

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА»

Институт «Информатики и кибернетики»

Специальность «Фотоника и оптоинформатика 6201-120303D»

Отчет по лабораторной работе № 5

Выполнил: студент Султанова А.М.,

группа 6201-120303D

Проверил: преподаватель Борисов Д.С.

Самара 2025

Задание 1

Переопределим в классе FunctionPoint следующие методы:

toString() возвращает текстовое описание точки (x; y).

equals(Object o) возвращает true, если другой объект тоже FunctionPoint и координаты совпадают. Необходимо корректно сравнивать double.

clone(): возвращает объект-копию, простое клонирование достаточно.

```
public String toString() { new *
    return "(" + x + "; " + y + ")";
}

public boolean equals(Object o) { new *
    if (o == this) return true;
    if (o == null) return false;
    if (!(o instanceof FunctionPoint)) return false;

    FunctionPoint p = (FunctionPoint) o;

    return Math.abs(this.x - p.x) < 1e-9 && Math.abs(this.y - p.y) < 1e-9;
}

public int hashCode() { new *
    long xb = Double.doubleToLongBits(x);
    long yb = Double.doubleToLongBits(y);

    int x1 = (int)(xb >>> 32);
    int x2 = (int)(xb & 0xFFFFFFFF);
    int y1 = (int)(yb >>> 32);
    int y2 = (int)(yb & 0xFFFFFFFF);

    return x1 ^ x2 ^ y1 ^ y2;
}

public Object clone() { new *
    return new FunctionPoint(this.x, this.y);
}
```

Задание 2

В классе `ArrayTabulatedFunction` были переопределены методы, унаследованные от класса `Object`, для расширения функционала табулированной функции:

`toString()` - формирует текстовое представление функции в виде множества точек $\{(x_0; y_0), (x_1; y_1), \dots\}$. Для эффективного создания строки использовался класс `StringBuilder` и метод `append()`, что позволило последовательно добавлять координаты каждой точки и разделять их запятыми.

`equals(Object o)` - реализовано сравнение с любым объектом, реализующим интерфейс `TabulatedFunction`. Метод возвращает `true`, если количество точек совпадает и все координаты совпадают с точностью до малой погрешности (`EPS`). Для объектов класса `ArrayTabulatedFunction` использовано прямое обращение к массиву точек для ускорения работы.

`hashCode()` - рассчитан на основе количества точек и хэш-кодов всех точек функции. Используется операция `XOR` для объединения хэш-кодов, что обеспечивает различие хэш-кодов функций с разным набором точек.

`clone()` - выполнено глубокое клонирование. Создаётся новый массив точек, каждая точка клонируется с помощью метода `clone()`, и создаётся новый объект `ArrayTabulatedFunction`, содержащий копии всех точек. Это обеспечивает независимость клона от исходного объекта.

```

public String toString() { & SultanovaArina
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    for (int i = 0; i < size; i++) {
        sb.append("(").append(points[i].getX()).append("; ").append(points[i].getY()).append(")");
        if (i != size - 1) sb.append(", ");
    }
    sb.append("}");
    return sb.toString();
}

public boolean equals(Object o) { & SultanovaArina *
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction f = (TabulatedFunction) o;

    if (this.size != f.getPointsCount()) return false;

    // оптимизация Array - Array
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction af = (ArrayTabulatedFunction) o;
        for (int i = 0; i < size; i++) {
            if (!this.points[i].equals(af.points[i])) return false;
        }
        return true;
    }

    // сравнение с LinkedList
    for (int i = 0; i < size; i++) {
        if (!this.points[i].equals(f.getPoint(i))) return false;
    }
    return true;
}

public int hashCode() { & SultanovaArina
    int hash = size;
    for (int i = 0; i < size; i++) {
        hash ^= points[i].hashCode();
    }
    return hash;
}

public Object clone() throws CloneNotSupportedException { & SultanovaArina
    FunctionPoint[] newPoints = new FunctionPoint[size];
    for (int i = 0; i < size; i++) {
        newPoints[i] = (FunctionPoint) points[i].clone(); // глубокое клонирование
    }
    return new ArrayTabulatedFunction(newPoints);
}

```

Задание 3

Расширяем класс `LinkedListTabulatedFunction`, переопределив методы `toString()`, `equals()`, `hashCode()` и `clone()`.

Метод `toString()` - Создан для текстового представления табулированной функции. Используется `StringBuilder` для формирования строки вида `{(x0; y0), (x1; y1), ...}`. Цикл проходит по всем узлам списка (`FunctionNode`) и добавляет координаты точек.

Метод `equals(Object o)` - Проверяет равенство двух табулированных функций.

Сначала сравнивается тип объекта и количество точек. Затем последовательно сравниваются координаты всех точек с использованием точности EPS для чисел с плавающей запятой. Реализовано так, что метод корректно работает при сравнении с любым объектом, реализующим интерфейс `TabulatedFunction`.

Метод `hashCode()` - Формирует уникальный хэш-код функции. Используется побитовое XOR между хэш-кодами всех точек и размером списка.

Обеспечивает согласованность методов `equals()` и `hashCode()`.

Метод `clone()` - Реализует глубокое клонирование списка. Создаётся новый объект `LinkedListTabulatedFunction`, поочерёдно добавляются копии всех точек из исходного списка. Гарантирует, что изменение исходного списка после клонирования не влияет на клон.

```
public String toString() {    ^SultanovaArina
    StringBuilder sb = new StringBuilder();
    sb.append("{");
    FunctionNode current = head.next;
    while (current != head) {
        sb.append("(").append(current.point.getX()).append("; ").append(current.point.getY()).append(")");
        if (current.next != head) sb.append(", ");
        current = current.next;
    }
    sb.append("}");
    return sb.toString();
}

public boolean equals(Object o) {    ^SultanovaArina*
    if (this == o) return true;
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction f = (TabulatedFunction) o;

    if (this.size != f.getPointsCount()) return false;

    // оптимизация LinkedList - LinkedList
    if (o instanceof LinkedListTabulatedFunction) {
        FunctionNode a = this.head.next;
        FunctionNode b = ((LinkedListTabulatedFunction) o).head.next;

        while (a != this.head) {
            if (!a.point.equals(b.point)) return false;
            a = a.next;
            b = b.next;
        }
        return true;
    }

    // сравнение с ArrayTabulatedFunction
    for (int i = 0; i < size; i++) {
        if (!this.getPoint(i).equals(f.getPoint(i))) return false;
    }

    return true;
}

public int hashCode() {    ^SultanovaArina
    int result = size;
    FunctionNode current = head.next;
    while (current != head) {
        result ^= current.point.hashCode();
        current = current.next;
    }
    return result;
}
```

```

public Object clone() throws CloneNotSupportedException {
    // создаём массив для клонирования точек
    FunctionPoint[] clonedPoints = new FunctionPoint[this.size]; // используем size
    FunctionNode current = this.head.next;

    for (int i = 0; i < this.size; i++) {
        clonedPoints[i] = (FunctionPoint) current.point.clone(); // глубокое клонирование точки
        current = current.next;
    }

    // создаём новый объект LinkedListTabulatedFunction из массива точек
    return new LinkedListTabulatedFunction(clonedPoints);
}

```

Задание 4

В интерфейс TabulatedFunction добавлен метод Object clone() throws CloneNotSupportedException.

```

package functions;

public interface TabulatedFunction extends Function, Cloneable {
    int getPointsCount(); 4 usages 2 implementations & SultanovaArina *
    double getPointX(int index); 9 usages 2 implementations & SultanovaArina
    double getPointY(int index); 5 usages 2 implementations & SultanovaArina
    void setPointX(int index, double x) throws InappropriateFunctionPointException; no usages 2 implementations & SultanovaArina
    void setPointY(int index, double y); 2 usages 2 implementations & SultanovaArina
    FunctionPoint getPoint(int index); 3 usages 2 implementations & SultanovaArina
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; no usages 2 implementations & SultanovaArina
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; no usages 2 implementations & SultanovaArina
    void deletePoint(int index); no usages 2 implementations & SultanovaArina

    Object clone() throws CloneNotSupportedException; 2 implementations & SultanovaArina
}

```

Все классы, реализующие интерфейс (ArrayTabulatedFunction и LinkedListTabulatedFunction), реализуют этот метод с глубоким копированием точек, что делает объекты клонируемыми с точки зрения JVM.

```

public Object clone() throws CloneNotSupportedException { new *
    FunctionPoint[] newPoints = new FunctionPoint[size];
    for (int i = 0; i < size; i++) {
        newPoints[i] = (FunctionPoint) points[i].clone(); // глубокое клонирование
    }
    return new ArrayTabulatedFunction(newPoints);
}

```

```
public Object clone() throws CloneNotSupportedException { new *
    // создаём массив для клонирования точек
    FunctionPoint[] clonedPoints = new FunctionPoint[this.size]; // используем size
    FunctionNode current = this.head.next;

    for (int i = 0; i < this.size; i++) {
        clonedPoints[i] = (FunctionPoint) current.point.clone(); // глубокое клонирование точки
        current = current.next;
    }

    // создаём новый объект LinkedListTabulatedFunction из массива точек
    return new LinkedListTabulatedFunction(clonedPoints);
}
```

Задание 5

Проверим написанные методы:

```

1 import functions.*;
2
3 public class Main { & SultanovaArina *
4     public static void main(String[] args) throws CloneNotSupportedException { new *
5         |
6         // Создание тестовых объектов
7         double[] values1 = {1.0, 2.0, 3.0, 4.0};
8         double[] values2 = {1.0, 2.0, 3.1, 4.0}; // немного изменённая точка для проверки equals и hashCode
9
10        ArrayTabulatedFunction arrayFunc1 = new ArrayTabulatedFunction( leftX: 0.0, rightX: 3.0, values1);
11        ArrayTabulatedFunction arrayFunc2 = new ArrayTabulatedFunction( leftX: 0.0, rightX: 3.0, values1);
12        ArrayTabulatedFunction arrayFunc3 = new ArrayTabulatedFunction( leftX: 0.0, rightX: 3.0, values2);
13
14        LinkedListTabulatedFunction listFunc1 = new LinkedListTabulatedFunction( leftX: 0.0, rightX: 3.0, values1);
15        LinkedListTabulatedFunction listFunc2 = new LinkedListTabulatedFunction( leftX: 0.0, rightX: 3.0, values1);
16        LinkedListTabulatedFunction listFunc3 = new LinkedListTabulatedFunction( leftX: 0.0, rightX: 3.0, values2);
17
18        // Проверка toString()
19        System.out.println("\ntoString() ");
20        System.out.println("ArrayTabulatedFunction: " + arrayFunc1);
21        System.out.println("LinkedListTabulatedFunction: " + listFunc1);
22
23        // Проверка equals()
24        System.out.println("\nequals()");
25        System.out.println("arrayFunc1.equals(arrayFunc2): " + arrayFunc1.equals(arrayFunc2)); // true
26        System.out.println("arrayFunc1.equals(arrayFunc3): " + arrayFunc1.equals(arrayFunc3)); // false
27        System.out.println("arrayFunc1.equals(listFunc1): " + arrayFunc1.equals(listFunc1)); // true
28        System.out.println("listFunc1.equals(listFunc3): " + listFunc1.equals(listFunc3)); // false
29
30        // Проверка hashCode()
31        System.out.println("\nhashCode()");
32        System.out.println("arrayFunc1.hashCode(): " + arrayFunc1.hashCode());
33        System.out.println("arrayFunc2.hashCode(): " + arrayFunc2.hashCode());
34        System.out.println("arrayFunc3.hashCode(): " + arrayFunc3.hashCode());
35        System.out.println("listFunc1.hashCode(): " + listFunc1.hashCode());
36        System.out.println("listFunc2.hashCode(): " + listFunc2.hashCode());
37        System.out.println("listFunc3.hashCode(): " + listFunc3.hashCode());
38
39        // Проверка clone()
40        System.out.println("\n clone() и проверка глубокого клонирования");
41        ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc1.clone();
42        LinkedListTabulatedFunction listClone = (LinkedListTabulatedFunction) listFunc1.clone();
43
44        System.out.println("Клоны до изменения исходных объектов:");
45        System.out.println("arrayClone: " + arrayClone);
46        System.out.println("listClone: " + listClone);
47
48        // Изменяем исходные объекты
49        arrayFunc1.setPointY( index: 0, y: 10.0);
50        listFunc1.setPointY( index: 0, y: 10.0);
51
52        System.out.println("\nПосле изменения исходных объектов:");
53        System.out.println("Исходный arrayFunc1: " + arrayFunc1);
54        System.out.println("Клон arrayClone (должен остаться прежним): " + arrayClone);
55
56        System.out.println("Исходный listFunc1: " + listFunc1);
57
58        System.out.println("Исходный listFunc1: " + listFunc1);
59        System.out.println("Клон listClone (должен остаться прежним): " + listClone);
60
61        // Проверка equals() между клоном и исходным после изменения
62        System.out.println("\nПроверка equals() после изменения исходных объектов:");
63        System.out.println("arrayFunc1.equals(arrayClone): " + arrayFunc1.equals(arrayClone)); // false
64        System.out.println("listFunc1.equals(listClone): " + listFunc1.equals(listClone)); // false
65    }

```

```
toString()
ArrayTabulatedFunction: {(0.0; 1.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
LinkedListTabulatedFunction: {(0.0; 1.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
```

```
equals()
arrayFunc1.equals(arrayFunc2): true
arrayFunc1.equals(arrayFunc3): false
arrayFunc1.equals(listFunc1): true
listFunc1.equals(listFunc3): false
```

```
hashCode()
arrayFunc1.hashCode(): 1074790404
arrayFunc2.hashCode(): 1074790404
arrayFunc3.hashCode(): -1931739131
listFunc1.hashCode(): 1074790404
listFunc2.hashCode(): 1074790404
listFunc3.hashCode(): -1931739131
```

clone() и проверка глубокого клонирования

Клоны до изменения исходных объектов:

```
arrayClone: {(0.0; 1.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
listClone: {(0.0; 1.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
```

После изменения исходных объектов:

```
Исходный arrayFunc1: {(0.0; 10.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
Клон arrayClone (должен остаться прежним): {(0.0; 1.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
Исходный listFunc1: {(0.0; 10.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
Клон listClone (должен остаться прежним): {(0.0; 1.0), (1.0; 2.0), (2.0; 3.0), (3.0; 4.0)}
```

Проверка equals() после изменения исходных объектов:

```
arrayFunc1.equals(arrayClone): false
listFunc1.equals(listClone): false
```

```
Process finished with exit code 0
```