

## Задание 1

```
@Override
public String toString() {
    // Форматирование с точкой с запятой как в примере задания
    return "(" + x + ";" + y + ")";
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    FunctionPoint that = (FunctionPoint) o;

    // Используем Double.compare для корректного сравнения чисел с плавающей точкой
    // и специальных значений (NaN, Infinity)
    return Double.compare(that.x, x) == 0 && Double.compare(that.y, y) == 0;
}

@Override
public int hashCode() {
    // Преобразуем double в long bits
    long xBits = Double.doubleToLongBits(x);
    long yBits = Double.doubleToLongBits(y);

    // Разбиваем каждый long на два int (старшие и младшие 4 байта)
    int xHigh = (int)(xBits >>> 32);
    int xLow = (int)xBits;
    int yHigh = (int)(yBits >>> 32);
    int yLow = (int)yBits;

    // Комбинируем все значения с помощью XOR
    return xHigh ^ xLow ^ yHigh ^ yLow;
}

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

## Задание 2

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(str: "{");
    for (int i = 0; i < pointsCount; i++) {
        if (i > 0) sb.append(str: ", ");
        sb.append(points[i].toString());
    }
    sb.append(str: "}");
    return sb.toString();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    // Проверяем, является ли объект табулированной функцией
    if (!(o instanceof TabulatedFunction)) return false;
    TabulatedFunction other = (TabulatedFunction) o;
    // Проверяем количество точек
    if (this.getPointsCount() != other.getPointsCount()) return false;
    // Оптимизация для ArrayTabulatedFunction
    if (o instanceof ArrayTabulatedFunction) {
        ArrayTabulatedFunction otherArray = (ArrayTabulatedFunction) o;
        // Быстрая проверка - тот же объект или разное количество точек
        if (this == otherArray) return true;
        if (this.pointsCount != otherArray.pointsCount) return false;

        // Сравниваем точки через их метод equals
        for (int i = 0; i < pointsCount; i++) {
            if (!this.points[i].equals(otherArray.points[i])) {
                return false;
            }
        }
    } else {
        // Общий случай для любой реализации TabulatedFunction
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = this.getPoint(i);
            FunctionPoint otherPoint = other.getPoint(i);

            if (!thisPoint.equals(otherPoint)) {
                return false;
            }
        }
    }
    return true;
}
```

```
@Override
public int hashCode() {
    int result = pointsCount; // Включаем количество точек

    for (int i = 0; i < pointsCount; i++) {
        // Используем хэш-код каждой точки и комбинируем через XOR
        result ^= points[i].hashCode();
    }

    return result;
}

@Override
public ArrayTabulatedFunction clone() {
    try {
        ArrayTabulatedFunction cloned = (ArrayTabulatedFunction) super.clone();

        // Глубокое клонирование массива точек
        cloned.points = new FunctionPoint[this.points.length];
        for (int i = 0; i < this.pointsCount; i++) {
            cloned.points[i] = (FunctionPoint) this.points[i].clone();
        }

        cloned.pointsCount = this.pointsCount;

        return cloned;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError(message: "Клонирование не поддерживается", e);
    }
}
```

### Задание 3

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(str: "{");

    FunctionNode currentNode = head.getNext();
    boolean first = true;
    while (currentNode != head) {
        if (!first) {
            sb.append(str: ", ");
        }
        sb.append(currentNode.getPoint().toString());
        currentNode = currentNode.getNext();
        first = false;
    }
    sb.append(str: "}");
    return sb.toString();
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;

    // Проверяем, является ли объект табулированной функцией
    if (!(o instanceof TabulatedFunction)) return false;

    TabulatedFunction other = (TabulatedFunction) o;

    // Проверяем количество точек
    if (this.getPointsCount() != other.getPointsCount()) return false;

    // Оптимизация для LinkedListTabulatedFunction
    if (o instanceof LinkedListTabulatedFunction) {
        LinkedListTabulatedFunction otherList = (LinkedListTabulatedFunction) o;

        // Если оба пустые
        if (this.pointsCount == 0 && otherList.pointsCount == 0) return true;

        // Обходим оба списка одновременно
        FunctionNode thisCurrent = this.head.getNext();
        FunctionNode otherCurrent = otherList.head.getNext();

        while (thisCurrent != this.head && otherCurrent != otherList.head) {
            if (!thisCurrent.getPoint().equals(otherCurrent.getPoint())) {
                return false;
            }
            thisCurrent = thisCurrent.getNext();
            otherCurrent = otherCurrent.getNext();
        }
    }

    // Проверяем, что оба дошли до конца одновременно
    return (thisCurrent == this.head && otherCurrent == otherList.head);
}
```

```
    } else {
        // Общий случай для любой реализации TabulatedFunction
        for (int i = 0; i < pointsCount; i++) {
            FunctionPoint thisPoint = this.getPoint(i);
            FunctionPoint otherPoint = other.getPoint(i);

            if (!thisPoint.equals(otherPoint)) {
                return false;
            }
        }
        return true;      You, сейчас • Uncommitted changes
    }
}

@Override
public int hashCode() {
    int result = pointsCount; // Включаем количество точек

    FunctionNode currentNode = head.getNext();
    while (currentNode != head) {
        result ^= currentNode.getPoint().hashCode();
        currentNode = currentNode.getNext();
    }

    return result;
}
```

```
@Override
public LinkedListTabulatedFunction clone() {
    try {
        LinkedListTabulatedFunction cloned = (LinkedListTabulatedFunction) super.clone();

        // Создаем новый головной узел
        cloned.head = new FunctionNode(point: null);
        cloned.head.setPrev(cloned.head);
        cloned.head.setNext(cloned.head);

        if (pointsCount > 0) {
            FunctionNode currentOriginal = this.head.getNext();
            FunctionNode prevCloned = null;
            FunctionNode firstCloned = null;

            // Создаем копии всех узлов
            while (currentOriginal != this.head) {
                // Создаем глубокую копию точки
                FunctionPoint copiedPoint = (FunctionPoint) currentOriginal.getPoint().clone();

                // Создаем новый узел
                FunctionNode newNode = new FunctionNode(copiedPoint);

                // Связываем
                if (prevCloned == null) {
                    firstCloned = newNode;
                    newNode.setPrev(cloned.head);
                    cloned.head.setNext(newNode);
                } else {
                    newNode.setPrev(prevCloned);
                    prevCloned.setNext(newNode);
                }

                // Обновляем кэш, если это был кэшированный узел
                if (currentOriginal == this.lastAccessedNode) {
                    cloned.lastAccessedNode = newNode;
                    cloned.lastAccessedIndex = this.lastAccessedIndex;
                }

                prevCloned = newNode;
                currentOriginal = currentOriginal.getNext();
            }

            // Замыкаем список
            if (prevCloned != null) {
                prevCloned.setNext(cloned.head);
                cloned.head.setPrev(prevCloned);
            }

            cloned.pointsCount = this.pointsCount;
        } else {
            cloned.pointsCount = 0;
            cloned.lastAccessedNode = cloned.head;
            cloned.lastAccessedIndex = -1;
        }

        return cloned;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError(message: "Клонирование не поддерживается", e);
    }
}
```

#### Задание 4

```
package functions;

You, 6 дней назад | 1 author (You)
public interface TabulatedFunction extends Function, Cloneable {

    int getPointsCount();

    // Методы работы с точками
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;

    // Методы работы с координатами точек
    double getPointX(int index);
    void setPointX(int index, double x) throws InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);

    // Методы модификации набора точек
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
    void deletePoint(int index);

    // Методы Object, которые должны быть переопределены
    String toString();
    boolean equals(Object o);
    int hashCode();          You, 6 дней назад • Uncommitted changes
    Object clone();
}
```

## Задание 5

```
1. ТЕСТИРОВАНИЕ toString()
=====
ArrayTabulatedFunction 1: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}
ArrayTabulatedFunction 2: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}
LinkedListTabulatedFunction 1: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}
LinkedListTabulatedFunction 2: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.3), (3.0; 7.1), (4.0; 9.5)}

Проверка формата toString():
Первый символ: '{' (ожидается '{')
Последний символ: '}' (ожидается '}')
Содержит ли точки в формате (x; y): true

2. ТЕСТИРОВАНИЕ equals()
=====
arrayFunc1.equals(arrayFunc2) (одинаковые массивы): true
arrayFunc1.equals(linkedFunc1) (разные классы, но одинаковые данные): true
linkedFunc1.equals(linkedFunc2) (разные данные): false
arrayFunc1.equals(arrayFunc3) (разные данные): false
arrayFunc1.equals(arrayFunc4) (разное количество точек): false
arrayFunc1.equals(null): false
arrayFunc1.equals("строка"): false
arrayFunc1.equals(arrayFunc1) (рефлексивность): true

Тест симметричности equals():
arrayFunc1.equals(linkedFunc1): true
linkedFunc1.equals(arrayFunc1): true

Тест транзитивности (для одинаковых объектов):
arrayFunc1.equals(arrayFunc2): true
arrayFunc2.equals(arrayFunc5): true
arrayFunc1.equals(arrayFunc5): true
```

### 3. ТЕСТИРОВАНИЕ hashCode()

```
=====
Хэш-код arrayFunc1: 357629957
Хэш-код arrayFunc2: 357629957
Хэш-код linkedFunc1: 357629957
Хэш-код linkedFunc2: -357695482
Хэш-код arrayFunc3: -357695482
Хэш-код arrayFunc4: 1930428419
```

Проверка согласованности equals() и hashCode():

```
arrayFunc1.equals(arrayFunc2) = true, hashCodes equal: true
arrayFunc1.equals(arrayFunc3) = false, hashCodes equal: false
arrayFunc1.equals(arrayFunc4) = false, hashCodes equal: false
```

Изменение объекта и проверка хэш-кода:

```
Хэш-код arrayFunc1 до изменения: 357629957
Хэш-код arrayFunc1 после изменения Y[2] на +0.005: -1313327911
Хэш-код arrayFunc1 после возврата исходного значения: 357629957
Хэш-код вернулся к исходному: true
```

### 4. ТЕСТИРОВАНИЕ clone()

Тестирование клонирования ArrayTabulatedFunction:

```
Оригинал: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}
Клон: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}
arrayOriginal.equals(arrayClone): true
arrayOriginal == arrayClone: false
```

Изменяем оригинал (увеличиваем Y[0] на 10):

```
Оригинал после изменения: {(0.0; 11.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}
Клон после изменения оригинала: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}
Y[0] в оригиналe: 11.2
Y[0] в клоне: 1.2
После изменения оригинала equals: false
? Клон не изменился (глубокое клонирование работает)
```

```
Тестирование клонирования LinkedListTabulatedFunction:  
Оригинал: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}  
Клон: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}  
linkedOriginal.equals(linkedClone): true  
linkedOriginal == linkedClone: false  
  
Изменяем оригинал (увеличиваем Y[1] на 5):  
Оригинал после изменения: {(0.0; 1.2), (1.0; 8.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}  
Клон после изменения оригинала: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}  
Y[1] в оригинале: 8.8  
Y[1] в клоне: 3.8  
После изменения оригинала equals: false  
? Клон не изменился (глубокое клонирование работает)  
  
Тест глубокого клонирования (множественные изменения):  
Оригинал после изменений: {(0.0; 100.0), (1.0; 3.8), (2.0; 200.0), (3.0; 7.1), (4.0; 300.0)}  
Клон после изменений оригинала: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}  
? Глубокое клонирование LinkedListTabulatedFunction работает корректно  
  
Дополнительный тест глубокого клонирования для ArrayTabulatedFunction:  
Оригинал после изменений: {(0.0; 1.2), (1.0; 50.0), (2.0; 15.2), (3.0; 75.0), (4.0; 9.5)}  
Клон после изменений оригинала: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}  
? Глубокое клонирование ArrayTabulatedFunction работает корректно  
  
5. ДОПОЛНИТЕЛЬНЫЕ ТЕСТЫ  
=====
```

Тест с конструкторами ArrayTabulatedFunction:

```
ArrayTabulatedFunction(0, 10, 5): {(0.0; 0.0), (2.5; 0.0), (5.0; 0.0), (7.5; 0.0), (10.0; 0.0)}  
ArrayTabulatedFunction(0, 10, values): {(0.0; 1.0), (2.5; 2.0), (5.0; 3.0), (7.5; 4.0), (10.0; 5.0)}
```

Тест с конструкторами LinkedListTabulatedFunction:

```
LinkedListTabulatedFunction(0, 10, 5): {(0.0; 0.0), (2.5; 0.0), (5.0; 0.0), (7.5; 0.0), (10.0; 0.0)}  
LinkedListTabulatedFunction(0, 10, values): {(0.0; 1.0), (2.5; 2.0), (5.0; 3.0), (7.5; 4.0), (10.0; 5.0)}
```

Тест с изменением координаты X (проверка исключения):  
? Корректно выброшено исключение: X координата первой точки должна быть меньше X координаты второй точки  
Объект после попытки некорректного изменения: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2), (3.0; 7.1), (4.0; 9.5)}