

# Лабораторная работа по ООП №5

Студент: Тарасова Полина

Группа: 6204-010302D

## Задание 1

Я переопределила методы `toString()`, `equals()`, `hashCode()` и `clone()` для класса `FunctionPoint`.

- Метод `toString()` выводит координаты точки в читаемом виде (x; y). Это удобно для проверки работы функций и отображения результата в консоли.
- Метод `equals(Object o)` сравнивает переданный объект с текущей точкой. Я использовала `Double.compare`, чтобы корректно сравнивать числа типа `double`. Это обеспечивает точное сравнение координат.
- Метод `hashCode()` реализован с использованием XOR. Каждое значение типа `double` преобразуется в два значения типа `int` через `Double.doubleToLongBits()`. Хэш-код рассчитывается как XOR всех полученных `int`-значений координат.
- Метод `clone()` возвращает новый объект `FunctionPoint` с такими же координатами, что обеспечивает простое клонирование.

```
@Override
public String toString() {
    return "(" + x + ";" + y + ")";
}

@Override
public boolean equals(Object o) {
    if (!(o instanceof FunctionPoint)) {
        return false;
    }
    FunctionPoint other = (FunctionPoint) o;
    if (Math.abs(this.x - other.x) < EPS && Math.abs(this.y - other.y) <
EPS) {
        return true;
    }
    else {
        return false;
    }
}

@Override
public int hashCode() {
    long bitsX = Double.doubleToLongBits(x);
    int lowerX = (int) (bitsX & 0xFFFFFFFFL);
    int upperX = (int) (bitsX >>> 32);
    int hashX = lowerX ^ upperX;
    long bitsY = Double.doubleToLongBits(y);
    int lowerY = (int) (bitsY & 0xFFFFFFFFL);
    int upperY = (int) (bitsY >>> 32);
    int hashY = lowerY ^ upperY;
    int finalHash = hashX ^ hashY;
    return finalHash;
}
```

```

}

@Override
public Object clone() {
    FunctionPoint clonePoint = new FunctionPoint(this.x, this.y);
    return clonePoint;
}

```

## Задание 2

В `ArrayTabulatedFunction` я переопределила методы `toString()`, `equals()`, `hashCode()` и `clone()` для массива точек.

- `toString()` выводит функцию в виде списка точек.
- `equals(Object o)` сравнивает все точки с другой функцией. Для ускорения сравнения с объектами того же класса я использовала прямой доступ к массиву точек.
- `hashCode()` рассчитывается как XOR хэш-кодов всех точек и количество точек.
- `clone()` создает глубокую копию функции: массив точек копируется полностью, чтобы изменения исходного объекта не затрагивали клон.

```

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

@Override
public boolean equals(Object o) {
    if (!(o instanceof TabulatedFunction)) {
        return false;
    }
    TabulatedFunction other = (TabulatedFunction) o;
    if (this.getPointsCount() != other.getPointsCount()) {
        return false;
    }
    if (o instanceof ArrayTabulatedFunction) {

```

```

        ArrayTabulatedFunction otherArray = (ArrayTabulatedFunction) o;
        for (int i = 0; i < pointsCount; i++) {

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

@Override
public int hashCode() {
    int hash = pointsCount;
    for (int i = 0; i < pointsCount; i++) {
        hash ^= points[i].hashCode();
    }
    return hash;
}

@Override
public Object clone() {
    double leftX = points[0].getX();
    double rightX = points[pointsCount - 1].getX();
    double[] values = new double[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        values[i] = points[i].getY();
    }
    ArrayTabulatedFunction cloneFunc = new
    ArrayTabulatedFunction(leftX, rightX, values);
    return cloneFunc;
}

```

## Задание 3

В `LinkedListTabulatedFunction` я аналогично переопределила `toString()`, `equals()`, `hashCode()` и `clone()`. Но здесь есть несколько особенностей:

- В `equals()` используется прямой доступ к узлам списка для ускорения сравнения.
- В `clone()` создается новый список, пересобирая узлы и копируя точки, чтобы обеспечить глубокое клонирование.

```

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

```

```

        while (current != head) {
            sb.append("(")
                .append(current.point.getX()).append("; ")
                .append(current.point.getY()).append(")");
            if (current != head.prev) {
                sb.append(", ");
            }
            current = current.next;
        }
        sb.append(")");
        return sb.toString();
    }

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof TabulatedFunction)) {
            return false;
        }
        TabulatedFunction other = (TabulatedFunction) o;
        if (this.getPointsCount() != other.getPointsCount()) {
            return false;
        }
        if (o instanceof LinkedListTabulatedFunction) {
            LinkedListTabulatedFunction otherList =
                (LinkedListTabulatedFunction) o;
            FunctionNode current1 = this.head.next;
            FunctionNode current2 = otherList.head.next;
            while (current1 != head && current2 != head) {
                if (!current1.point.equals(current2.point)) {
                    return false;
                }
                current1 = current1.next;
                current2 = current2.next;
            }
        }
        else {
            for (int i = 0; i < pointsCount; i++) {
                if (!this.getPoint(i).equals(other.getPoint(i))) {
                    return false;
                }
            }
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = pointsCount;
        FunctionNode current = head.next;
        while (current != head) {
            hash ^= current.point.hashCode();
            current = current.next;
        }
        return hash;
    }

    @Override
    public Object clone() {
        LinkedListTabulatedFunction clone = new
            LinkedListTabulatedFunction();
        FunctionNode current = this.head.next;
        while (current != this.head) {
            FunctionPoint newPoint = (FunctionPoint) current.point.clone();
            FunctionNode newNode = new FunctionNode(newPoint);

```

```

        FunctionNode last = clone.head.prev;
        last.next = newNode;
        newNode.prev = last;
        newNode.next = clone.head;
        clone.head.prev = newNode;
        current = current.next;
    }
    clone.pointsCount = this.pointsCount;
    return clone;
}

```

## Задание 4

Я добавила метод clone() в интерфейс TabulatedFunction, чтобы объекты любого типа функции могли быть клонированы с точки зрения JVM.

```

public interface TabulatedFunction extends Function, Cloneable {
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws
    InappropriateFunctionPointException;
    double getPointX(int index);
    void setPointX(int index, double x) throws
    InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws
    InappropriateFunctionPointException;
    Object clone();
}

```

## Задание 5

В классе Main я проверила все методы для объектов ArrayTabulatedFunction и LinkedListTabulatedFunction. Все методы переопределены корректно, клонирование глубокое, equals() и hashCode() согласованы.

```

public class Main {
    public static void main(String[] args) throws IOException {
        double EPS = 1e-9;

        double[] vals1 = {0, 1, 4, 9, 16};
        double[] vals2 = {0, 3, 6, 10, 19};
        TabulatedFunction func1 = new ArrayTabulatedFunction(0, 5, vals1);
        TabulatedFunction func2 = new ArrayTabulatedFunction(0, 5, vals2);
        TabulatedFunction func3 = new ArrayTabulatedFunction(0, 5, vals1);
        System.out.println("Вывод через ArrayTabulatedFunction:");
        System.out.println("func1 = " + func1);
        System.out.println("func2 = " + func2);
        System.out.println("func3 = " + func3);
    }
}

```

```

        System.out.println();

        TabulatedFunction func4 = new LinkedListTabulatedFunction(0, 5,
vals1);
        TabulatedFunction func5 = new LinkedListTabulatedFunction(0, 5,
vals2);
        System.out.println("Вывод через LinkedListTabulatedFunction:");
        System.out.println("func4 = " + func4);
        System.out.println("func5 = " + func5);
        System.out.println();

        System.out.println("Проверка equals:");
        System.out.println("Сравнение func1 и func3: " +
func1.equals(func3));
        System.out.println("Сравнение func1 и func2: " +
func1.equals(func2));
        System.out.println("Сравнение func1 и func4: " +
func1.equals(func4));
        System.out.println("Сравнение func1 и func5: " +
func1.equals(func5));
        System.out.println();

        System.out.println("Проверка hashCode:");
        System.out.println("func1 = " + func1.hashCode());
        System.out.println("func2 = " + func2.hashCode());
        System.out.println("func3 = " + func3.hashCode());
        System.out.println("func4 = " + func4.hashCode());
        System.out.println("func5 = " + func5.hashCode());
        System.out.println();
        System.out.println("Вывод hashCode функций с изменёнными точками на
0.001:");
        func1.setPointY(1, func1.getPointY(1) + 0.001);
        func2.setPointY(2, func2.getPointY(1) + 0.001);
        func3.setPointY(3, func3.getPointY(1) + 0.001);
        func4.setPointY(4, func4.getPointY(1) + 0.001);
        func5.setPointY(0, func5.getPointY(1) + 0.001);
        System.out.println("func1 = " + func1.hashCode());
        System.out.println("func2 = " + func2.hashCode());
        System.out.println("func3 = " + func3.hashCode());
        System.out.println("func4 = " + func4.hashCode());
        System.out.println("func5 = " + func5.hashCode());
        System.out.println();

        TabulatedFunction funcClone1 = (TabulatedFunction) func1.clone();
        TabulatedFunction funcClone2 = (TabulatedFunction) func5.clone();
        System.out.println("Проверка Clone:");
        System.out.println("Копия func1: " + funcClone1);
        System.out.println("Копия func5: " + funcClone2);
        System.out.println();
        System.out.println("Сравнение исходной и клонированной функций:");
        System.out.println("funcClone1: " + func1.equals(funcClone1));
        System.out.println("funcClone2: " + func5.equals(funcClone2));
        System.out.println();
        System.out.println("Проверка hashCode у копий функций:");
        System.out.println("funcClone1: " + funcClone1.hashCode());
        System.out.println("funcClone2: " + funcClone2.hashCode());
        System.out.println();
        func1.setPointY(1, func1.getPointY(1) + 3);
        func5.setPointY(0, func1.getPointY(1) + 6);
        System.out.println("Проверка на глубокое копирование:");
        System.out.println("Изменённая func1: " + func1);
        System.out.println("Копия func1: " + funcClone1);
        System.out.println("Изменённая func5: " + func5);
        System.out.println("Копия func5: " + funcClone2);

```

```
}  
}
```

Пример работы программы:

Вывод через ArrayTabulatedFunction:

```
func1 = {(0.0; 0.0), (1.25; 1.0), (2.5; 4.0), (3.75; 9.0), (5.0; 16.0)}
```

```
func2 = {(0.0; 0.0), (1.25; 3.0), (2.5; 6.0), (3.75; 10.0), (5.0; 19.0)}
```

```
func3 = {(0.0; 0.0), (1.25; 1.0), (2.5; 4.0), (3.75; 9.0), (5.0; 16.0)}
```

Вывод через LinkedListTabulatedFunction:

```
func4 = {(0.0; 0.0), (1.25; 1.0), (2.5; 4.0), (3.75; 9.0), (5.0; 16.0)}
```

```
func5 = {(0.0; 0.0), (1.25; 3.0), (2.5; 6.0), (3.75; 10.0), (5.0; 19.0)}
```

Проверка equals:

Сравнение func1 и func3: true

Сравнение func1 и func2: false

Сравнение func1 и func4: true

Сравнение func1 и func5: false

Проверка hashCode:

```
func1 = 1572869
```

```
func2 = 2146238469
```

```
func3 = 1572869
```



func4 = 1572869

func5 = 2146238469

Вывод hashCode функций с изменёнными точками на 0.001:

func1 = -1821591433

func2 = 910646332

func3 = -323045257

func4 = -324224905

func5 = 1985961020

Проверка Clone:

Копия func1: {(0.0; 0.0), (1.25; 1.001), (2.5; 4.0), (3.75; 9.0), (5.0; 16.0)}

Копия func5: {(0.0; 3.001), (1.25; 3.0), (2.5; 6.0), (3.75; 10.0), (5.0; 19.0)}

Сравнение исходной и клонированной функций:

funcClone1: true

funcClone2: true

Проверка hashCode у копий функций:

funcClone1: -1821591433

funcClone2: 1985961020

Проверка на глубокое копирование:

Изменённая func1: {(0.0; 0.0), (1.25; 4.0009999999999994), (2.5; 4.0), (3.75; 9.0), (5.0; 16.0)}

Копия func1: {(0.0; 0.0), (1.25; 1.001), (2.5; 4.0), (3.75; 9.0), (5.0; 16.0)}

Изменённая func5: {(0.0; 10.001), (1.25; 3.0), (2.5; 6.0), (3.75; 10.0), (5.0; 19.0)}

Копия func5: {(0.0; 3.001), (1.25; 3.0), (2.5; 6.0), (3.75; 10.0), (5.0; 19.0)}