), tf.getPointY(i));
        if (!this.getPoint(i).equals(otherPoint)) return false;
    }
    return true;
}
return false;
}
}

```

Рисунок 10,11 – реализация метода equals(Object o) в LinkedListTabulatedFunction

```

@Override
public int hashCode() {
    int h = size; // учитываем размер, чтобы различать функции разной длины
    FunctionNode cur = head.next;
    for (int i = 0; i < size; ++i) {
        int ph = cur.point.hashCode();
        h ^= Integer.rotateLeft(ph, (i & 15));
        cur = cur.next;
    }
    return h;
}

```

Рисунок 12 – реализация метода hashCode() в LinkedListTabulatedFunction


```

@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 13 – 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.

```

package functions;

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

```

Рисунок 14 –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 15 - 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 16 - Базовые сценарии equals()

```
-- equals(): проверка математических свойств --
Рефлексивность: arr1 == arr1: true
Рефлексивность: list1 == list1: true
Симметричность: arr1==list1 и list1==arr1: true
Транзитивность: arr1==arr2 и arr2==arr2copy --> arr1==arr2copy: true
Консистентность: повторный вызов дает тот же результат: true
```

Рисунок 17 - Проверка
рефлексивности/симметричности/транзитивности/консистентности

```
-- hashCode() --
хеш(arr1) = -434110461
хеш(arr2) = -434110461
хеш(list1) = -434110461
хеш(list2) = -434110461
равные массивы - равные хеши: true
равные списки - равные хеши: true

hashCode после изменения точки
Массив: хеш до = -434110461, после = 1252874704, изменился = true
Список: хеш до = -434110461, после = 1252874704, изменился = true
```

Рисунок 17,18 - Совпадение hashCode() у равных объектов, hashCode
до/после изменения точки

```
-- clone(): проверка глубокого копирования --
arr1.equals(arrClone): true
list1.equals(listClone): true
Клон массива не изменился после модификации оригинала: true
Клон списка не изменился после модификации оригинала: true
```

Рисунок 19 - Независимость клонов от оригиналов

```
-- FunctionPoint: отдельное тестирование --  
point.toString(): (1.1; -7.5)  
point.equals(pointClone): true  
хеш(point) == хеш(pointClone): true  
point.equals(similarPoint) (должно быть false): false
```

Рисунок 20 - Тесты FunctionPoint: toString/equals/hashCode/clone