Министерство науки и высшего образования Российской

Федерации федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

Институт информатики и кибернетики

Отчет по лабораторной работе № 5

Дисциплина: «ООП»

Тема «Методы класса Object»

Выполнил: Сучков Борис Антонович

Группа: 6201-120303D

Самара 2025

Задание на лабораторную работу

Расширить возможности классов, связанных с табулированными функциями, переопределив в них методы, унаследованные из класса Object.

Задание 1

Переопределите в классе FunctionPoint следующие методы.

- **String toString():** Должен возвращать текстовое описание точки. Например: (1.1; -7.5), где 1.1 и -7.5 – абсцисса и ордината точки соответственно.
- **boolean equals(Object o):** Должен возвращать true тогда и только тогда, когда переданный объект также является точкой и его координаты в точности совпадают с координатами объекта, у которого вызывается метод. Не забудьте обеспечить корректное сравнение чисел с плавающей точкой.
- **int hashCode():** Должен возвращать значение хэш-кода для объекта точки. Можно выбрать реализацию хэш-функции или воспользоваться простейшей реализацией, основанной на применении операции исключающего ИЛИ (XOR). В этом случае хэш-код рассчитывается как побитовое XOR для набора значений типа int. Этот набор должен включать в себя всю информацию, описывающую состояние объекта, т.е. два значения координат. Поскольку они имеют тип double, необходимо без потерь перевести эту информацию к типу int, например, представив одно значение типа double (8 байт) как два значения типа int (4 байта и 4 байта). Сделать это можно с помощью метода Double.doubleToLongBits(), оператора побитового И (&) (для выделения младших четырёх байтов) и оператора битового логического сдвига (>>) (для выделения старших четырёх байтов).
- **Object clone():** Должен возвращать объект-копию для объекта точки. Достаточно простого клонирования, так как точка не имеет ссылок на другие объекты.

Задание 2

Переопределите в классе ArrayTabulatedFunction следующие методы.

- **String toString():** Должен возвращать описание табулированной функции. Например: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}, где в круглых скобках указываются координаты точек.
- **boolean equals(Object o):** Должен возвращать true тогда и только тогда, когда переданный объект также является табулированной функцией (реализует интерфейс TabulatedFunction) и её набор точек в точности совпадает с набором точек функции, у которой вызывается метод. В случае если переданный объект является экземпляром класса ArrayTabulatedFunction, время работы метода должно быть сокращено за счёт прямого обращения к элементам состояния переданного объекта. Не забудьте обеспечить корректное сравнение чисел с плавающей точкой. • **int hashCode():** Должен возвращать значение хэш-кода для объекта табулированной функции. Можно выбрать реализацию хэш-функции или воспользоваться простейшей реализацией, основанной на применении операции исключающего ИЛИ (XOR). В этом случае хэш-код рассчитывается как побитовое XOR для набора значений типа int. В данный набор входят хэш-коды всех точек табулированной функции, а также количество точек в

функции. Последнее нужно для того, чтобы значения хэш-кода были различны для функций, отличающихся наличием нулевой точки (например, {(-1; 1), (0; 0), (1, 1)} и {(-1; 1), (1, 1)}). • **Object clone():** Должен возвращать объект-копию для объекта табулированной функции.
Поскольку табулированная функция ссылается на другие объекты, клонирование должно быть глубоким.

Задание 3

Аналогично, переопределите методы toString(), equals(), hashCode() и clone() в классе LinkedListTabulatedFunction. При написании методов учтите следующие особенности.

- Метод equals() также должен корректно работать при сравнении с любым объектом типа TabulatedFunction, а при сравнении с объектом типа LinkedListTabulatedFunction время работы метода должно быть сокращено за счёт возможности прямого обращения к полям переданного объекта.
- Клонирование в методе clone() тоже должно быть глубоким, однако классическое глубокое клонирование в данном случае не совсем разумно. Если сделать объекты класса FunctionNode клонируемыми, после их клонирования значения полей ссылок придётся изменить (т.к. они будут ссылаться на объекты из исходного списка), и значение ссылающегося на объект точки поля тоже придётся изменить (т.к. его нужно будет заменить клоном объекта точки). Поэтому проще окажется «пересобрать» новый объект списка, причём сделать это проще без использования методов добавления в список, т.к. это приведёт к выполнению большого количества нерезультативных операций и заметно скажется на скорости выполнения программы.

Задание 4

Сделайте так, чтобы все объекты типа TabulatedFunction были клонируемыми с точки зрения JVM и внесите метод clone() в этот интерфейс.

Задание 5

Проверьте работу написанных методов.

- Проверьте работу метода toString() для объектов типов ArrayTabulatedFunction и LinkedListTabulatedFunction, выведя строковое представление объектов в консоль.
- Проверьте работу метода equals(), вызывая его для одинаковых и различающихся объектов одинаковых и различающихся классов.
- Проверьте работу метода hashCode(), выведя в консоль его значения для всех использованных объектов. Убедитесь в согласованности работы методов equals() и hashCode(). Также попробуйте незначительно изменить один из объектов (например, изменить одну из координат одной из точек на несколько тысячных) и проверьте, как изменится значение хэш-кода объекта.
- Проверьте работу метода clone() для объектов обоих классов табулированных функций. Убедитесь, что произведено именно глубокое клонирование: для этого после

клонирования измените исходные объекты и проверьте, что объекты-клоны не изменились.

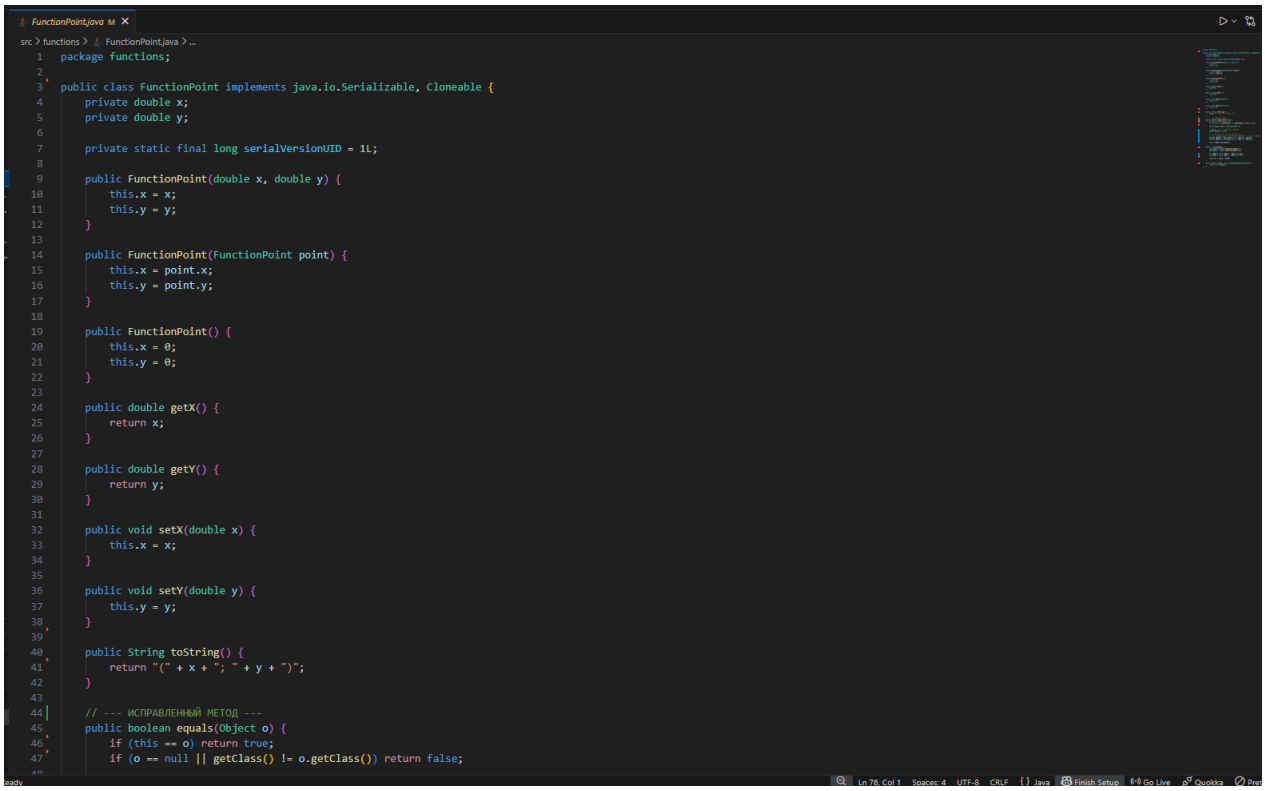
Задание 1

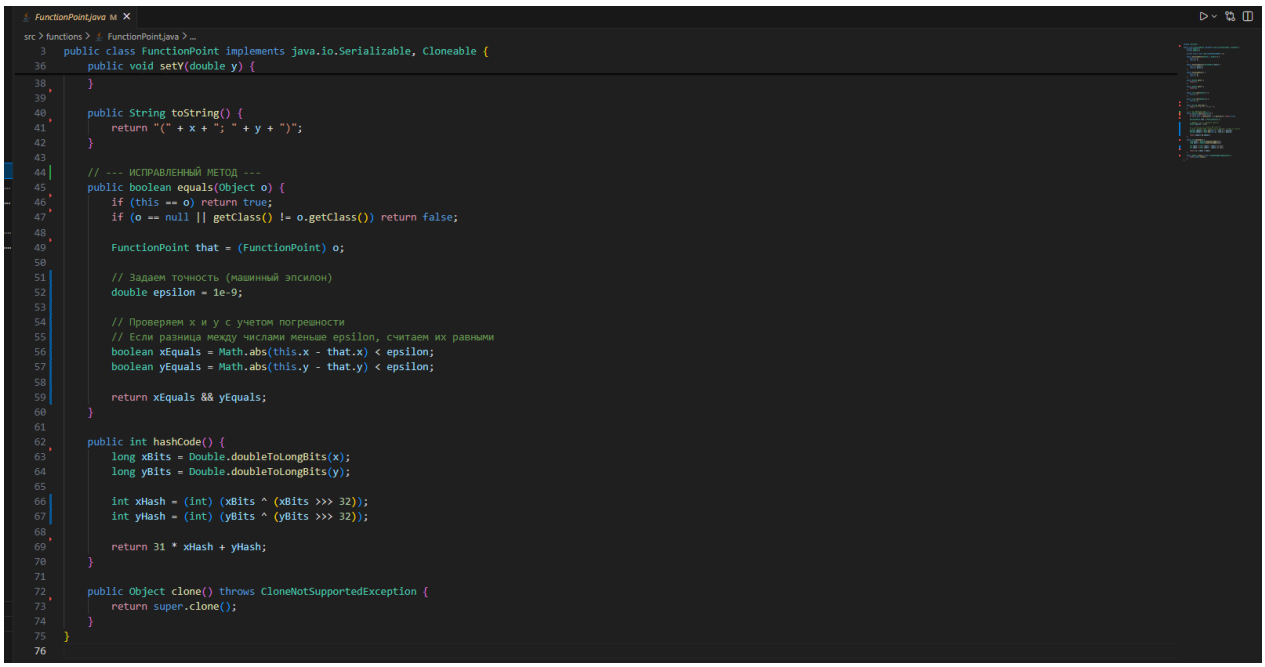
Я переопределил в классе FunctionPoint следующие методы.

• **String toString():** Возвращает текстовое описание точки. Например: (1.1; -7.5), где 1.1 и -7.5 – абсцисса и ордината точки соответственно.

• **boolean equals(Object o):** Возвращает true тогда и только тогда, когда переданный объект также является точкой и его координаты в точности совпадают с координатами объекта, у которого вызывается метод. Я обеспечил корректное сравнение чисел с плавающей точкой.

• **int hashCode():** Возвращает значение хэш-кода для объекта точки. • **Object clone():** Возвращает объект-копию для объекта точки. Достаточно простого клонирования, так как точка не имеет ссылок на другие объекты.



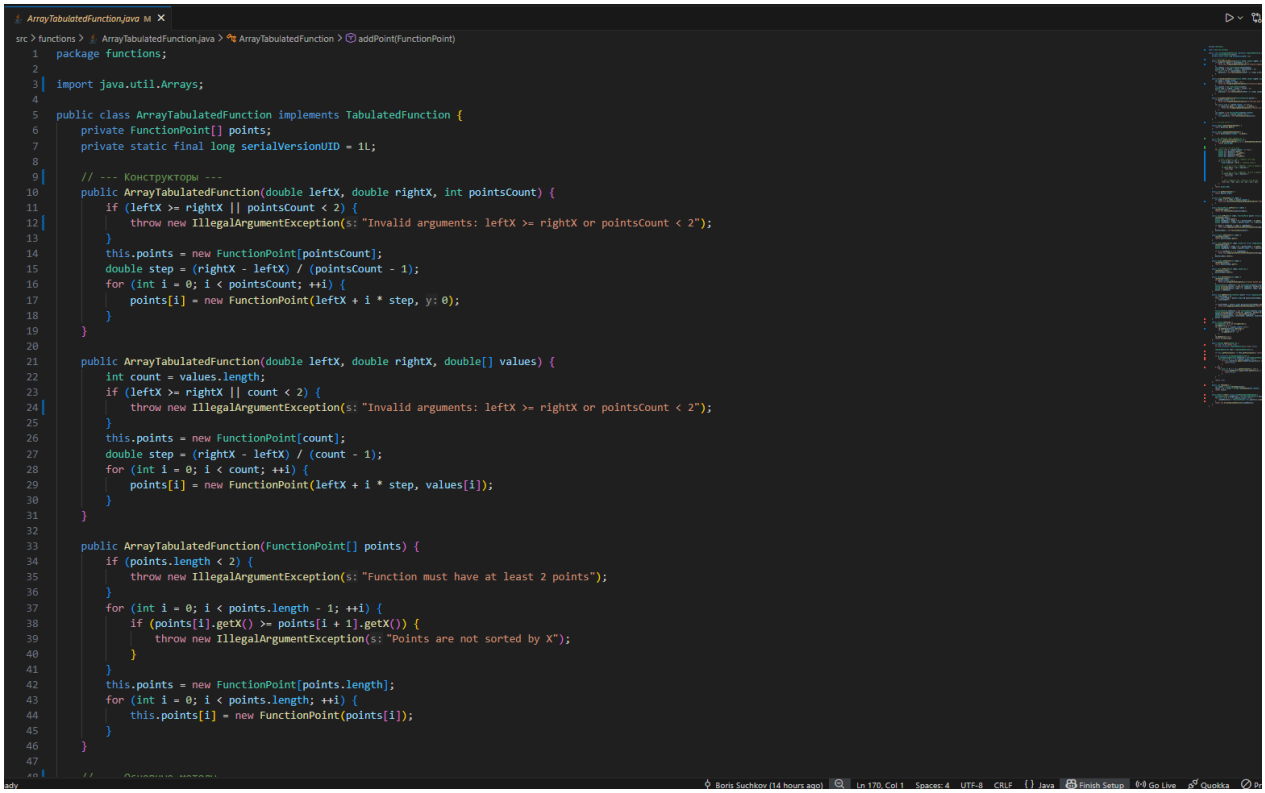


```java
public class FunctionPoint implements java.io.Serializable, Cloneable {
    public void setY(double y) {
    }

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

    // --- ИСПРАВЛЕННЫЙ МЕТОД ---
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        FunctionPoint that = (FunctionPoint) o;

        // Задаем точность (машинный эпсилон)
        double epsilon = 1e-9;

        // Проверяем x и y с учетом погрешности
        // Если разница между числами меньше epsilon, считаем их равными
        boolean xEquals = Math.abs(this.x - that.x) < epsilon;
        boolean yEquals = Math.abs(this.y - that.y) < epsilon;

        return xEquals && yEquals;
    }

    public int hashCode() {
        long xBits = Double.doubleToLongBits(x);
        long yBits = Double.doubleToLongBits(y);

        int xHash = (int) (xBits ^ (xBits >>> 32));
        int yHash = (int) (yBits ^ (yBits >>> 32));

        return 31 * xHash + yHash;
    }

    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

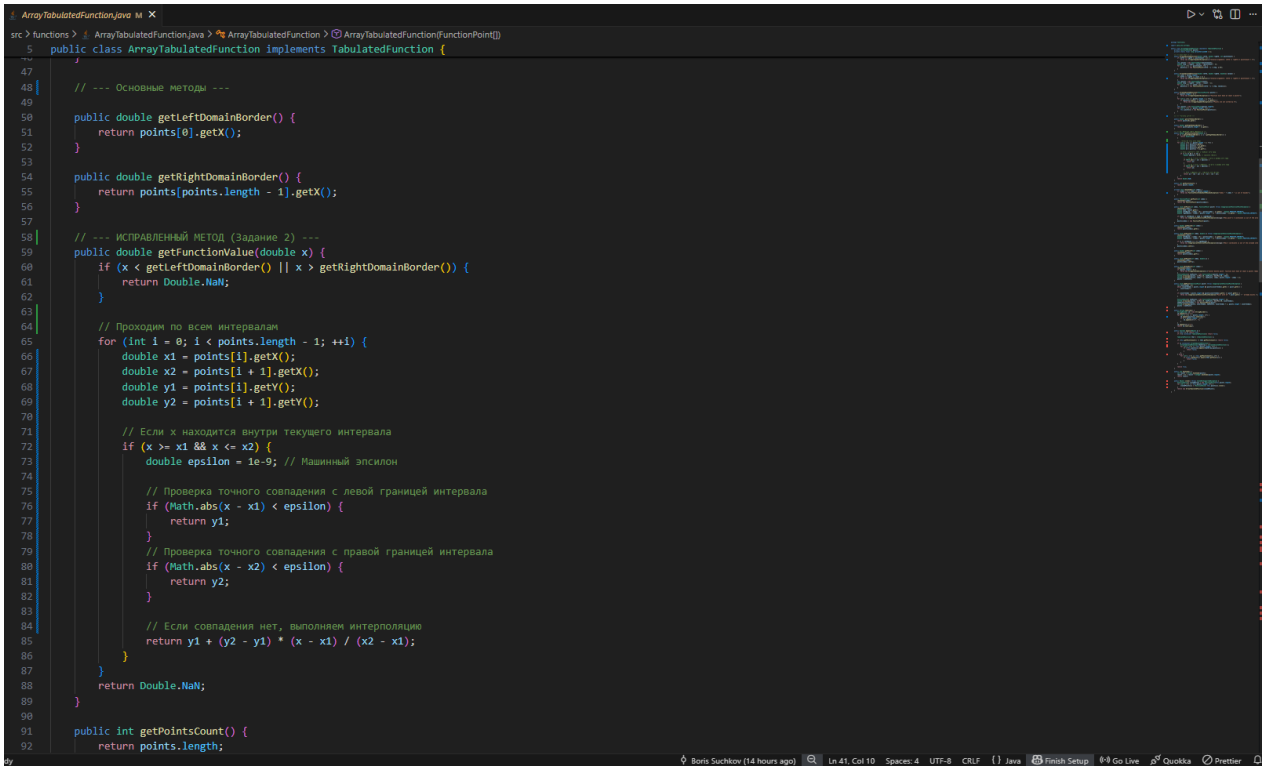
Задание 2

Я переопределил в классе ArrayTabulatedFunction следующие методы.

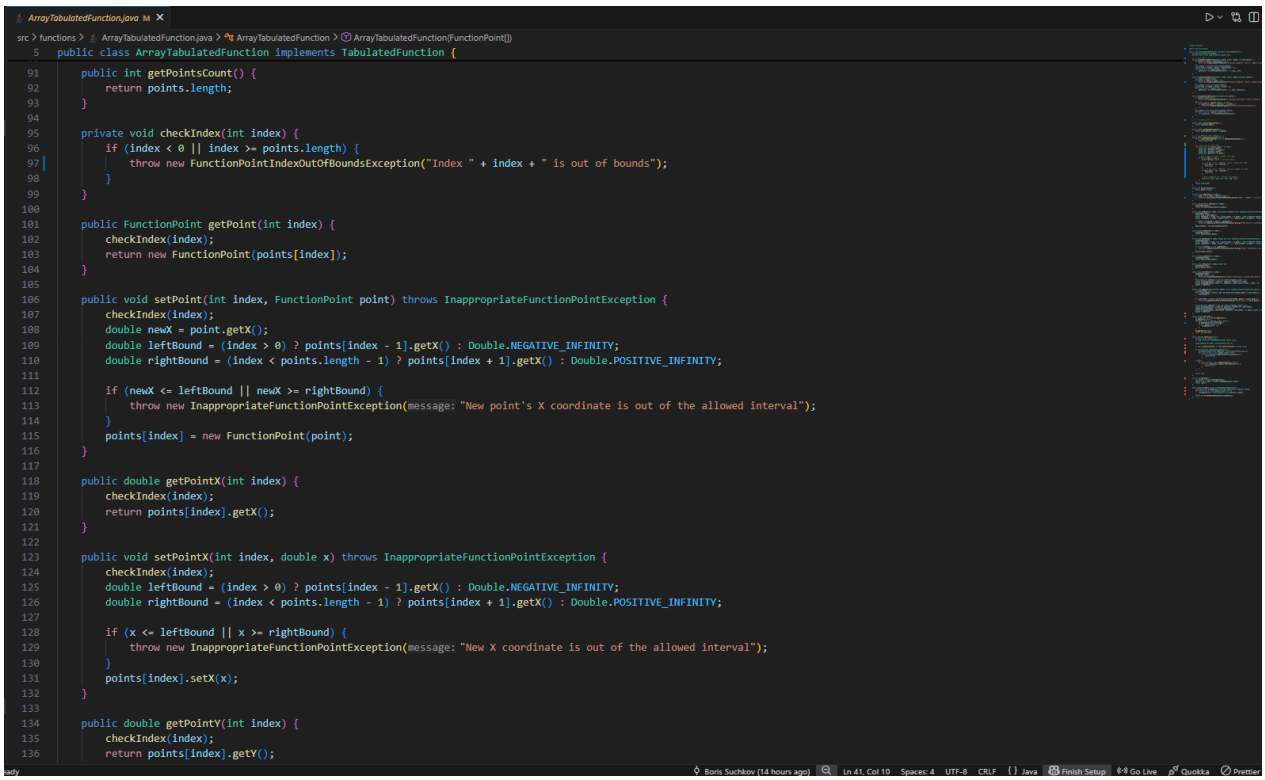
- **String toString():** Возвращает описание табулированной функции. Например: {(0.0; 1.2), (1.0; 3.8), (2.0; 15.2)}, где в круглых скобках указываются координаты точек.
- **boolean equals(Object o):** Возвращает true тогда и только тогда, когда переданный объект также является табулированной функцией (реализует интерфейс TabulatedFunction) и её набор точек в точности совпадает с набором точек функции, у которой вызывается метод. В случае если переданный объект является экземпляром класса ArrayTabulatedFunction, время работы метода сокращено за счёт прямого обращения к элементам состояния переданного объекта. Я обеспечил корректное сравнение чисел с плавающей точкой. • **int hashCode():** Возвращает значение хэш-кода для объекта табулированной функции.
- **Object clone():** Возвращает объект-копию для объекта табулированной функции. Поскольку табулированная функция ссылается на другие объекты, клонирование глубокое.



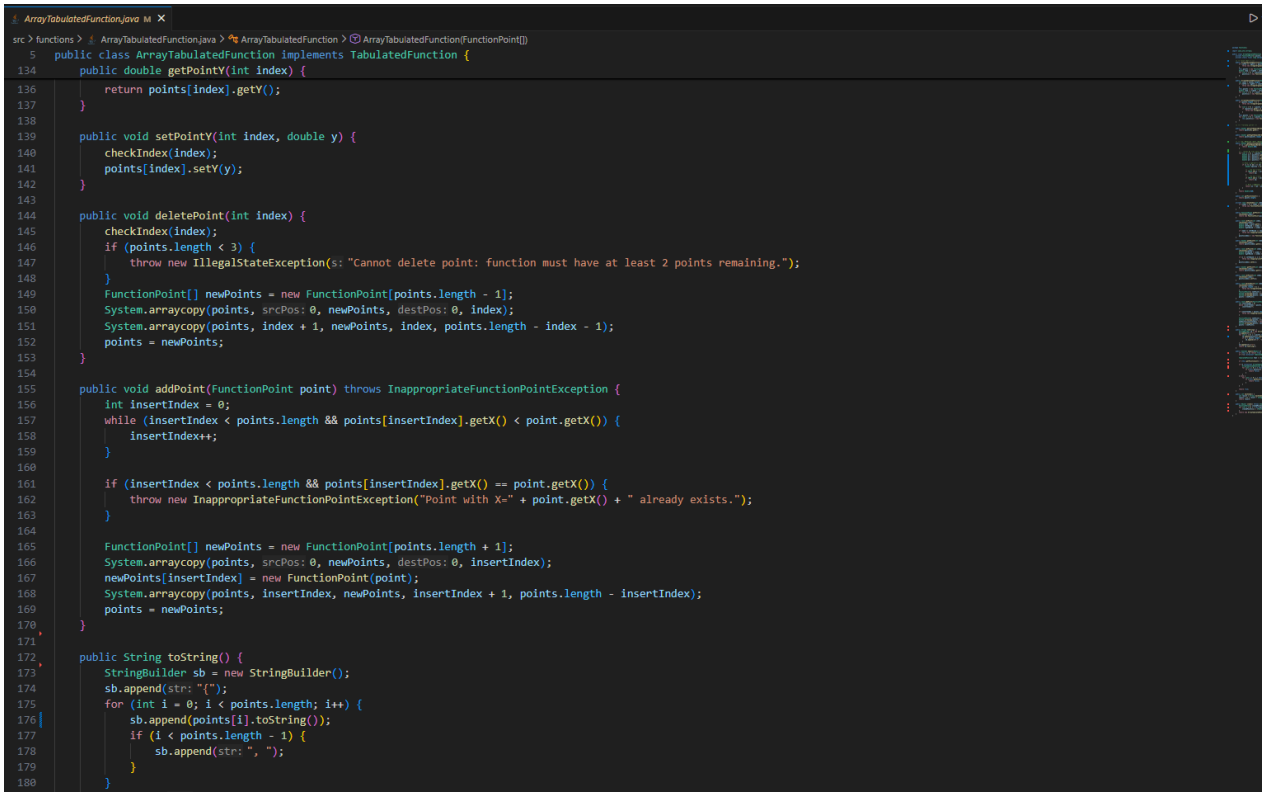
```java
package functions;

import java.util.Arrays;

public class ArrayTabulatedFunction implements TabulatedFunction {
    private FunctionPoint[] points;
    private static final long serialVersionUID = 1L;

    // --- Конструкторы ---
    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX || pointsCount < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments: leftX >= rightX or pointsCount < 2");
        }
        this.points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; ++i) {
            points[i] = new FunctionPoint(leftX + i * step, y: 0);
        }
    }

    public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
        int count = values.length;
        if (leftX >= rightX || count < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments: leftX >= rightX or pointsCount < 2");
        }
        this.points = new FunctionPoint[count];
        double step = (rightX - leftX) / (count - 1);
        for (int i = 0; i < count; ++i) {
            points[i] = new FunctionPoint(leftX + i * step, values[i]);
        }
    }

    public ArrayTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException(s: "Function must have at least 2 points");
        }
        for (int i = 0; i < points.length - 1; ++i) {
            if (points[i].getX() >= points[i + 1].getX()) {
                throw new IllegalArgumentException(s: "Points are not sorted by X");
            }
        }
        this.points = new FunctionPoint[points.length];
        for (int i = 0; i < points.length; ++i) {
            this.points[i] = new FunctionPoint(points[i]);
        }
    }
```



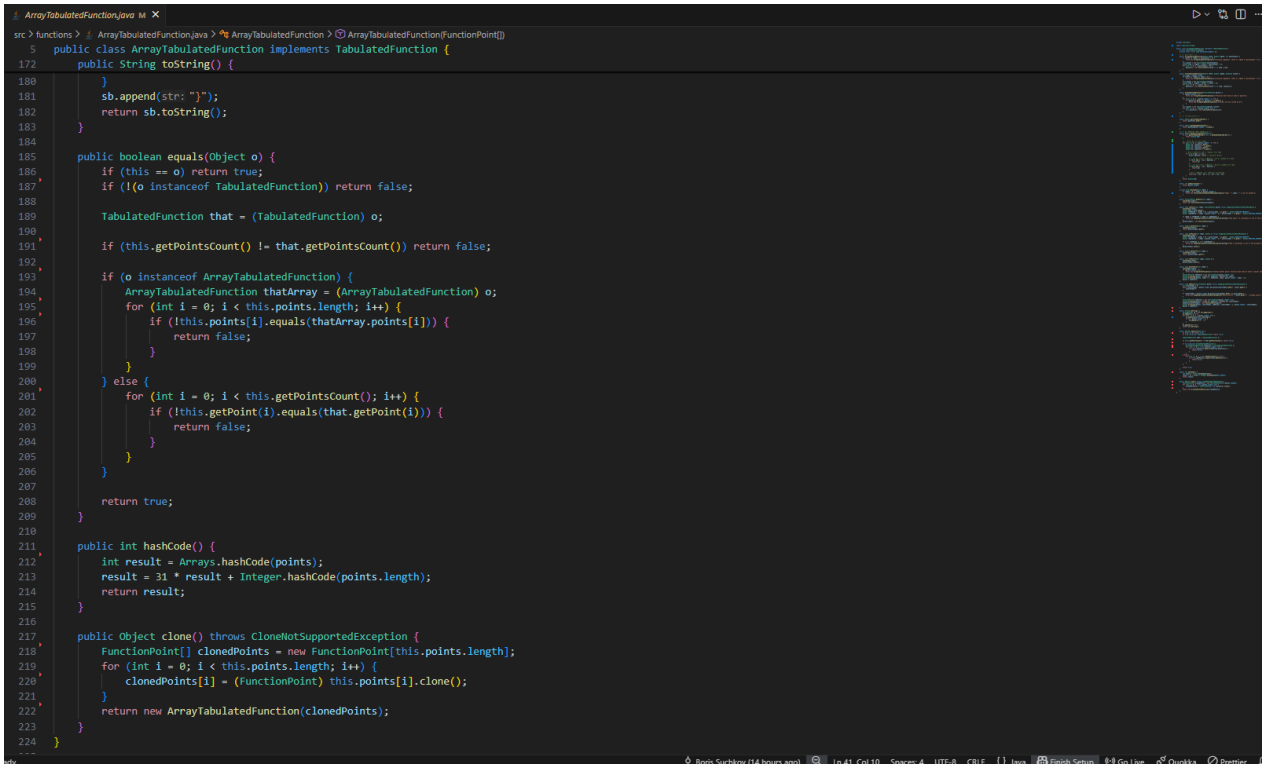
```java
public class ArrayTabulatedFunction implements TabulatedFunction {

    // --- Основные методы ---

    public double getLeftDomainBorder() {
        return points[0].getX();
    }

    public double getRightDomainBorder() {
        return points[points.length - 1].getX();
    }

    // --- ИСПРАВЛЕННЫЙ МЕТОД (Задание 2) ---
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }

        // Проходим по всем интервалам
        for (int i = 0; i < points.length - 1; ++i) {
            double x1 = points[i].getX();
            double x2 = points[i + 1].getX();
            double y1 = points[i].getY();
            double y2 = points[i + 1].getY();

            // Если x находится внутри текущего интервала
            if (x >= x1 && x <= x2) {
                double epsilon = 1e-9; // Машинный эпсилон

                // Проверка точного совпадения с левой границей интервала
                if (Math.abs(x - x1) < epsilon) {
                    return y1;
                }
                // Проверка точного совпадения с правой границей интервала
                if (Math.abs(x - x2) < epsilon) {
                    return y2;
                }

                // Если совпадения нет, выполняем интерполяцию
                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
            }
        }
        return Double.NaN;
    }

    public int getPointsCount() {
        return points.length;
```



```
ArrayTabulatedFunction.java M
src > functions > ArrayTabulatedFunction.java > ArrayTabulatedFunction > ArrayTabulatedFunction(FunctionPoint[])
public class ArrayTabulatedFunction implements TabulatedFunction {
    public int getPointsCount() {
        return points.length;
    }

    private void checkIndex(int index) {
        if (index < 0 || index >= points.length) {
            throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds");
        }
    }

    public FunctionPoint getPoint(int index) {
        checkIndex(index);
        return new FunctionPoint(points[index]);
    }

    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
        checkIndex(index);
        double newX = point.getX();
        double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
        double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;

        if (newX <= leftBound || newX >= rightBound) {
            throw new InappropriateFunctionPointException(message: "New point's X coordinate is out of the allowed interval");
        }
        points[index] = new FunctionPoint(point);
    }

    public double getPointX(int index) {
        checkIndex(index);
        return points[index].getX();
    }

    public void setPointX(int index, double x) throws InappropriateFunctionPointException {
        checkIndex(index);
        double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
        double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;

        if (x <= leftBound || x >= rightBound) {
            throw new InappropriateFunctionPointException(message: "New X coordinate is out of the allowed interval");
        }
        points[index].setX(x);
    }

    public double getPointY(int index) {
        checkIndex(index);
        return points[index].getY();
```



```
ArrayTabulatedFunction.java M
src > functions > ArrayTabulatedFunction.java > ArrayTabulatedFunction > ArrayTabulatedFunction(FunctionPoint[])
public class ArrayTabulatedFunction implements TabulatedFunction {
    public double getPointY(int index) {
        return points[index].getY();
    }

    public void setPointY(int index, double y) {
        checkIndex(index);
        points[index].setY(y);
    }

    public void deletePoint(int index) {
        checkIndex(index);
        if (points.length < 3) {
            throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
        }
        FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
        System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, index);
        System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
        points = newPoints;
    }

    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
        int insertIndex = 0;
        while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) {
            insertIndex++;
        }

        if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) {
            throw new InappropriateFunctionPointException("Point with X=" + point.getX() + " already exists.");
        }

        FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
        System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, insertIndex);
        newPoints[insertIndex] = new FunctionPoint(point);
        System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
        points = newPoints;
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(str: "{");
        for (int i = 0; i < points.length; i++) {
            sb.append(points[i].toString());
            if (i < points.length - 1) {
                sb.append(str: ", ");
            }
        }
```



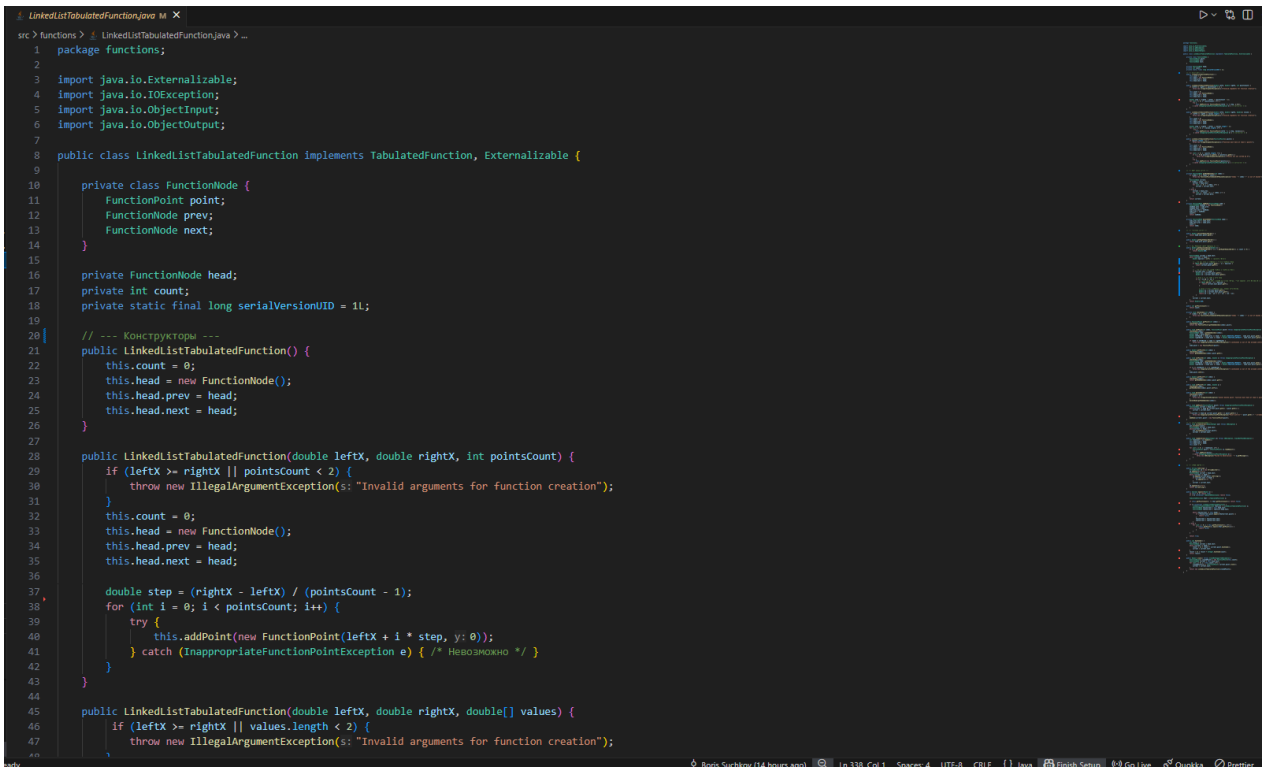
```
ArrayTabulatedFunction.java M
src > functions > ArrayTabulatedFunction.java > ArrayTabulatedFunction > ArrayTabulatedFunction(FunctionPoint[])
  5  public class ArrayTabulatedFunction implements TabulatedFunction {
172      public String toString() {
180          }
181          sb.append(str: "}");
182          return sb.toString();
183      }
184
185      public boolean equals(Object o) {
186          if (this == o) return true;
187          if (!(o instanceof TabulatedFunction)) return false;
188
189          TabulatedFunction that = (TabulatedFunction) o;
190
191          if (this.getPointsCount() != that.getPointsCount()) return false;
192
193          if (o instanceof ArrayTabulatedFunction) {
194              ArrayTabulatedFunction thatArray = (ArrayTabulatedFunction) o;
195              for (int i = 0; i < this.points.length; i++) {
196                  if (!this.points[i].equals(thatArray.points[i])) {
197                      return false;
198                  }
199              }
200          } else {
201              for (int i = 0; i < this.getPointsCount(); i++) {
202                  if (!this.getPoint(i).equals(that.getPoint(i))) {
203                      return false;
204                  }
205              }
206          }
207
208          return true;
209      }
210
211      public int hashCode() {
212          int result = Arrays.hashCode(points);
213          result = 31 * result + Integer.hashCode(points.length);
214          return result;
215      }
216
217      public Object clone() throws CloneNotSupportedException {
218          FunctionPoint[] clonedPoints = new FunctionPoint[this.points.length];
219          for (int i = 0; i < this.points.length; i++) {
220              clonedPoints[i] = (FunctionPoint) this.points[i].clone();
221          }
222          return new ArrayTabulatedFunction(clonedPoints);
223      }
224  }
```

```
215      }
216
217      public Object clone() throws CloneNotSupportedException {
218          FunctionPoint[] clonedPoints = new FunctionPoint[this.points.length];
219          for (int i = 0; i < this.points.length; i++) {
220              clonedPoints[i] = (FunctionPoint) this.points[i].clone();
221          }
222          return new ArrayTabulatedFunction(clonedPoints);
223      }
224  }
225
```

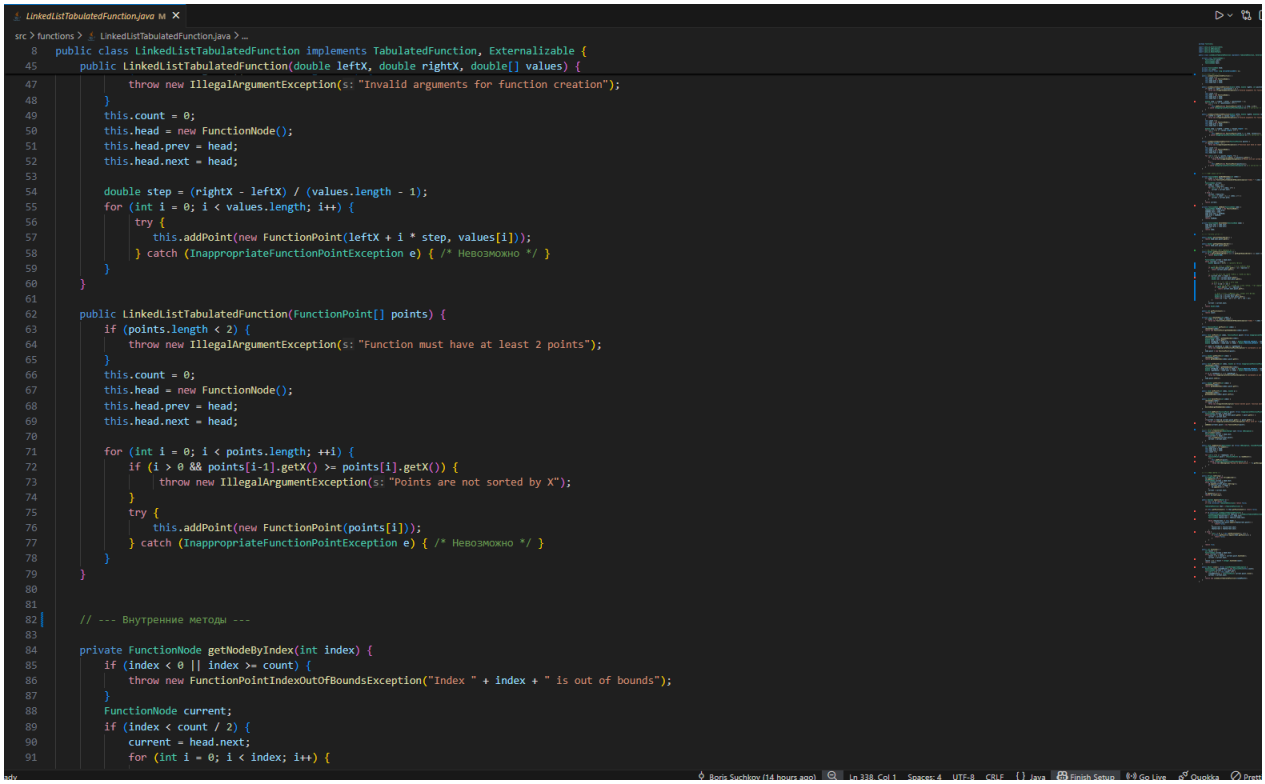
Задание 3

Аналогично, я переопределил методы toString(), equals(), hashCode() и clone() в классе LinkedListTabulatedFunction. При написании методов я учёл следующие особенности.

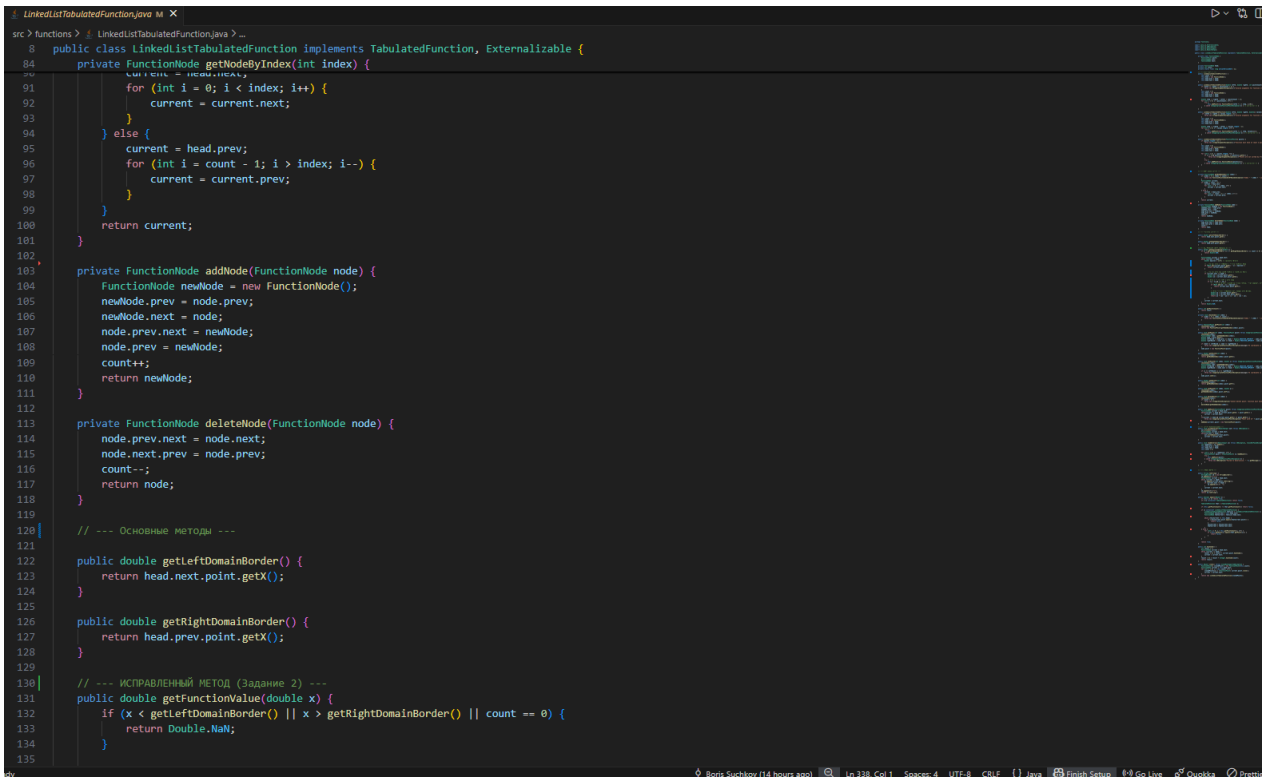
- Метод equals() также корректно работает при сравнении с любым объектом типа TabulatedFunction, а при сравнении с объектом типа LinkedListTabulatedFunction время работы метода сокращено за счёт возможности прямого обращения к полям переданного объекта.
- Клонирование в методе clone() тоже глубокое, однако классическое глубокое клонирование в данном случае не совсем разумно. Если сделать объекты класса FunctionNode клонируемыми, после их клонирования значения полей ссылок придётся изменить (т.к. они будут ссылаться на объекты из исходного списка), и значение ссылающегося на объект точки поля тоже придётся изменить (т.к. его нужно будет заменить клоном объекта точки). Поэтому проще окажется «пересобрать» новый объект списка, причём сделать это проще без использования методов добавления в список, т.к. это приведёт к выполнению большого количества нерезультативных операций и заметно скажется на скорости выполнения программы.



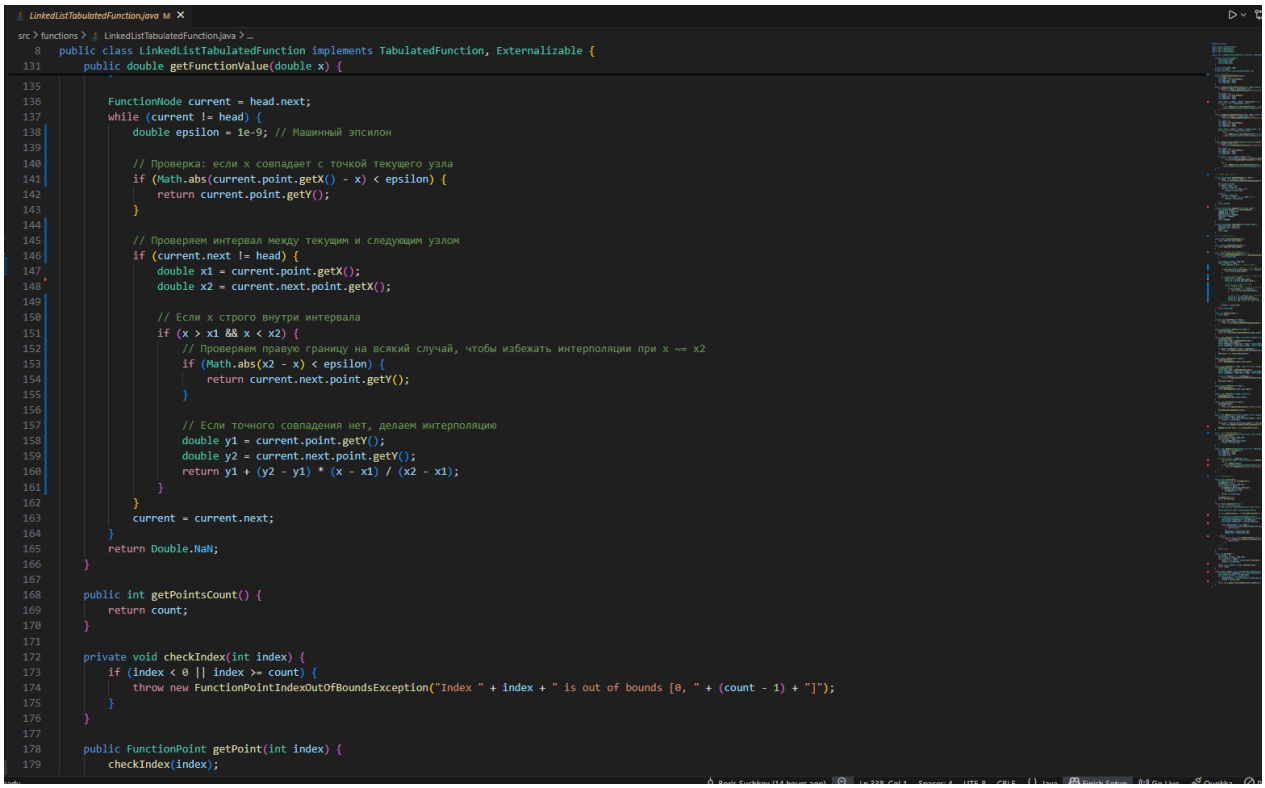
```java
package functions;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {

    private class FunctionNode {
        FunctionPoint point;
        FunctionNode prev;
        FunctionNode next;
    }

    private FunctionNode head;
    private int count;
    private static final long serialVersionUID = 1L;

    // --- Конструкторы ---
    public LinkedListTabulatedFunction() {
        this.count = 0;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;
    }

    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX || pointsCount < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
        }
        this.count = 0;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            try {
                this.addPoint(new FunctionPoint(leftX + i * step, y: 0));
            } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
        }
    }

    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
        if (leftX >= rightX || values.length < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
```



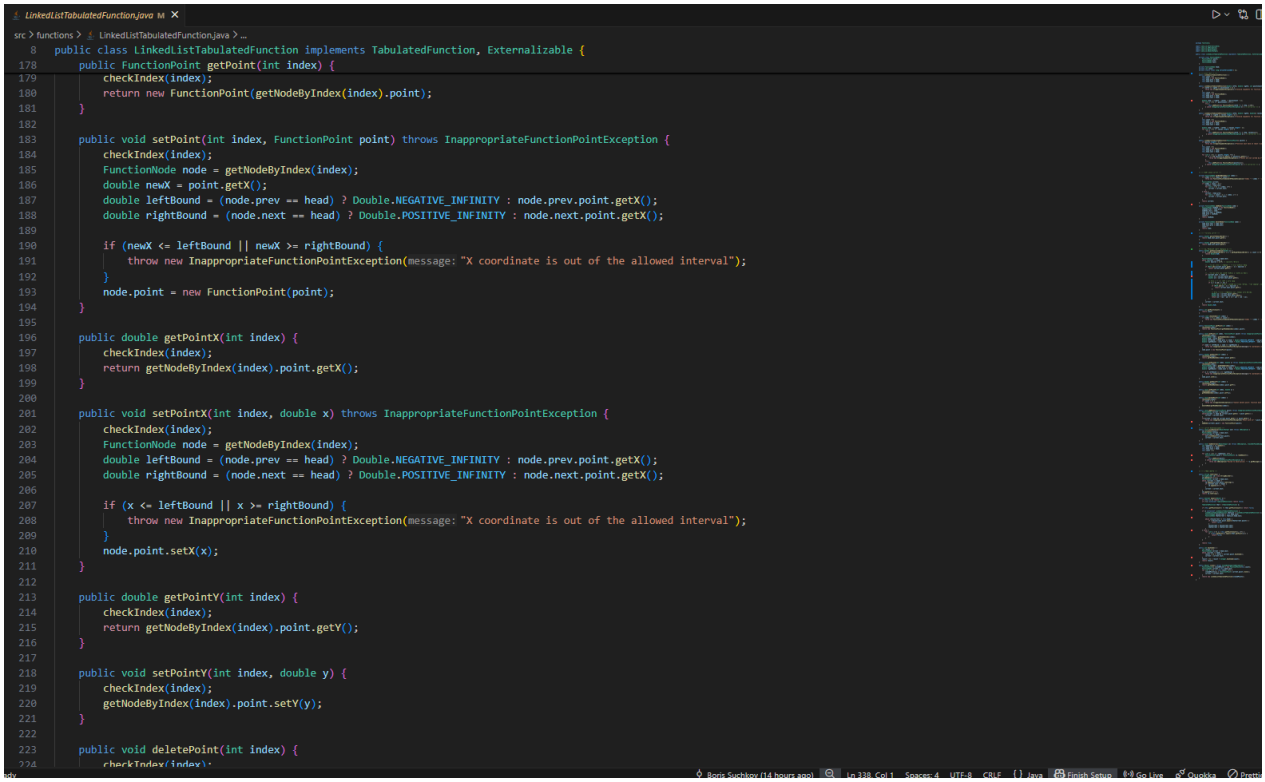
```java
public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
        }
        this.count = 0;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            try {
                this.addPoint(new FunctionPoint(leftX + i * step, values[i]));
            } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
        }
    }

    public LinkedListTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException(s: "Function must have at least 2 points");
        }
        this.count = 0;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        for (int i = 0; i < points.length; ++i) {
            if (i > 0 && points[i-1].getX() >= points[i].getX()) {
                throw new IllegalArgumentException(s: "Points are not sorted by X");
            }
            try {
                this.addPoint(new FunctionPoint(points[i]));
            } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
        }
    }


    // --- Внутренние методы ---

    private FunctionNode getNodeByIndex(int index) {
        if (index < 0 || index >= count) {
            throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds");
        }
        FunctionNode current;
        if (index < count / 2) {
            current = head.next;
            for (int i = 0; i < index; i++) {
```



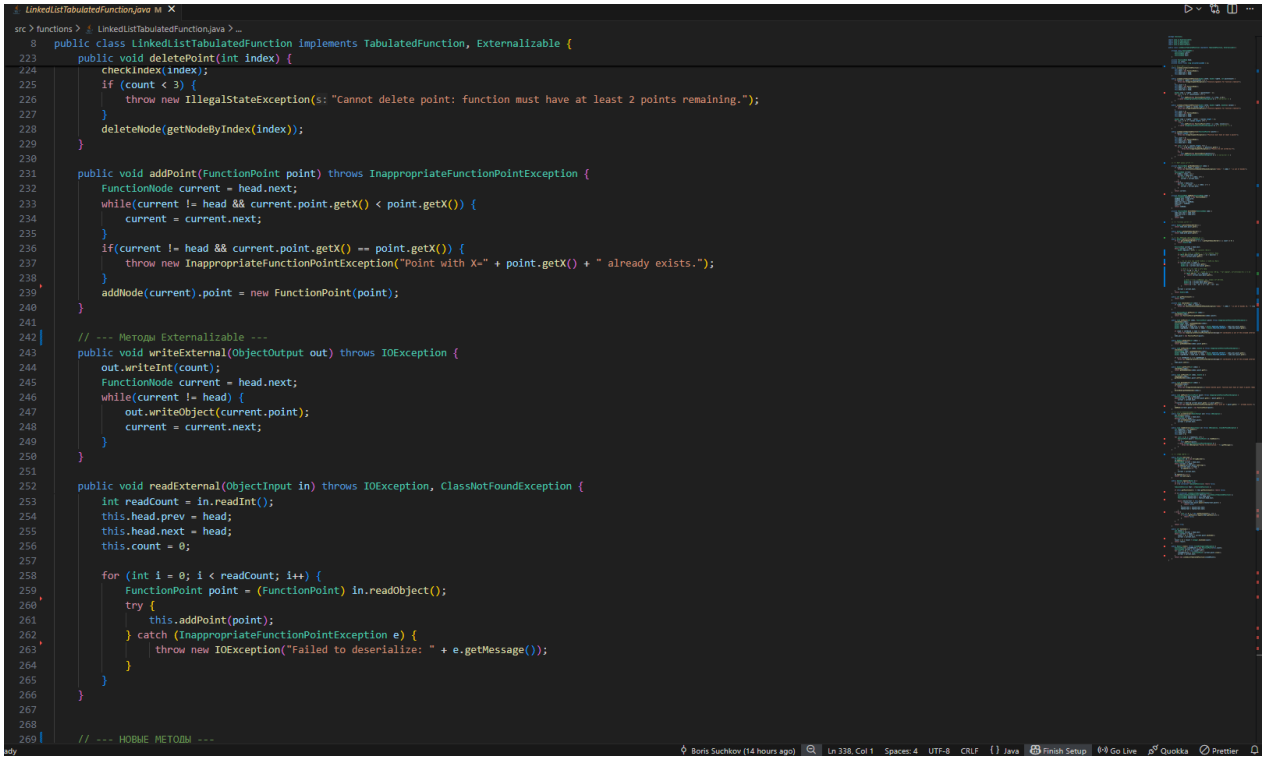
```
LinkedListTabulatedFunction.java M
src > functions > LinkedListTabulatedFunction.java > ...
  8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
 84      private FunctionNode getNodeByIndex(int index) {
 91              for (int i = 0; i < index; i++) {
 92                  current = current.next;
 93              }
 94          } else {
 95              current = head.prev;
 96              for (int i = count - 1; i > index; i--) {
 97                  current = current.prev;
 98              }
 99          }
100          return current;
101      }
102
103      private FunctionNode addNode(FunctionNode node) {
104          FunctionNode newNode = new FunctionNode();
105          newNode.prev = node.prev;
106          newNode.next = node;
107          node.prev.next = newNode;
108          node.prev = newNode;
109          count++;
110          return newNode;
111      }
112
113      private FunctionNode deleteNode(FunctionNode node) {
114          node.prev.next = node.next;
115          node.next.prev = node.prev;
116          count--;
117          return node;
118      }
119
120      // --- Основные методы ---
121
122      public double getLeftDomainBorder() {
123          return head.next.point.getX();
124      }
125
126      public double getRightDomainBorder() {
127          return head.prev.point.getX();
128      }
129
130      // --- ИСПРАВЛЕННЫЙ МЕТОД (Задание 2) ---
131      public double getFunctionValue(double x) {
132          if (x < getLeftDomainBorder() || x > getRightDomainBorder() || count == 0) {
133              return Double.NaN;
134          }
135
```



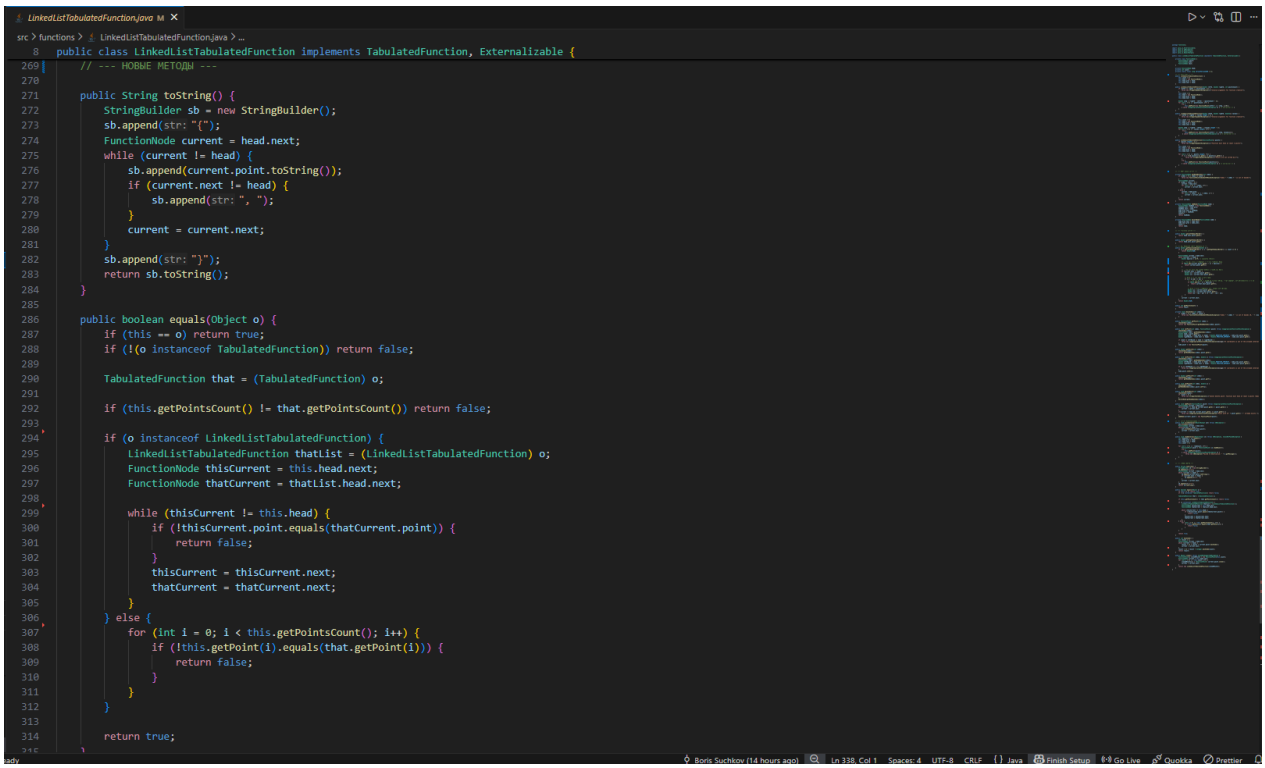
```
LinkedListTabulatedFunction.java M
src > functions > LinkedListTabulatedFunction.java > ...
  8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
131      public double getFunctionValue(double x) {
135
136          FunctionNode current = head.next;
137          while (current != head) {
138              double epsilon = 1e-9; // Машинный эпсилон
139
140              // Проверка: если x совпадает с точкой текущего узла
141              if (Math.abs(current.point.getX() - x) < epsilon) {
142                  return current.point.getY();
143              }
144
145              // Проверяем интервал между текущим и следующим узлом
146              if (current.next != head) {
147                  double x1 = current.point.getX();
148                  double x2 = current.next.point.getX();
149
150                  // Если x строго внутри интервала
151                  if (x > x1 && x < x2) {
152                      // Проверяем правую границу на всякий случай, чтобы избежать интерполяции при x ~= x2
153                      if (Math.abs(x2 - x) < epsilon) {
154                          return current.next.point.getY();
155                      }
156
157                      // Если точного совпадения нет, делаем интерполяцию
158                      double y1 = current.point.getY();
159                      double y2 = current.next.point.getY();
160                      return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
161                  }
162              }
163              current = current.next;
164          }
165          return Double.NaN;
166      }
167
168      public int getPointsCount() {
169          return count;
170      }
171
172      private void checkIndex(int index) {
173          if (index < 0 || index >= count) {
174              throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds [0, " + (count - 1) + "]");
175          }
176      }
177
178      public FunctionPoint getPoint(int index) {
179          checkIndex(index);
```



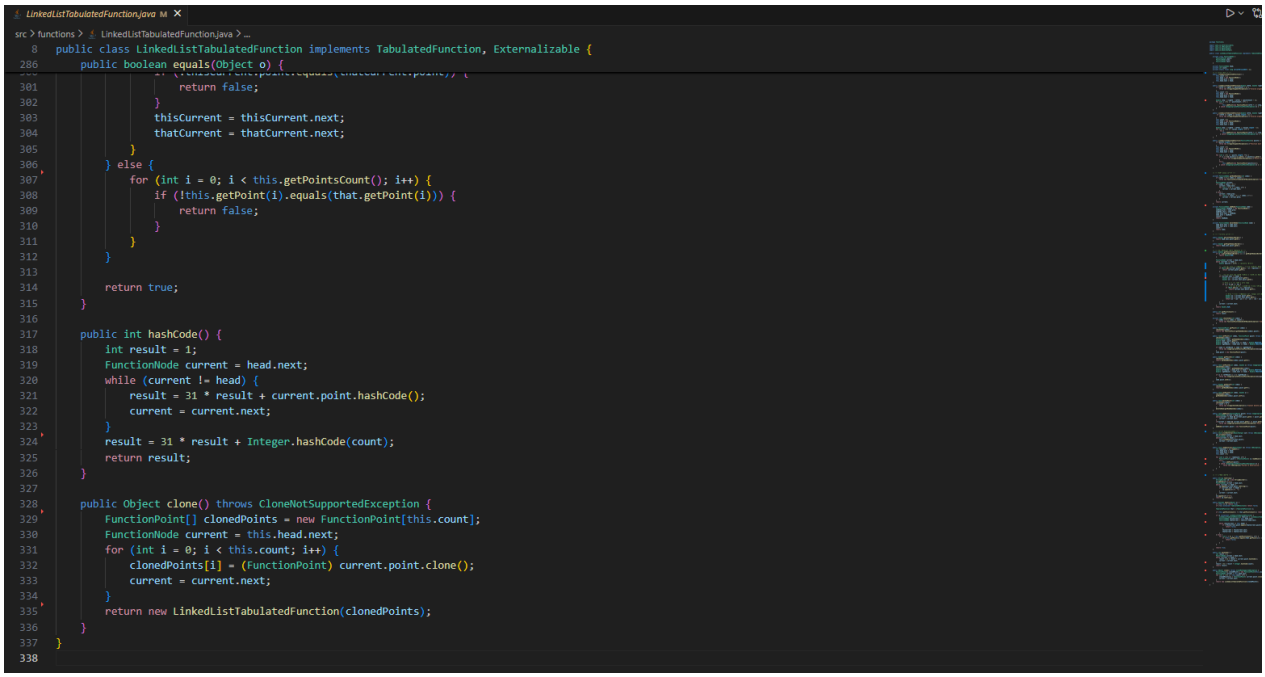
```
LinkedListTabulatedFunction.java M
src > functions > LinkedListTabulatedFunction.java > ...
  8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
178      public FunctionPoint getPoint(int index) {
179          checkIndex(index);
180          return new FunctionPoint(getNodeByIndex(index).point);
181      }
182
183      public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
184          checkIndex(index);
185          FunctionNode node = getNodeByIndex(index);
186          double newX = point.getX();
187          double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
188          double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
189
190          if (newX <= leftBound || newX >= rightBound) {
191              throw new InappropriateFunctionPointException(message: "X coordinate is out of the allowed interval");
192          }
193          node.point = new FunctionPoint(point);
194      }
195
196      public double getPointX(int index) {
197          checkIndex(index);
198          return getNodeByIndex(index).point.getX();
199      }
200
201      public void setPointX(int index, double x) throws InappropriateFunctionPointException {
202          checkIndex(index);
203          FunctionNode node = getNodeByIndex(index);
204          double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
205          double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
206
207          if (x <= leftBound || x >= rightBound) {
208              throw new InappropriateFunctionPointException(message: "X coordinate is out of the allowed interval");
209          }
210          node.point.setX(x);
211      }
212
213      public double getPointY(int index) {
214          checkIndex(index);
215          return getNodeByIndex(index).point.getY();
216      }
217
218      public void setPointY(int index, double y) {
219          checkIndex(index);
220          getNodeByIndex(index).point.setY(y);
221      }
222
223      public void deletePoint(int index) {
224          checkIndex(index);
```



```
LinkedListTabulatedFunction.java M
src > functions > LinkedListTabulatedFunction.java > ...
  8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
223      public void deletePoint(int index) {
224          checkIndex(index);
225          if (count < 3) {
226              throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
227          }
228          deleteNode(getNodeByIndex(index));
229      }
230
231      public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
232          FunctionNode current = head.next;
233          while(current != head && current.point.getX() < point.getX()) {
234              current = current.next;
235          }
236          if(current != head && current.point.getX() == point.getX()) {
237              throw new InappropriateFunctionPointException("Point with X=" + point.getX() + " already exists.");
238          }
239          addNode(current).point = new FunctionPoint(point);
240      }
241
242      // --- Методы Externalizable ---
243      public void writeExternal(ObjectOutput out) throws IOException {
244          out.writeInt(count);
245          FunctionNode current = head.next;
246          while(current != head) {
247              out.writeObject(current.point);
248              current = current.next;
249          }
250      }
251
252      public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
253          int readCount = in.readInt();
254          this.head.prev = head;
255          this.head.next = head;
256          this.count = 0;
257
258          for (int i = 0; i < readCount; i++) {
259              FunctionPoint point = (FunctionPoint) in.readObject();
260              try {
261                  this.addPoint(point);
262              } catch (InappropriateFunctionPointException e) {
263                  throw new IOException("Failed to deserialize: " + e.getMessage());
264              }
265          }
266      }
267
268
269      // --- НОВЫЕ МЕТОДЫ ---
```



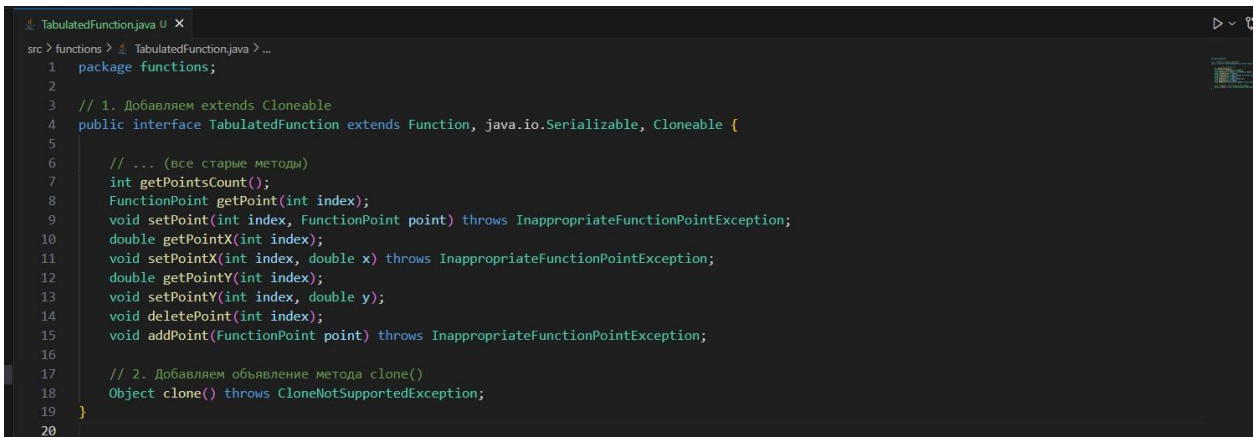
```java
public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
    // --- НОВЫЕ МЕТОДЫ ---

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

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

        TabulatedFunction that = (TabulatedFunction) o;

        if (this.getPointsCount() != that.getPointsCount()) return false;

        if (o instanceof LinkedListTabulatedFunction) {
            LinkedListTabulatedFunction thatList = (LinkedListTabulatedFunction) o;
            FunctionNode thisCurrent = this.head.next;
            FunctionNode thatCurrent = thatList.head.next;

            while (thisCurrent != this.head) {
                if (!thisCurrent.point.equals(thatCurrent.point)) {
                    return false;
                }
                thisCurrent = thisCurrent.next;
                thatCurrent = thatCurrent.next;
            }
        } else {
            for (int i = 0; i < this.getPointsCount(); i++) {
                if (!this.getPoint(i).equals(that.getPoint(i))) {
                    return false;
                }
            }
        }

        return true;
    }
```



```java
public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
    public boolean equals(Object o) {
                    return false;
                }
                thisCurrent = thisCurrent.next;
                thatCurrent = thatCurrent.next;
            }
        } else {
            for (int i = 0; i < this.getPointsCount(); i++) {
                if (!this.getPoint(i).equals(that.getPoint(i))) {
                    return false;
                }
            }
        }

        return true;
    }

    public int hashCode() {
        int result = 1;
        FunctionNode current = head.next;
        while (current != head) {
            result = 31 * result + current.point.hashCode();
            current = current.next;
        }
        result = 31 * result + Integer.hashCode(count);
        return result;
    }

    public Object clone() throws CloneNotSupportedException {
        FunctionPoint[] clonedPoints = new FunctionPoint[this.count];
        FunctionNode current = this.head.next;
        for (int i = 0; i < this.count; i++) {
            clonedPoints[i] = (FunctionPoint) current.point.clone();
            current = current.next;
        }
        return new LinkedListTabulatedFunction(clonedPoints);
    }
}
```

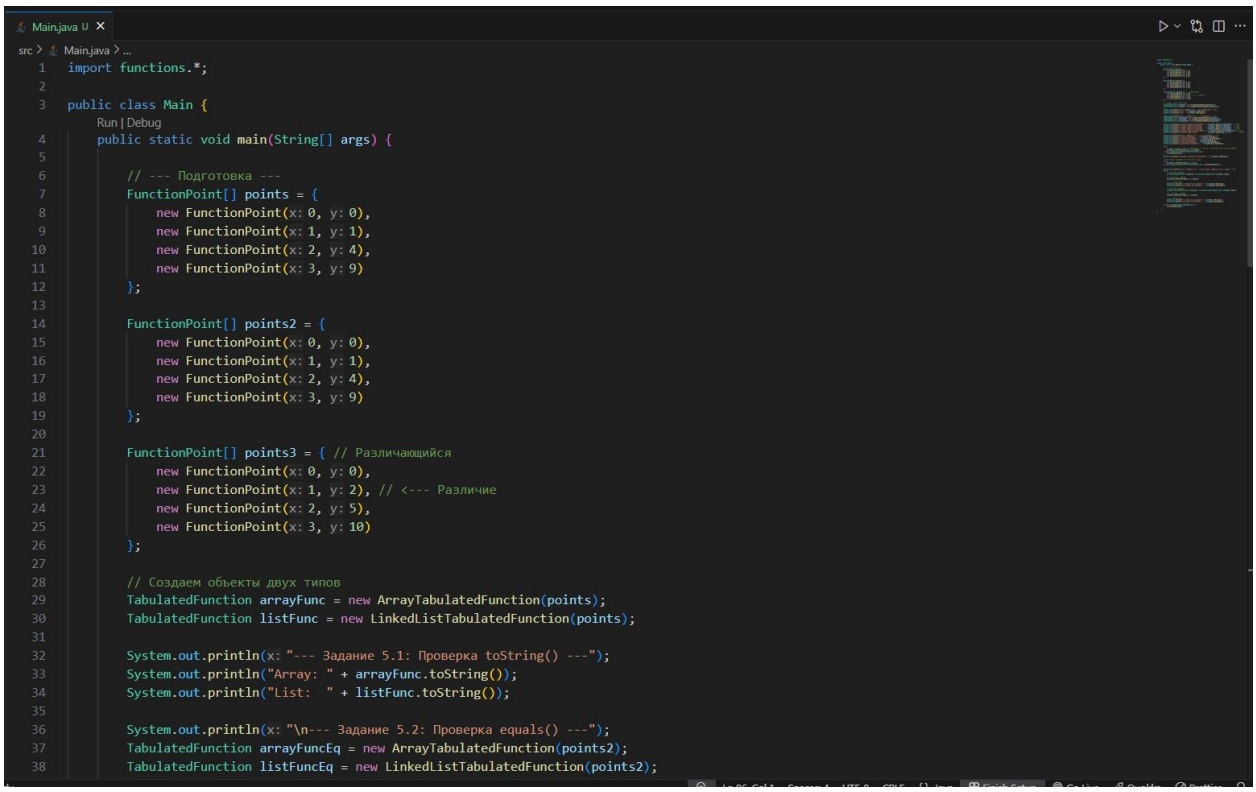
Задание 4

Я сделал так, чтобы все объекты типа TabulatedFunction были клонируемыми с точки зрения JVM и внёс метод clone() в этот интерфейс.

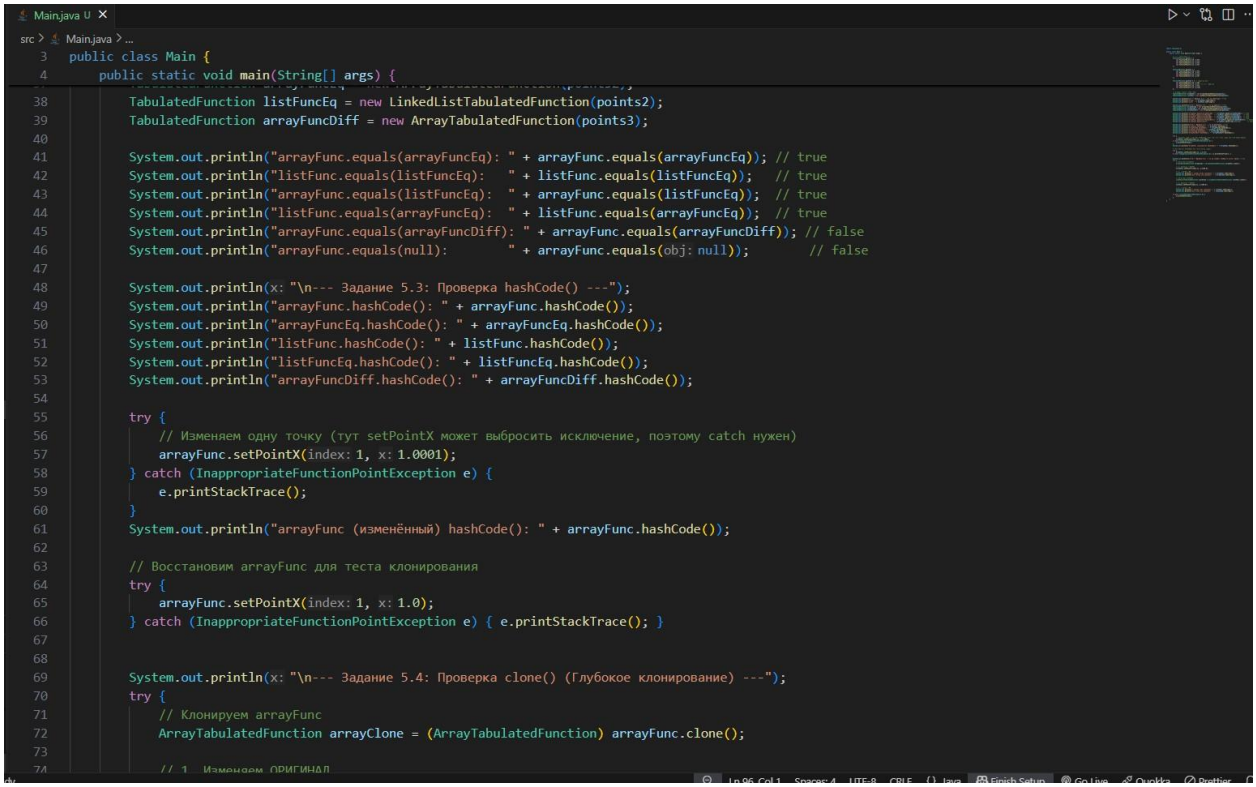


Задание 5 Я проверил работу написанных методов.

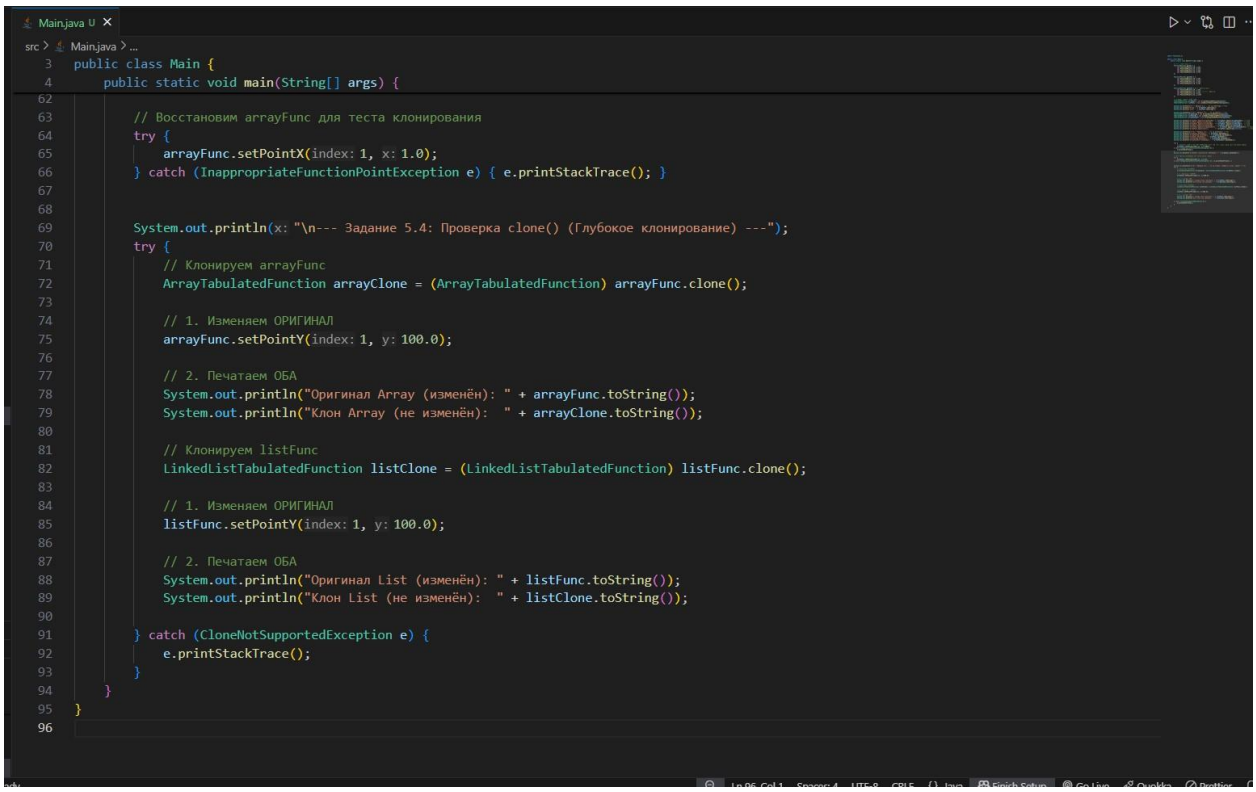
- Я проверил работу метода toString() для объектов типов ArrayTabulatedFunction и LinkedListTabulatedFunction, вывев строковое представление объектов в консоль.
- Я проверил работу метода equals(), вызвав его для одинаковых и различающихся объектов одинаковых и различающихся классов.
- Я проверил работу метода hashCode(), вывев в консоль его значения для всех использованных объектов. Я убедился в согласованности работы методов equals() и hashCode(). Также я попробовал незначительно изменить один из объектов и проверил, как изменится значение хэш-кода объекта.
- Я проверил работу метода clone() для объектов обоих классов табулированных функций. Я убедился, что произведено именно глубокое клонирование: для этого после клонирования я изменил исходные объекты и проверил, что объекты-клоны не изменились.



```java
import functions.*;

public class Main {
    Run | Debug
    public static void main(String[] args) {

        // --- Подготовка ---
        FunctionPoint[] points = {
            new FunctionPoint(x: 0, y: 0),
            new FunctionPoint(x: 1, y: 1),
            new FunctionPoint(x: 2, y: 4),
            new FunctionPoint(x: 3, y: 9)
        };

        FunctionPoint[] points2 = {
            new FunctionPoint(x: 0, y: 0),
            new FunctionPoint(x: 1, y: 1),
            new FunctionPoint(x: 2, y: 4),
            new FunctionPoint(x: 3, y: 9)
        };

        FunctionPoint[] points3 = { // Различающийся
            new FunctionPoint(x: 0, y: 0),
            new FunctionPoint(x: 1, y: 2), // <--- Различие
            new FunctionPoint(x: 2, y: 5),
            new FunctionPoint(x: 3, y: 10)
        };

        // Создаем объекты двух типов
        TabulatedFunction arrayFunc = new ArrayTabulatedFunction(points);
        TabulatedFunction listFunc = new LinkedListTabulatedFunction(points);

        System.out.println(x: "--- Задание 5.1: Проверка toString() ---");
        System.out.println("Array: " + arrayFunc.toString());
        System.out.println("List:  " + listFunc.toString());

        System.out.println(x: "\n--- Задание 5.2: Проверка equals() ---");
        TabulatedFunction arrayFuncEq = new ArrayTabulatedFunction(points2);
        TabulatedFunction listFuncEq = new LinkedListTabulatedFunction(points2);
```



```java
public class Main {
    public static void main(String[] args) {
        TabulatedFunction listFuncEq = new LinkedListTabulatedFunction(points2);
        TabulatedFunction arrayFuncDiff = new ArrayTabulatedFunction(points3);

        System.out.println("arrayFunc.equals(arrayFuncEq): " + arrayFunc.equals(arrayFuncEq)); // true
        System.out.println("listFunc.equals(listFuncEq):   " + listFunc.equals(listFuncEq));   // true
        System.out.println("arrayFunc.equals(listFuncEq):  " + arrayFunc.equals(listFuncEq));  // true
        System.out.println("listFunc.equals(arrayFuncEq):  " + listFunc.equals(arrayFuncEq));  // true
        System.out.println("arrayFunc.equals(arrayFuncDiff): " + arrayFunc.equals(arrayFuncDiff)); // false
        System.out.println("arrayFunc.equals(null):        " + arrayFunc.equals(obj: null));       // false

        System.out.println(x: "\n--- Задание 5.3: Проверка hashCode() ---");
        System.out.println("arrayFunc.hashCode(): " + arrayFunc.hashCode());
        System.out.println("arrayFuncEq.hashCode(): " + arrayFuncEq.hashCode());
        System.out.println("listFunc.hashCode(): " + listFunc.hashCode());
        System.out.println("listFuncEq.hashCode(): " + listFuncEq.hashCode());
        System.out.println("arrayFuncDiff.hashCode(): " + arrayFuncDiff.hashCode());

        try {
            // Изменяем одну точку (тут setPointX может выбросить исключение, поэтому catch нужен)
            arrayFunc.setPointX(index: 1, x: 1.0001);
        } catch (InappropriateFunctionPointException e) {
            e.printStackTrace();
        }
        System.out.println("arrayFunc (изменённый) hashCode(): " + arrayFunc.hashCode());

        // Восстановим arrayFunc для теста клонирования
        try {
            arrayFunc.setPointX(index: 1, x: 1.0);
        } catch (InappropriateFunctionPointException e) { e.printStackTrace(); }


        System.out.println(x: "\n--- Задание 5.4: Проверка clone() (Глубокое клонирование) ---");
        try {
            // Клонируем arrayFunc
            ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc.clone();

            // 1. Изменяем ОРИГИНАЛ
```



```java
public class Main {
    public static void main(String[] args) {

        // Восстановим arrayFunc для теста клонирования
        try {
            arrayFunc.setPointX(index: 1, x: 1.0);
        } catch (InappropriateFunctionPointException e) { e.printStackTrace(); }


        System.out.println(x: "\n--- Задание 5.4: Проверка clone() (Глубокое клонирование) ---");
        try {
            // Клонируем arrayFunc
            ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc.clone();

            // 1. Изменяем ОРИГИНАЛ
            arrayFunc.setPointY(index: 1, y: 100.0);

            // 2. Печатаем ОБА
            System.out.println("Оригинал Array (изменён): " + arrayFunc.toString());
            System.out.println("Клон Array (не изменён):  " + arrayClone.toString());

            // Клонируем listFunc
            LinkedListTabulatedFunction listClone = (LinkedListTabulatedFunction) listFunc.clone();

            // 1. Изменяем ОРИГИНАЛ
            listFunc.setPointY(index: 1, y: 100.0);

            // 2. Печатаем ОБА
            System.out.println("Оригинал List (изменён): " + listFunc.toString());
            System.out.println("Клон List (не изменён):  " + listClone.toString());

        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
}
```

```
PS C:\projects\Lab-5-2025>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:54875' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\d095eae8d3bb290a020d048fa721c1ed\redhat.java\jdt_ws\Lab-5-2025_45d506f6\bin' 'Main'
--- Задание 5.1: Проверка toString() ---
Array: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
List:  {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}

--- Задание 5.2: Проверка equals() ---
arrayFunc.equals(arrayFuncEq): true
listFunc.equals(listFuncEq):   true
arrayFunc.equals(listFuncEq):  true
listFunc.equals(arrayFuncEq):  true
arrayFunc.equals(arrayFuncDiff): false
arrayFunc.equals(null):        false

--- Задание 5.3: Проверка hashCode() ---
arrayFunc.hashCode(): -1578444637
arrayFuncEq.hashCode(): -1578444637
listFunc.hashCode(): -1578444637
listFuncEq.hashCode(): -1578444637
arrayFuncDiff.hashCode(): -149104477
arrayFunc (изменённый) hashCode(): -2111778756

--- Задание 5.4: Проверка clone() (Глубокое клонирование) ---
Оригинал Array (изменён): {(0.0; 0.0), (1.0; 100.0), (2.0; 4.0), (3.0; 9.0)}
Клон Array (не изменён):  {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
Оригинал List (изменён): {(0.0; 0.0), (1.0; 100.0), (2.0; 4.0), (3.0; 9.0)}
Клон List (не изменён):  {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}
PS C:\projects\Lab-5-2025>
```