

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени
академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»
ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ
«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА №1

Студент Синцов М.Ю
Группа 6201-120303D

Руководитель Борисов Д. С.

САМАРА 2025

Задание 1 Переопределение методов в FunctionPoint

Цель: Переопределить методы `toString()`, `equals()`, `hashCode()` и `clone()` в классе `FunctionPoint`.

Выполнение:

1. `toString()`: Возвращает строку в формате `[x,y]` с одним знаком после запятой
 2. `equals()`: Сравнивает точки с учетом погрешности
 3. `hashCode()`: Использует `Double.doubleToLongBits()` для преобразования координат в целочисленные значения
 4. `clone()`: Реализует поверхностное клонирование через `super.clone()`. Ключевой момент: Класс был дополнен реализацией интерфейса `Cloneable` для поддержки метода `clone()`.
- Ключевой момент: Класс был дополнен реализацией интерфейса `Cloneable` для поддержки метода `clone()`.

```
@Override
public String toString() {
    return ("[" + x + "," + y + "]");
}
@Override
public boolean equals(Object o) {
    if (o == null || getClass() != o.getClass())
        return false;

    FunctionPoint that = (FunctionPoint) o;

    return Double.compare(this.x, that.x) == 0 &&
           Double.compare(this.y, that.y) == 0;
}
@Override
public int hashCode() {
    long xBits = Double.doubleToLongBits(x);
    long yBits = Double.doubleToLongBits(y);

    // XOR для старших и младших частей обоих double
    int hash = (int) (xBits ^ (xBits >>> 32));
    hash = 31 * hash + (int) (yBits ^ (yBits >>> 32));

    return hash;
}
@Override
public Object clone() {
    try {
        return super.clone();
    } catch (CloneNotSupportedException e) {
        throw new InternalError(e);
    }
}
```

Задание 2

Цель: Переопределить методы `toString()`, `equals()`, `hashCode()` и `clone()` в классе `ArrayTabulatedFunction`.

Выполнение:

1. `toString()`: Возвращает строку в формате {[x₁,y₁], [x₂,y₂], ...}
2. `equals()`:
 - о Проверяет количество точек
 - о Сравнивает координаты с учетом погрешности
 - о Оптимизирован для сравнения с объектами того же класса
3. `hashCode()`: Вычисляется как XOR хэш-кодов всех точек с учетом количества точек
4. `clone()`: Реализует глубокое клонирование массива точек

Ключевой момент: Метод `clone()` создает новый массив точек, клонируя каждую точку отдельно.

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append('{');
    for(int i = 0; i < pointsCount; i++) {
        sb.append(points[i]);
        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;
    TabulatedFunction that = (TabulatedFunction) o;

    if (this.pointsCount != that.getPointsCount()) return false;
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction arrayThat = (ArrayTabulatedFunction) o;

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

        for (int i = 0; i < pointsCount; i++) {
            double thisX = this.getPointX(i);
            double thisY = this.getPointY(i);
            double thatX = that.getPointX(i);
            double thatY = that.getPointY(i);

            if (Math.abs(thisX - thatX) >= EPSILON ||
                Math.abs(thisY - thatY) >= EPSILON) {
                return false;
            }
        }
    }
}
```

```

        }
    }
    return true;
}
@Override
public int hashCode() {
    int hash = pointsCount;

    for (int i = 0; i < pointsCount; i++) {
        hash = 31 * hash + points[i].hashCode();
    }

    return hash;
}
@Override
public Object clone() {
    try {
        ArrayTabulatedFunction clone = (ArrayTabulatedFunction)
super.clone();

        clone.points = new FunctionPoint[this.points.length];
        for (int i = 0; i < this.pointsCount; i++) {
            clone.points[i] = (FunctionPoint) this.points[i].clone();
        }

        clone.pointsCount = this.pointsCount;

        return clone;
    } catch (CloneNotSupportedException e) {
        throw new InternalError(e);
    }
}
}

```

Задание 3

Цель: Переопределить те же методы для связного списка с учетом особенностей структуры данных.

Выполнение:

1. `toString()`: Обход списка и вывод точек в том же формате
2. `equals()`: Проверяет количество точек о Для сравнения с `LinkedListTabulatedFunction` использует прямой обход списка о Для других реализаций использует интерфейсные методы
3. `hashCode()`: Обход списка для вычисления XOR хэш-кодов
4. `clone()`: "Пересборка" списка путем создания новых узлов

Ключевой момент: Метод `clone()` не использует методы добавления точек для избежания лишних операций.

```

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    FunctionNode node = head.next;
    sb.append('{');
    for(int i = 0; i < pointsCount; i++) {
        sb.append(node.point);
        node = node.next;
        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;
    TabulatedFunction that = (TabulatedFunction) o;

    if (this.pointsCount != that.getPointsCount()) return false;
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction arrayThat = (LinkedListTabulatedFunction)
o;

        FunctionNode node = this.head.next;
        FunctionNode Thatnode = arrayThat.head.next;
        for (int i = 0; i < pointsCount; i++) {
            if (!Thatnode.point.equals(node.point)) {
                return false;
            }
            node = node.next;
            Thatnode = Thatnode.next;
        }
    } else {

        for (int i = 0; i < pointsCount; i++) {
            double thisX = this.getPointX(i);
            double thisY = this.getPointY(i);
            double thatX = that.getPointX(i);
            double thatY = that.getPointY(i);

            if (Math.abs(thisX - thatX) >= EPSILON ||
                Math.abs(thisY - thatY) >= EPSILON) {
                return false;
            }
        }
    }
    return true;
}

@Override
public int hashCode() {
    int hash = pointsCount;

    FunctionNode node = this.head.next;
    for (int i = 0; i < pointsCount; i++) {
        hash = 31 * hash + node.point.hashCode();
        node = node.next;
    }

    return hash;
}

@Override
public Object clone() {
    try {

```

```

        LinkedListTabulatedFunction clone = (LinkedListTabulatedFunction)
super.clone();
        clone.initializeList();
        if (this.pointsCount > 0) {

            FunctionNode currentNode = this.head.next;
            FunctionNode lastCreatedNode = null;

            while (currentNode != this.head) {
                FunctionNode newNode = new FunctionNode((FunctionPoint)
currentNode.point.clone());
                if (lastCreatedNode == null) {
                    clone.head.next = newNode;
                    newNode.prev = clone.head;
                } else {
                    lastCreatedNode.next = newNode;
                    newNode.prev = lastCreatedNode;
                }

                lastCreatedNode = newNode;
                clone.pointsCount++;
                currentNode = currentNode.next;
            }

            if (lastCreatedNode != null) {
                lastCreatedNode.next = clone.head;
                clone.head.prev = lastCreatedNode;
            }
        }

        return clone;
    } catch (CloneNotSupportedException e) {
        throw new InternalError(e);
    }
}

```

Задание 4

Цель: Сделать все объекты TabulatedFunction клонируемыми.

Выполнение:

1. Интерфейс TabulatedFunction расширен от Cloneable
2. В интерфейс добавлен метод clone()
3. Все реализации автоматически становятся клонируемыми

```

package functions;
import java.io.*;
public interface TabulatedFunction extends Function, Serializable, Cloneable {
    public int getPointsCount();
    public void showPoints();
    public FunctionPoint getPoint(int index);
    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException;
    public double getPointX(int index);
    public void setPointX(int index, double x) throws

```

```

InappropriateFunctionPointException;
    public double getPointY(int index);
    public void setPointY(int index, double y);
    public void deletePoint(int index);
    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException;
    public Object clone();
}

```

Задание 5

```

import functions.*;
import functions.basic.*;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;

import static java.nio.file.Files.newBufferedWriter;

public class Main {

    public static void main(String[] args) {
        System.out.println("==== ЛАБОРАТОРНАЯ РАБОТА №5: ПЕРЕОПРЕДЕЛЕНИЕ
МЕТОДОВ Object ===\n");

        try {
            // Часть 1: Тестирование FunctionPoint
            System.out.println("1. ТЕСТИРОВАНИЕ FunctionPoint\n");
            testFunctionPoint();

            // Часть 2: Тестирование ArrayTabulatedFunction
            System.out.println("\n\n2. ТЕСТИРОВАНИЕ
ArrayTabulatedFunction\n");
            testArrayTabulatedFunction();

            // Часть 3: Тестирование LinkedListTabulatedFunction
            System.out.println("\n\n3. ТЕСТИРОВАНИЕ
LinkedListTabulatedFunction\n");
            testLinkedListTabulatedFunction();

            // Часть 4: Сравнение разных реализаций
            System.out.println("\n\n4. СРАВНЕНИЕ РЕАЛИЗАЦИЙ\n");
            compareImplementations();

            System.out.println("\n==== ВСЕ ТЕСТЫ УСПЕШНО ЗАВЕРШЕНЫ ===\n");
        } catch (Exception e) {
            System.out.println("Ошибка: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static void testFunctionPoint() {
        System.out.println("  1.1 Метод toString():");
        FunctionPoint p1 = new FunctionPoint(1.5, 2.7);
        FunctionPoint p2 = new FunctionPoint(0.0, -3.1415);
        System.out.println("      p1 = " + p1);
        System.out.println("      p2 = " + p2);

        System.out.println("\n  1.2 Метод equals():");
        FunctionPoint p3 = new FunctionPoint(1.5, 2.7);
        FunctionPoint p4 = new FunctionPoint(1.50000001, 2.70000001);
    }
}

```

```

        FunctionPoint p5 = new FunctionPoint(1.5, 3.0);

        System.out.println("p1.equals(p1) = " + p1.equals(p1)); // true
        System.out.println("p1.equals(p3) = " + p1.equals(p3)); // true
        System.out.println("p1.equals(p4) = " + p1.equals(p4)); // true

        System.out.println("p1.equals(p5) = " + p1.equals(p5)); //
        System.out.println("p1.equals(null) = " + p1.equals(null)); //
        System.out.println("\\" строка \") = " +
p1.equals("строка")); // false

        System.out.println("\n1.3 Метод hashCode():");
        p1.hashCode() = " + p1.hashCode();
        p3.hashCode() = " + p3.hashCode();
        p5.hashCode() = " + p5.hashCode();
        p1.hashCode() == p3.hashCode() ? " +
(p1.hashCode() == p3.hashCode()));

        System.out.println("\n1.4 Метод clone():");
        FunctionPoint p6 = (FunctionPoint) p1.clone();
        System.out.println("Оригинал: " + p1);
        System.out.println("Клон: " + p6);
        System.out.println("p1.equals(p6) ? " + p1.equals(p6));
        p1 == p6? " + (p1 == p6));

        // Изменяем оригинал
        p1.setX(10.0);
        System.out.println("После изменения оригинала:");
        System.out.println("Оригинал: " + p1);
        System.out.println("Клон: " + p6);
        System.out.println("p1.equals(p6) ? " + p1.equals(p6));
    }

    private static void testArrayTabulatedFunction() throws
InappropriateFunctionPointException {
    // Создаем функцию с 5 точками
    double[] values = {0.0, 1.0, 4.0, 9.0, 16.0}; // y = x2
    ArrayTabulatedFunction func1 = new ArrayTabulatedFunction(0, 4,
values);

    System.out.println("2.1 Метод toString():");
    System.out.println("func1 = " + func1);

    System.out.println("\n2.2 Метод equals():");

    // Создаем идентичную функцию
    ArrayTabulatedFunction func2 = new ArrayTabulatedFunction(0, 4,
values);

    // Создаем похожую, но не идентичную функцию
    double[] values3 = {0.0, 1.0, 4.0, 9.0, 16.1}; // Последняя точка
отличается
    ArrayTabulatedFunction func3 = new ArrayTabulatedFunction(0, 4,
values3);

    // Создаем функцию с другим количеством точек
    double[] values4 = {0.0, 4.0, 16.0}; // Только 3 точки
    ArrayTabulatedFunction func4 = new ArrayTabulatedFunction(0, 4,
values4);

    System.out.println("func1.equals(func1) = " +
func1.equals(func1)); // true
}

```

```

        System.out.println("func1.equals(func2)); // true
        System.out.println("func1.equals(func3)); // false
        System.out.println("func1.equals(func4)); // false
        System.out.println("func1.equals(null)); // false
        System.out.println("func1.equals("строка")); // false

        System.out.println("\n 2.3 Метод hashCode():");
        func1.hashCode() = " + func1.hashCode();
        func2.hashCode() = " + func2.hashCode();
        func3.hashCode() = " + func3.hashCode();
        func4.hashCode() = " + func4.hashCode();
        func1.hashCode() == func2.hashCode() ? " +
        func1.hashCode() == func2.hashCode());
        func1.hashCode() == func3.hashCode() ? " +
        func1.hashCode() == func3.hashCode()));

System.out.println("\n 2.4 Метод clone():");
ArrayTabulatedFunction clone1 = (ArrayTabulatedFunction)
func1.clone();
        System.out.println("Оригинал: " + func1);
        System.out.println("Клон: " + clone1);
        System.out.println("func1.equals(clone1)? " +
func1.equals(clone1));
        System.out.println("func1 == clone1? " + (func1 == clone1));

// Проверяем глубокое клонирование
System.out.println("\n      Проверка глубокого клонирования:");
System.out.println("      Изменяем точку в оригинале...");
func1.setPointY(2, 100.0); // Меняем третью точку

        System.out.println("      Оригинал после изменения: " + func1);
        System.out.println("      Клон после изменения оригинала: " +
clone1);
        System.out.println("      Точка [2] в оригинале: " +
func1.getPoint(2));
        System.out.println("      Точка [2] в клоне: " + clone1.getPoint(2));
        System.out.println("      Они равны? " +
func1.getPoint(2).equals(clone1.getPoint(2)));
        System.out.println("      Это один и тот же объект? " +
(func1.getPoint(2) == clone1.getPoint(2)));
    }

private static void testLinkedListTabulatedFunction() throws
InappropriateFunctionPointException {
    // Создаем функцию с 5 точками
    double[] values = {0.0, 1.0, 4.0, 9.0, 16.0}; // y = x2
    LinkedListTabulatedFunction func1 = new
LinkedListTabulatedFunction(0, 4, values);

    System.out.println(" 3.1 Метод toString():");
    System.out.println("      func1 = " + func1);

    System.out.println("\n 3.2 Метод equals():");

    // Создаем идентичную функцию
    LinkedListTabulatedFunction func2 = new
LinkedListTabulatedFunction(0, 4, values);

    // Создаем похожую, но не идентичную функцию
}

```

```

        double[] values3 = {0.0, 1.0, 4.0, 9.0, 16.1}; // Последняя точка
отличается
        LinkedListTabulatedFunction func3 = new
LinkedListTabulatedFunction(0, 4, values3);

        System.out.println("      func1.equals(func1) = " +
func1.equals(func1)); // true
        System.out.println("      func1.equals(func2) = " +
func1.equals(func2)); // true
        System.out.println("      func1.equals(func3) = " +
func1.equals(func3)); // false

        System.out.println("\n      3.3 Метод hashCode():");
        System.out.println("      func1.hashCode() = " + func1.hashCode());
        System.out.println("      func2.hashCode() = " + func2.hashCode());
        System.out.println("      func3.hashCode() = " + func3.hashCode());
        System.out.println("      func1.hashCode() == func2.hashCode() ? " +
(func1.hashCode() == func2.hashCode()));
        System.out.println("      func1.hashCode() == func3.hashCode() ? " +
(func1.hashCode() == func3.hashCode()));

        System.out.println("\n      3.4 Метод clone():");
        LinkedListTabulatedFunction clone1 = (LinkedListTabulatedFunction)
func1.clone();
        System.out.println("      Оригинал: " + func1);
        System.out.println("      Клон: " + clone1);
        System.out.println("      func1.equals(clone1)? " +
func1.equals(clone1));
        System.out.println("      func1 == clone1? " + (func1 == clone1));

        // Проверяем глубокое клонирование
        System.out.println("\n      Проверка глубокого клонирования:");
        System.out.println("      Изменяем точку в оригинале...");
        func1.setPointY(2, 100.0); // Меняем третью точку

        System.out.println("      Оригинал после изменения: " + func1);
        System.out.println("      Клон после изменения оригинала: " +
clone1);
        System.out.println("      Точка [2] в оригинале: " +
func1.getPoint(2));
        System.out.println("      Точка [2] в клоне: " + clone1.getPoint(2));
        System.out.println("      Они равны? " +
func1.getPoint(2).equals(clone1.getPoint(2)));
        System.out.println("      Это один и тот же объект? " +
(func1.getPoint(2) == clone1.getPoint(2)));

        // Проверяем структуру списка после клонирования
        System.out.println("\n      Проверка структуры списка:");
        System.out.println("      Удаляем точку из оригинала...");
        func1.deletePoint(1); // Удаляем вторую точку

        System.out.println("      Оригинал после удаления: " + func1);
        System.out.println("      Клон после удаления в оригинале: " +
clone1);
        System.out.println("      Количество точек в оригинале: " +
func1.getPointsCount());
        System.out.println("      Количество точек в клоне: " +
clone1.getPointsCount());
    }

    private static void compareImplementations() throws
InappropriateFunctionPointException {
    // Создаем одинаковые функции разными реализациями
    double[] values = {0.0, 1.0, 4.0, 9.0, 16.0};
}

```

```

        ArrayTabulatedFunction arrayFunc = new ArrayTabulatedFunction(0, 4,
values);
        LinkedListTabulatedFunction listFunc = new
LinkedListTabulatedFunction(0, 4, values);

        System.out.println("  4.1 Сравнение toString():");
        System.out.println("    Array: " + arrayFunc);
        System.out.println("    LinkedList: " + listFunc);

        System.out.println("\n  4.2 Сравнение equals():");
        System.out.println("    arrayFunc.equals(listFunc) = " +
arrayFunc.equals(listFunc));
        System.out.println("    listFunc.equals(arrayFunc) = " +
listFunc.equals(arrayFunc));

        System.out.println("\n  4.3 Сравнение hashCode():");
        System.out.println("    arrayFunc.hashCode() = " +
arrayFunc.hashCode());
        System.out.println("    listFunc.hashCode() = " +
listFunc.hashCode());
        System.out.println("    Хэш-коды равны? " + (arrayFunc.hashCode() ==
listFunc.hashCode()));

        System.out.println("\n  4.4 Тест на небольшое изменение:");
        System.out.println("    Исходный хэш arrayFunc: " +
arrayFunc.hashCode());

        // Незначительно меняем точку
        arrayFunc.setPointY(2, 4.001); // Меняем с 4.0 на 4.001
        System.out.println("    Хэш после изменения y[2] на 4.001: " +
arrayFunc.hashCode());

        // Возвращаем обратно
        arrayFunc.setPointY(2, 4.0);
        System.out.println("    Хэш после возврата y[2] к 4.0: " +
arrayFunc.hashCode());

        // Меняем другую точку
        arrayFunc.setPointY(3, 9.005); // Меняем с 9.0 на 9.005
        System.out.println("    Хэш после изменения y[3] на 9.005: " +
arrayFunc.hashCode());

        // Проверяем equals после изменений
        System.out.println("\n  4.5 Проверка согласованности equals() и
hashCode():");
        System.out.println("    arrayFunc.equals(listFunc) ? " +
arrayFunc.equals(listFunc));
        System.out.println("    Хэш-коды равны? " + (arrayFunc.hashCode() ==
listFunc.hashCode()));

        // Возвращаем обратно для чистоты теста
        arrayFunc.setPointY(3, 9.0);
    }

    // Дополнительный тест для проверки версионности
    private static void testVersioning () {
        System.out.println("\n\nДополнительно: проверка версионности");

        try {
            // Создаем функцию
            TabulatedFunction func = TabulatedFunctions.tabulate(new
Sin(), 0, Math.PI, 5);

```

```

    // Сохраняем
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    try (ObjectOutputStream out = new ObjectOutputStream(baos)) {
        out.writeObject(func);
    }

    byte[] data = baos.toByteArray();
    System.out.println("Размер сериализованных данных: " +
data.length + " байт");

    // Восстанавливаем
    ByteArrayInputStream bais = new ByteArrayInputStream(data);
    try (ObjectInputStream in = new ObjectInputStream(bais)) {
        TabulatedFunction restored = (TabulatedFunction)
in.readObject();
        System.out.println("Функция успешно восстановлена");
        System.out.println("Класс: " +
restored.getClass().getName());
    }

} catch (Exception e) {
    System.out.println("Ошибка версионности: " + e.getMessage());
}
}
}
}

```

Результат выполнения.

1. ТЕСТИРОВАНИЕ FunctionPoint

1.1 Метод `toString()`:

`p1 = [1.5,2.7]`

`p2 = [0.0,-3.1415]`

1.2 Метод `equals()`:

`p1.equals(p1) = true`

`p1.equals(p3) = true`

`p1.equals(p4) = false`

`p1.equals(p5) = false`

`p1.equals(null) = false`

`p1.equals("строка") = false`

1.3 Метод `hashCode()`:

`p1.hashCode() = -1734082557`

`p3.hashCode() = -1734082557`

```
p5.hashCode() = -15728640  
p1.hashCode() == p3.hashCode()? true
```

1.4 Метод clone():

Оригинал: [1.5,2.7]

Клон: [1.5,2.7]

p1.equals(p6)? true

p1 == p6? false

После изменения оригинала:

Оригинал: [10.0,2.7]

Клон: [1.5,2.7]

p1.equals(p6)? false

2. ТЕСТИРОВАНИЕ ArrayTabulatedFunction

2.1 Метод toString():

```
func1 = {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}
```

2.2 Метод equals():

func1.equals(func1) = true

func1.equals(func2) = true

func1.equals(func3) = false

func1.equals(func4) = false

func1.equals(null) = false

func1.equals("строка") = false

2.3 Метод hashCode():

func1.hashCode() = -1428276453

func2.hashCode() = -1428276453

func3.hashCode() = 1146600222

func4.hashCode() = 68246813

```
func1.hashCode() == func2.hashCode()? true  
func1.hashCode() == func3.hashCode()? false
```

2.4 Метод clone():

Оригинал: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

Клон: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

func1.equals(clone1)? true

func1 == clone1? false

Проверка глубокого клонирования:

Изменяем точку в оригинале...

Оригинал после изменения: {[0.0,0.0], [1.0,1.0], [2.0,100.0], [3.0,9.0], [4.0,16.0]}

Клон после изменения оригинала: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

Точка [2] в оригинале: [2.0,100.0]

Точка [2] в клоне: [2.0,4.0]

Они равны? false

Это один и тот же объект? false

3. ТЕСТИРОВАНИЕ LinkedListTabulatedFunction

3.1 Метод toString():

```
func1 = {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}
```

3.2 Метод equals():

func1.equals(func1) = true

func1.equals(func2) = true

func1.equals(func3) = false

3.3 Метод hashCode():

func1.hashCode() = -1428276453

func2.hashCode() = -1428276453

```
func3.hashCode() = 1146600222  
func1.hashCode() == func2.hashCode()? true  
func1.hashCode() == func3.hashCode()? false
```

3.4 Метод clone():

Оригинал: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

Клон: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

func1.equals(clone1)? true

func1 == clone1? false

Проверка глубокого клонирования:

Изменяем точку в оригинале...

Оригинал после изменения: {[0.0,0.0], [1.0,1.0], [2.0,100.0], [3.0,9.0], [4.0,16.0]}

Клон после изменения оригинала: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

Точка [2] в оригинале: [2.0,100.0]

Точка [2] в клоне: [2.0,4.0]

Они равны? false

Это один и тот же объект? false

Проверка структуры списка:

Удаляем точку из оригинала...

Оригинал после удаления: {[0.0,0.0], [2.0,100.0], [3.0,9.0], [4.0,16.0]}

Клон после удаления в оригинале: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

Количество точек в оригинале: 4

Количество точек в клоне: 5

4. СРАВНЕНИЕ РЕАЛИЗАЦИЙ

4.1 Сравнение toString():

Array: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

LinkedList: {[0.0,0.0], [1.0,1.0], [2.0,4.0], [3.0,9.0], [4.0,16.0]}

4.2 Сравнение equals():

arrayFunc.equals(listFunc) = true

listFunc.equals(arrayFunc) = true

4.3 Сравнение hashCode():

arrayFunc.hashCode() = -1428276453

listFunc.hashCode() = -1428276453

Хэш-коды равны? true

4.4 Тест на небольшое изменение:

Исходный хэш arrayFunc: -1428276453

Хэш после изменения y[2] на 4.001: -1794615816

Хэш после возврата y[2] к 4.0: -1428276453

Хэш после изменения y[3] на 9.005: 1276390735

4.5 Проверка согласованности equals() и hashCode():

arrayFunc.equals(listFunc)? false

Хэш-коды равны? false

==== ВСЕ ТЕСТЫ УСПЕШНО ЗАВЕРШЕНЫ ===

Выводы

1. Успешно переопределены все требуемые методы в трех классах
2. Обеспечена согласованность методов equals() и hashCode()
3. Реализовано глубокое клонирование для сложных структур данных
4. Сохранена совместимость между разными реализациями TabulatedFunction
5. Достигнута оптимизация:

о Прямой доступ к полям при сравнении объектов одного класса

о Эффективное клонирование связного списка без использования методов добавления

Проверенные свойства:

- ✓ Рефлексивность equals()
- ✓ Симметричность equals()
- ✓ Согласованность equals() и hashCode()
- ✓ Глубокое клонирование
- ✓ Независимость клона от оригинала
- ✓ Единый формат вывода для всех реализаций

Работа демонстрирует понимание принципов переопределения методов класса Object и их корректное применение для пользовательских классов в Java.