

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.

```
FunctionPoint.java X
src > functions > FunctionPoint.java > ...
1 package functions;
2
3 // Добавляем implements Cloneable для задания 4
4 public class FunctionPoint implements java.io.Serializable, Cloneable {
5     private double x;
6     private double y;
7
8     private static final long serialVersionUID = 1L;
9
10    public FunctionPoint(double x, double y) {
11        this.x = x;
12        this.y = y;
13    }
14
15    public FunctionPoint(FunctionPoint point) {
16        this.x = point.x;
17        this.y = point.y;
18    }
19
20    public FunctionPoint() {
21        this.x = 0;
22        this.y = 0;
23    }
24
25    public double getX() {
26        return x;
27    }
28
29    public double getY() {
30        return y;
31    }
32
33    public void setX(double x) {
34        this.x = x;
35    }
36
37    public void setY(double y) {
38        this.y = y;
39    }
}
```

```
FunctionPoint.java X
src > functions > FunctionPoint.java > ...
4 public class FunctionPoint implements java.io.Serializable, Cloneable {
37     public void setY(double y) {
38         this.y = y;
39     }
40
41     // --- НОВЫЕ МЕТОДЫ (Задание 1) ---
42
43     @Override
44     public String toString() {
45         // Возвращает строку в формате "(x; y)"
46         return "(" + x + "; " + y + ")";
47     }
48
49     @Override
50     public boolean equals(Object o) {
51         // 1. Проверка на тот же самый объект
52         if (this == o) return true;
53         // 2. Проверка, что o - это вообще FunctionPoint
54         if (o == null || getClass() != o.getClass()) return false;
55
56         // 3. Приведение типа и сравнение полей
57         FunctionPoint that = (FunctionPoint) o;
58
59         // Используем Double.compare для точного сравнения битов double
60         return Double.compare(that.x, x) == 0 && Double.compare(that.y, y) == 0;
61     }
62
63     @Override
64     public int hashCode() {
65         // Используем XOR, как предложено в задании
66         long xBits = Double.doubleToLongBits(x);
67         long yBits = Double.doubleToLongBits(y);
68
69         int xHash = (int) (xBits ^ (xBits >> 32)); // Хэш для x
70         int yHash = (int) (yBits ^ (yBits >> 32)); // Хэш для y
71
72         // Комбинируем хэши
73         return 31 * xHash + yHash;
74     }
}
```

```

66     long xBits = Double.doubleToLongBits(x);
67     long yBits = Double.doubleToLongBits(y);
68
69     int xHash = (int) (xBits ^ (xBits >> 32)); // Хэш для x
70     int yHash = (int) (yBits ^ (yBits >> 32)); // Хэш для y
71
72     // Комбинируем хэши
73     return 31 * xHash + yHash;
74 }
75
76 @Override
77 public Object clone() throws CloneNotSupportedException {
78     // Вызываем clone(), родительского класса Object
79     return super.clone();
80 }
81 }
82

```

## 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.

```

ArrayTabulatedFunction.java X
src > functions > ArrayTabulatedFunction.java > ...
1 package functions;
2
3 // Import java.io.Serializable;
4 import java.util.Arrays; // Импортируем Arrays для hashCode
5
6 public class ArrayTabulatedFunction implements TabulatedFunction {
7     private FunctionPoint[] points;
8     private static final long serialVersionUID = 1L;
9
10    // --- Конструкторы (из JP 2, 3, 4) ---
11    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
12        if (leftX >= rightX || pointsCount < 2) {
13            throw new IllegalArgumentException(s: "Invalid arguments for function creation: leftX >= rightX or pointsCount < 2");
14        }
15        this.points = new FunctionPoint[pointsCount];
16        double step = (rightX - leftX) / (pointsCount - 1);
17        for (int i = 0; i < pointsCount; ++i) {
18            points[i] = new FunctionPoint(leftX + i * step, y: 0);
19        }
20    }
21
22    public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
23        int count = values.length;
24        if (leftX >= rightX || count < 2) {
25            throw new IllegalArgumentException(s: "Invalid arguments for function creation: leftX >= rightX or pointsCount < 2");
26        }
27        this.points = new FunctionPoint[count];
28        double step = (rightX - leftX) / (count - 1);
29        for (int i = 0; i < count; ++i) {
30            points[i] = new FunctionPoint(leftX + i * step, values[i]);
31        }
32    }
33
34    public ArrayTabulatedFunction(FunctionPoint[] points) {
35        if (points.length < 2) {
36            throw new IllegalArgumentException(s: "Function must have at least 2 points");
37        }
38        for (int i = 0; i < points.length - 1; ++i) {
39            if (points[i].getX() >= points[i + 1].getX()) {
40                throw new IllegalArgumentException(s: "Points are not sorted by X");
41            }
42        }
43        this.points = new FunctionPoint[points.length];
44        for (int i = 0; i < points.length; ++i) {
45            this.points[i] = new FunctionPoint(points[i]);
46        }
47    }
48

```

```

ArrayTabulatedFunction.java U X
src > functions > . ArrayTabulatedFunction.java > ...
6 public class ArrayTabulatedFunction implements TabulatedFunction {
34     public ArrayTabulatedFunction(FunctionPoint[] points) {
46     }
47 }
48
49 // --- Основные методы (getLeftDomainBorder, getFunctionValue, etc.) ---
50
51 public double getLeftDomainBorder() {
52     return points[0].getX();
53 }
54
55 public double getRightDomainBorder() {
56     return points[points.length - 1].getX();
57 }
58
59 public double getFunctionValue(double x) {
60     if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
61         return Double.NaN;
62     }
63     for (int i = 0; i < points.length - 1; ++i) {
64         if (points[i].getX() <= x && x <= points[i + 1].getX()) {
65             double x1 = points[i].getX();
66             double y1 = points[i].getY();
67             double x2 = points[i + 1].getX();
68             double y2 = points[i + 1].getY();
69             return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
70         }
71     }
72     return Double.NaN;
73 }
74
75 public int getPointsCount() {
76     return points.length;
77 }
78
79 private void checkIndex(int index) {
80     if (index < 0 || index >= points.length) {
81         throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds [0, " + (points.length - 1) + "]");
82     }
83 }
84
85 public FunctionPoint getPoint(int index) {
86     checkIndex(index);
87     return new FunctionPoint(points[index]);
88 }
89
90 public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {

```

```

ArrayTabulatedFunction.java U X
src > functions > . ArrayTabulatedFunction.java > ...
85     public FunctionPoint getPoint(int index) {
88     }
89
90     public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
91         checkIndex(index);
92         double newX = point.getX();
93         double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
94         double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
95
96         if (newX <= leftBound || newX >= rightBound) {
97             throw new InappropriateFunctionPointException(message: "New point's X coordinate is out of the allowed interval");
98         }
99         points[index] = new FunctionPoint(point);
100     }
101
102     public double getPointX(int index) {
103         checkIndex(index);
104         return points[index].getX();
105     }
106
107     public void setPointX(int index, double x) throws InappropriateFunctionPointException {
108         checkIndex(index);
109         double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
110         double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
111
112         if (x <= leftBound || x >= rightBound) {
113             throw new InappropriateFunctionPointException(message: "New X coordinate is out of the allowed interval");
114         }
115         points[index].setX(x);
116     }
117
118     public double getPointY(int index) {
119         checkIndex(index);
120         return points[index].getY();
121     }
122
123     public void setPointY(int index, double y) {
124         checkIndex(index);
125         points[index].setY(y);
126     }
127
128     public void deletePoint(int index) {
129         checkIndex(index);
130         if (points.length < 3) {
131             throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
132         }
133         FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];

```

```

src > functions > ArrayTabulatedFunction.java > ...
6 public class ArrayTabulatedFunction implements TabulatedFunction {
128 public void deletePoint(int index) {
129     // ...
131     throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
132 }
133 FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
134 System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, index);
135 System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
136 points = newPoints;
137 }
138 }
139 public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
140     int insertIndex = 0;
141     while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) {
142         insertIndex++;
143     }
144     if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) {
145         throw new InappropriateFunctionPointException("Point with X=" + point.getX() + " already exists.");
146     }
147     FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
148     System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, insertIndex);
149     newPoints[insertIndex] = new FunctionPoint(point);
150     System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
151     points = newPoints;
152 }
153 }
154 // --- НОВЫЕ МЕТОДЫ (Задание 2) ---
155 }
156 public String toString() {
157     // Формат: {(x1; y1), (x2; y2), ...}
158     StringBuilder sb = new StringBuilder();
159     sb.append("{");
160     for (int i = 0; i < points.length; i++) {
161         sb.append(points[i].toString()); // Используем toString() из FunctionPoint
162         if (i < points.length - 1) {
163             sb.append(", ");
164         }
165     }
166     sb.append("}");
167     return sb.toString();
168 }
169 }
170 public boolean equals(Object o) {
171     if (this == o) return true;
172     // Проверим, что 'o' - это любая реализация TabulatedFunction
173     if (!(o instanceof TabulatedFunction)) return false;
174 }

```

```

src > functions > ArrayTabulatedFunction.java > ...
6 public class ArrayTabulatedFunction implements TabulatedFunction {
172 public boolean equals(Object o) {
173     if (this == o) return true;
174     // Проверим, что 'o' - это любая реализация TabulatedFunction
175     if (!(o instanceof TabulatedFunction)) return false;
176     TabulatedFunction that = (TabulatedFunction) o;
177     // Сравниваем количество точек
178     if (this.getPointsCount() != that.getPointsCount()) return false;
179     // Оптимизация для ArrayTabulatedFunction (прямой доступ к массиву)
180     if (o instanceof ArrayTabulatedFunction) {
181         ArrayTabulatedFunction thatArray = (ArrayTabulatedFunction) o;
182         // Сравниваем каждый элемент массива
183         for (int i = 0; i < this.points.length; i++) {
184             // Используем equals() из FunctionPoint
185             if (!this.points[i].equals(thatArray.points[i])) {
186                 return false;
187             }
188         }
189     } else {
190         // Стандартное сравнение через getPoint() для других реализаций
191         for (int i = 0; i < this.getPointsCount(); i++) {
192             if (!this.getPoint(i).equals(that.getPoint(i))) {
193                 return false;
194             }
195         }
196     }
197     return true;
198 }
199 }
200 public int hashCode() {
201     // Добавляем count (points.length) к хэшу, как в задании
202     int result = Arrays.hashCode(points);
203     result = 31 * result + Integer.hashCode(points.length);
204     return result;
205 }
206 }
207 public Object clone() throws CloneNotSupportedException {
208     // Глубокое клонирование
209     FunctionPoint[] clonedPoints = new FunctionPoint[this.points.length];
210     for (int i = 0; i < this.points.length; i++) {
211         // Клонирование каждой точки
212         clonedPoints[i] = (FunctionPoint) this.points[i].clone();
213     }
214     // Создаем новый объект с клонированным массивом
215     return new ArrayTabulatedFunction(clonedPoints);
216 }
217 }
218 }
219 }
220 }
221 }
222 }

```

```

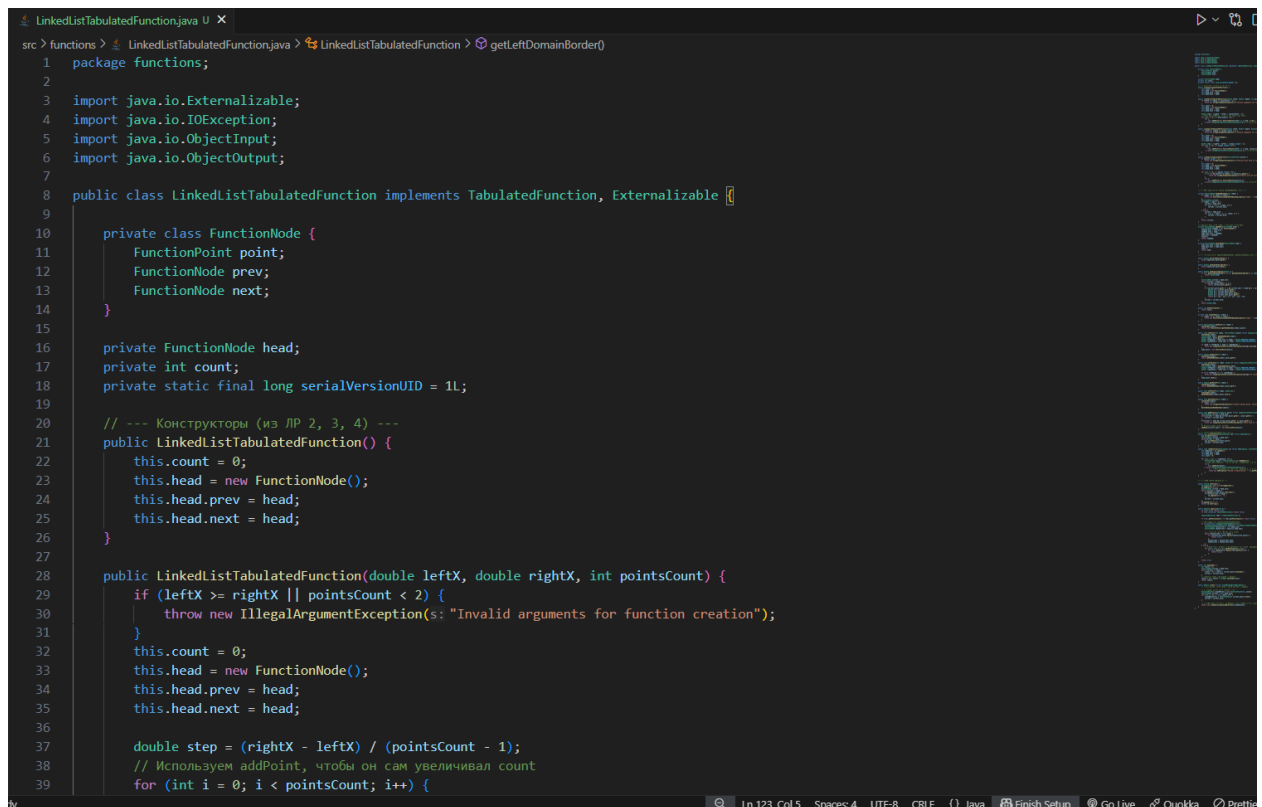
212 // Глубокое клонирование
213 FunctionPoint[] clonedPoints = new FunctionPoint[this.points.length];
214 for (int i = 0; i < this.points.length; i++) {
215     // Клонирование каждой точки
216     clonedPoints[i] = (FunctionPoint) this.points[i].clone();
217 }
218 // Создаем новый объект с клонированным массивом
219 return new ArrayTabulatedFunction(clonedPoints);
220 }
221 }
222 }

```

### 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.



```
1 package functions;
2
3 import java.io.Externalizable;
4 import java.io.IOException;
5 import java.io.ObjectInput;
6 import java.io.ObjectOutput;
7
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
9
10     private class FunctionNode {
11         FunctionPoint point;
12         FunctionNode prev;
13         FunctionNode next;
14     }
15
16     private FunctionNode head;
17     private int count;
18     private static final long serialVersionUID = 1L;
19
20     // --- Конструкторы (из ЛР 2, 3, 4) ---
21     public LinkedListTabulatedFunction() {
22         this.count = 0;
23         this.head = new FunctionNode();
24         this.head.prev = head;
25         this.head.next = head;
26     }
27
28     public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
29         if (leftX >= rightX || pointsCount < 2) {
30             throw new IllegalArgumentException(s: "Invalid arguments for function creation");
31         }
32         this.count = 0;
33         this.head = new FunctionNode();
34         this.head.prev = head;
35         this.head.next = head;
36
37         double step = (rightX - leftX) / (pointsCount - 1);
38         // Используем addPoint, чтобы он сам увеличивал count
39         for (int i = 0; i < pointsCount; i++) {
```

```

src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
28     public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
    // --- Внутренние методы списка (getNodeByIndex, etc.) ---
39     for (int i = 0; i < pointsCount; i++) {
40         try {
41             this.addPoint(new FunctionPoint(leftX + i * step, y: 0));
42         } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
43     }
44 }
45
46 public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
47     if (leftX >= rightX || values.length < 2) {
48         throw new IllegalArgumentException(s: "Invalid arguments for function creation");
49     }
50     this.count = 0;
51     this.head = new FunctionNode();
52     this.head.prev = head;
53     this.head.next = head;
54
55     double step = (rightX - leftX) / (values.length - 1);
56     for (int i = 0; i < values.length; i++) {
57         try {
58             this.addPoint(new FunctionPoint(leftX + i * step, values[i]));
59         } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
60     }
61 }
62
63 public LinkedListTabulatedFunction(FunctionPoint[] points) {
64     if (points.length < 2) {
65         throw new IllegalArgumentException(s: "Function must have at least 2 points");
66     }
67     this.count = 0;
68     this.head = new FunctionNode();
69     this.head.prev = head;
70     this.head.next = head;
71
72     for (int i = 0; i < points.length; ++i) {
73         if (i > 0 && points[i-1].getX() >= points[i].getX()) {
74             throw new IllegalArgumentException(s: "Points are not sorted by X");
75         }
76     }
77 }

```

```

src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
63     public LinkedListTabulatedFunction(FunctionPoint[] points) {
74         throw new IllegalArgumentException(s: "Points are not sorted by X");
75     }
76     try {
77         this.addPoint(new FunctionPoint(points[i]));
78     } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
79 }
80 }
81
82 // --- Внутренние методы списка (getNodeByIndex, etc.) ---
83
84 private FunctionNode getNodeByIndex(int index) {
85     if (index < 0 || index >= count) {
86         throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds");
87     }
88     FunctionNode current;
89     if (index < count / 2) {
90         current = head.next;
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 // Добавляет узел ПЕРЕД 'node' и возвращает новый узел
104 private FunctionNode addNode(FunctionNode node) {
105     FunctionNode newNode = new FunctionNode();
106     newNode.prev = node.prev;
107     newNode.next = node;
108     node.prev.next = newNode;
109     node.prev = newNode;
110 }

```

```

src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
105 private FunctionNode addNode(FunctionNode node) {
106     node.prev.next = new Node();
107     node.prev = newNode;
108     count++;
109     return newNode;
110 }
111
112 private FunctionNode deleteNode(FunctionNode node) {
113     node.prev.next = node.next;
114     node.next.prev = node.prev;
115     count--;
116     return node;
117 }
118
119 // --- Основные методы (getLeftDomainBorder, getFunctionValue, etc.) ---
120
121 public double getLeftDomainBorder() {
122     return head.next.point.getX();
123 }
124
125 public double getRightDomainBorder() {
126     return head.prev.point.getX();
127 }
128
129 public double getFunctionValue(double x) {
130     if (x < getLeftDomainBorder() || x > getRightDomainBorder() || count == 0) {
131         return Double.NaN;
132     }
133
134     FunctionNode current = head.next;
135     while (current != head) {
136         if (current.point.getX() == x) {
137             return current.point.getY();
138         }
139         if (current.point.getX() < x && current.next != head && x < current.next.point.getX()) {
140             double x1 = current.point.getX();
141             double y1 = current.point.getY();
142             double x2 = current.next.point.getX();
143

```

```

144         double x2 = current.next.point.getX();
145         double y2 = current.next.point.getY();
146         return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
147     }
148     current = current.next;
149 }
150 return Double.NaN;
151 }
152
153 public int getPointsCount() {
154     return count;
155 }
156
157 private void checkIndex(int index) {
158     if (index < 0 || index >= count) {
159         throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds [0, " + (count - 1) + "]");
160     }
161 }
162
163 public FunctionPoint getPoint(int index) {
164     checkIndex(index);
165     return new FunctionPoint(getNodeByIndex(index).point);
166 }
167
168 public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
169     checkIndex(index);
170     FunctionNode node = getNodeByIndex(index);
171     double newX = point.getX();
172     double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
173     double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
174
175     if (newX <= leftBound || newX >= rightBound) {
176         throw new InappropriateFunctionPointException(message: "X coordinate is out of the allowed interval");
177     }
178     node.point = new FunctionPoint(point);
179 }
180
181

```

```
src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
169 public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
180 }
181
182 public double getPointX(int index) {
183     checkIndex(index);
184     return getNodeByIndex(index).point.getX();
185 }
186
187 public void setPointX(int index, double x) throws InappropriateFunctionPointException {
188     checkIndex(index);
189     FunctionNode node = getNodeByIndex(index);
190     double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
191     double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
192
193     if (x <= leftBound || x >= rightBound) {
194         throw new InappropriateFunctionPointException(message: "X coordinate is out of the allowed interval");
195     }
196     node.point.setX(x);
197 }
198
199 public double getPointY(int index) {
200     checkIndex(index);
201     return getNodeByIndex(index).point.getY();
202 }
203
204 public void setPointY(int index, double y) {
205     checkIndex(index);
206     getNodeByIndex(index).point.setY(y);
207 }
208
209 public void deletePoint(int index) {
210     checkIndex(index);
211     if (count < 3) {
212         throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
213     }
214     deleteNode(getNodeByIndex(index));
215 }
216
```

```
src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
215 }
216
217 public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
218     FunctionNode current = head.next;
219     while(current != head && current.point.getX() < point.getX()) {
220         current = current.next;
221     }
222     if(current != head && current.point.getX() == point.getX()) {
223         throw new InappropriateFunctionPointException("Point with X=" + point.getX() + " already exists.");
224     }
225     // Вставляем узел ПЕРЕД 'current'
226     addNode(current).point = new FunctionPoint(point);
227 }
228
229 // --- Методы Externalizable (из ЛР 4) ---
230 public void writeExternal(ObjectOutput out) throws IOException {
231     out.writeInt(count);
232     FunctionNode current = head.next;
233     while(current != head) {
234         out.writeObject(current.point);
235         current = current.next;
236     }
237 }
238
239 public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
240     int readCount = in.readInt();
241     this.head.prev = head;
242     this.head.next = head;
243     this.count = 0;
244
245     for (int i = 0; i < readCount; i++) {
246         FunctionPoint point = (FunctionPoint) in.readObject();
247         // Используем addPoint, чтобы не нарушать инкапсуляцию и проверки
248         try {
249             this.addPoint(point);
250         } catch (InappropriateFunctionPointException e) {
251             // При десериализации не должно быть дубликатов, но проверка нужна
252             throw new IOException("Failed to deserialize: " + e.getMessage());
253         }
254     }
255 }
```

```
src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
239     public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
251         // При десериализации не должно быть дубликатов, но проверка нужна
252         throw new IOException("Failed to deserialize: " + e.getMessage());
253     }
254 }
255
256
257 // --- НОВЫЕ МЕТОДЫ (Задание 3) ---
258
259 public String toString() {
260     StringBuilder sb = new StringBuilder();
261     sb.append(str: "{");
262     FunctionNode current = head.next;
263     while (current != head) {
264         sb.append(current.point.toString());
265         if (current.next != head) {
266             sb.append(str: ", ");
267         }
268         current = current.next;
269     }
270     sb.append(str: "}");
271     return sb.toString();
272 }
273
274
275 public boolean equals(Object o) {
276     if (this == o) return true;
277     if (!(o instanceof TabulatedFunction)) return false;
278
279     TabulatedFunction that = (TabulatedFunction) o;
280
281     if (this.getPointsCount() != that.getPointsCount()) return false;
282
283     // Оптимизация для LinkedListTabulatedFunction
284     if (o instanceof LinkedListTabulatedFunction) {
285         LinkedListTabulatedFunction thatList = (LinkedListTabulatedFunction) o;
286         FunctionNode thisCurrent = this.head.next;
287         FunctionNode thatCurrent = thatList.head.next;
```

```
src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
275     public boolean equals(Object o) {
287         FunctionNode thatCurrent = thatList.head.next;
288
289         // Проходим по обоим спискам одновременно
290         while (thisCurrent != this.head) {
291             if (!thisCurrent.point.equals(thatCurrent.point)) {
292                 return false;
293             }
294             thisCurrent = thisCurrent.next;
295             thatCurrent = thatCurrent.next;
296         }
297     } else {
298         // Стандартное сравнение через getPoint() для других реализаций
299         for (int i = 0; i < this.getPointsCount(); i++) {
300             if (!this.getPoint(i).equals(that.getPoint(i))) {
301                 return false;
302             }
303         }
304     }
305
306     return true;
307 }
308
309 public int hashCode() {
310     int result = 1;
311     FunctionNode current = head.next;
312     while (current != head) {
313         result = 31 * result + current.point.hashCode();
314         current = current.next;
315     }
316     // Добавляем count, как указано в задании
317     result = 31 * result + Integer.hashCode(count);
318     return result;
319 }
320
321 public Object clone() throws CloneNotSupportedException {
322     // "Пересобираем" новый объект, как предложено в задании
323 }
```

```
LinkedListTabulatedFunction.java U X
src > functions > LinkedListTabulatedFunction.java > LinkedListTabulatedFunction > getLeftDomainBorder()
8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
275 public boolean equals(Object o) {
303     }
304 }
305
306     return true;
307 }
308
309 public int hashCode() {
310     int result = 1;
311     FunctionNode current = head.next;
312     while (current != head) {
313         result = 31 * result + current.point.hashCode();
314         current = current.next;
315     }
316     // Добавляем count, как указано в задании
317     result = 31 * result + Integer.hashCode(count);
318     return result;
319 }
320
321 public Object clone() throws CloneNotSupportedException {
322     // "Пересобираем" новый объект, как предложено в задании
323
324     // 1. Создаем массив клонированных точек
325     FunctionPoint[] clonedPoints = new FunctionPoint[this.count];
326     FunctionNode current = this.head.next;
327     for (int i = 0; i < this.count; i++) {
328         clonedPoints[i] = (FunctionPoint) current.point.clone();
329         current = current.next;
330     }
331
332     // 2. Используем конструктор из Задания 1 для создания нового объекта
333     return new LinkedListTabulatedFunction(clonedPoints);
334 }
335
336 }
```

## Задание 4

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

```
TabulatedFunction.java U X
src > functions > TabulatedFunction.java > ...
1 package functions;
2
3 // 1. Добавляем extends Cloneable
4 public interface TabulatedFunction extends Function, java.io.Serializable, Cloneable {
5
6     // ... (все старые методы)
7     int getPointsCount();
8     FunctionPoint getPoint(int index);
9     void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
10    double getPointX(int index);
11    void setPointX(int index, double x) throws InappropriateFunctionPointException;
12    double getPointY(int index);
13    void setPointY(int index, double y);
14    void deletePoint(int index);
15    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
16
17    // 2. Добавляем объявление метода clone()
18    Object clone() throws CloneNotSupportedException;
19 }
20
```

## Задание 5

Я проверил работу написанных методов.

- Я проверил работу метода toString() для объектов типов ArrayTabulatedFunction и LinkedListTabulatedFunction, выведя 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(), выведя 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.

```
Mainjava U X
src > Mainjava > ...
1 import functions.*;
2
3 public class Main {
4     public static void main(String[] args) {
5
6         // --- Подготовка ---
7         FunctionPoint[] points = {
8             new FunctionPoint(x: 0, y: 0),
9             new FunctionPoint(x: 1, y: 1),
10            new FunctionPoint(x: 2, y: 4),
11            new FunctionPoint(x: 3, y: 9)
12        };
13
14        FunctionPoint[] points2 = {
15            new FunctionPoint(x: 0, y: 0),
16            new FunctionPoint(x: 1, y: 1),
17            new FunctionPoint(x: 2, y: 4),
18            new FunctionPoint(x: 3, y: 9)
19        };
20
21        FunctionPoint[] points3 = { // Различающийся
22            new FunctionPoint(x: 0, y: 0),
23            new FunctionPoint(x: 1, y: 2), // <--- Различие
24            new FunctionPoint(x: 2, y: 5),
25            new FunctionPoint(x: 3, y: 10)
26        };
27
28        // Создаем объекты двух типов
29        TabulatedFunction arrayFunc = new ArrayTabulatedFunction(points);
30        TabulatedFunction listFunc = new LinkedListTabulatedFunction(points);
31
32        System.out.println(x: "--- Задание 5.1: Проверка toString() ---");
33        System.out.println("Array: " + arrayFunc.toString());
34        System.out.println("List: " + listFunc.toString());
35
36        System.out.println(x: "\n--- Задание 5.2: Проверка equals() ---");
37        TabulatedFunction arrayFuncEq = new ArrayTabulatedFunction(points2);
38        TabulatedFunction listFuncEq = new LinkedListTabulatedFunction(points2);
```

```
Mainjava U X
src > Mainjava > ...
3 public class Main {
4     public static void main(String[] args) {
5
6         // --- Подготовка ---
7         FunctionPoint[] points = {
8             new FunctionPoint(x: 0, y: 0),
9             new FunctionPoint(x: 1, y: 1),
10            new FunctionPoint(x: 2, y: 4),
11            new FunctionPoint(x: 3, y: 9)
12        };
13
14        FunctionPoint[] points2 = {
15            new FunctionPoint(x: 0, y: 0),
16            new FunctionPoint(x: 1, y: 1),
17            new FunctionPoint(x: 2, y: 4),
18            new FunctionPoint(x: 3, y: 9)
19        };
20
21        FunctionPoint[] points3 = { // Различающийся
22            new FunctionPoint(x: 0, y: 0),
23            new FunctionPoint(x: 1, y: 2), // <--- Различие
24            new FunctionPoint(x: 2, y: 5),
25            new FunctionPoint(x: 3, y: 10)
26        };
27
28        // Создаем объекты двух типов
29        TabulatedFunction arrayFunc = new ArrayTabulatedFunction(points);
30        TabulatedFunction listFunc = new LinkedListTabulatedFunction(points);
31
32        System.out.println(x: "--- Задание 5.1: Проверка toString() ---");
33        System.out.println("Array: " + arrayFunc.toString());
34        System.out.println("List: " + listFunc.toString());
35
36        System.out.println(x: "\n--- Задание 5.2: Проверка equals() ---");
37        TabulatedFunction arrayFuncEq = new ArrayTabulatedFunction(points2);
38        TabulatedFunction listFuncEq = new LinkedListTabulatedFunction(points2);
39        TabulatedFunction arrayFuncDiff = new ArrayTabulatedFunction(points3);
40
41        System.out.println("arrayFunc.equals(arrayFuncEq): " + arrayFunc.equals(arrayFuncEq)); // true
42        System.out.println("listFunc.equals(listFuncEq): " + listFunc.equals(listFuncEq)); // true
43        System.out.println("arrayFunc.equals(listFuncEq): " + arrayFunc.equals(listFuncEq)); // true
44        System.out.println("listFunc.equals(arrayFuncEq): " + listFunc.equals(arrayFuncEq)); // true
45        System.out.println("arrayFunc.equals(arrayFuncDiff): " + arrayFunc.equals(arrayFuncDiff)); // false
46        System.out.println("arrayFunc.equals(null): " + arrayFunc.equals(obj: null)); // false
47
48        System.out.println(x: "\n--- Задание 5.3: Проверка hashCode() ---");
49        System.out.println("arrayFunc.hashCode(): " + arrayFunc.hashCode());
50        System.out.println("arrayFuncEq.hashCode(): " + arrayFuncEq.hashCode());
51        System.out.println("listFunc.hashCode(): " + listFunc.hashCode());
52        System.out.println("listFuncEq.hashCode(): " + listFuncEq.hashCode());
53        System.out.println("arrayFuncDiff.hashCode(): " + arrayFuncDiff.hashCode());
54
55        try {
56            // Изменяем одну точку (здесь setPointX может выбросить исключение, поэтому catch нужен)
57            arrayFunc.setPointX(index: 1, x: 1.0001);
58        } catch (InappropriateFunctionPointException e) {
59            e.printStackTrace();
60        }
61        System.out.println("arrayFunc (изменённый) hashCode(): " + arrayFunc.hashCode());
62
63        // Восстановим arrayFunc для теста клонирования
64        try {
65            arrayFunc.setPointX(index: 1, x: 1.0);
66        } catch (InappropriateFunctionPointException e) { e.printStackTrace(); }
67
68
69        System.out.println(x: "\n--- Задание 5.4: Проверка clone() (Глубокое клонирование) ---");
70        try {
71            // Клонировем arrayFunc
72            ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc.clone();
73
74            // 1. Изменяем ОРИГИНАЛ
```

```
Mainjava U X
src > Mainjava > ...
3 public class Main {
4     public static void main(String[] args) {
62
63         // Восстановим arrayFunc для теста клонирования
64         try {
65             arrayFunc.setPointX(index: 1, x: 1.0);
66         } catch (InappropriateFunctionPointException e) { e.printStackTrace(); }
67
68
69         System.out.println(x: "\n--- Задание 5.4: Проверка clone() (Глубокое клонирование) ---");
70         try {
71             // Клонировем arrayFunc
72             ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc.clone();
73
74             // 1. Изменяем ОРИГИНАЛ
75             arrayFunc.setPointY(index: 1, y: 100.0);
76
77             // 2. Печатаем ОБА
78             System.out.println("Оригинал Array (изменён): " + arrayFunc.toString());
79             System.out.println("Клон Array (не изменён): " + arrayClone.toString());
80
81             // Клонировем listFunc
82             LinkedListTabulatedFunction listClone = (LinkedListTabulatedFunction) listFunc.clone();
83
84             // 1. Изменяем ОРИГИНАЛ
85             listFunc.setPointY(index: 1, y: 100.0);
86
87             // 2. Печатаем ОБА
88             System.out.println("Оригинал List (изменён): " + listFunc.toString());
89             System.out.println("Клон List (не изменён): " + listClone.toString());
90
91         } catch (CloneNotSupportedException e) {
92             e.printStackTrace();
93         }
94     }
95 }
96
```

```
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\d095eae8d3bb298a020d048fa721c1ed\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>
```