

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.

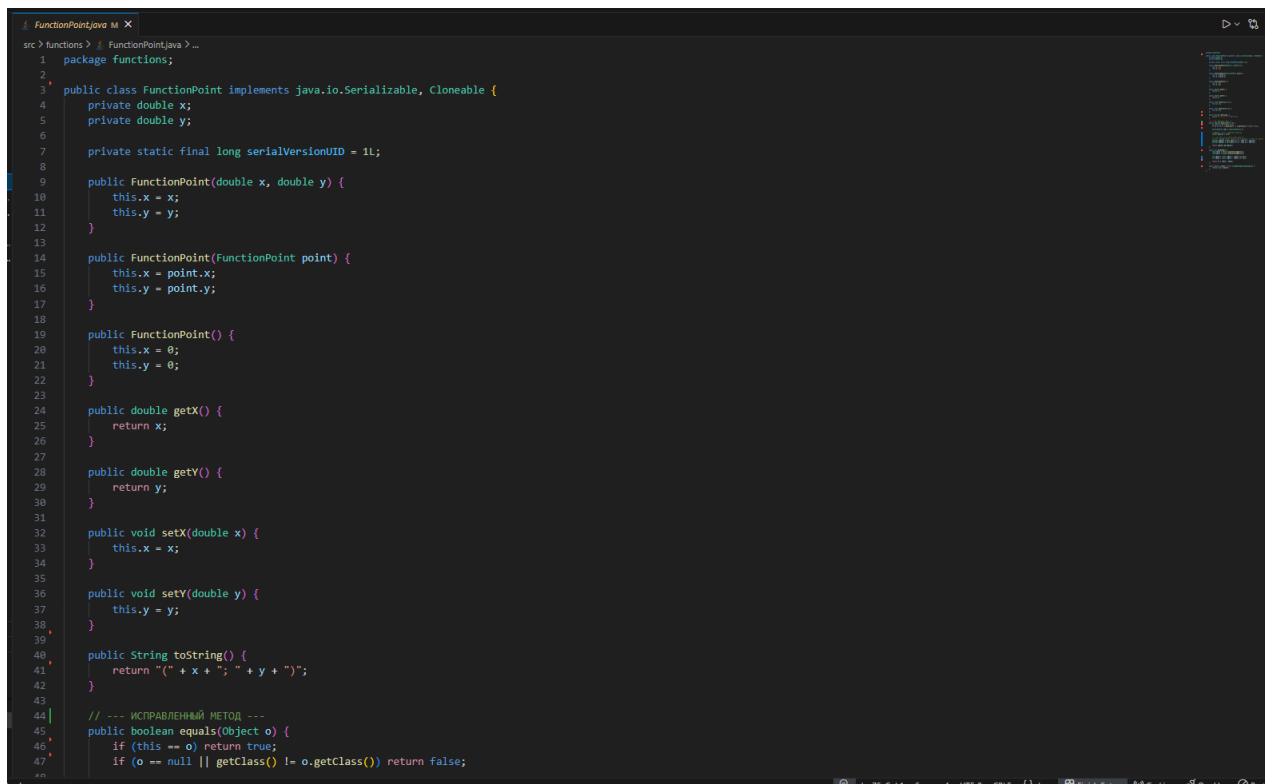
- Проверьте работу метода `toString()` для объектов типов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`, выведя строковое представление объектов в консоль.
- Проверьте работу метода `equals()`, вызывая его для одинаковых и различающихся объектов одинаковых и различающихся классов.
- Проверьте 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.



The screenshot shows a code editor window with the file 'FunctionPoint.java' open. The code implements a class 'FunctionPoint' that represents a point in a 2D space. It includes methods for setting and getting coordinates, a constructor, and overridden methods for toString(), equals(), and hashCode(). The code is annotated with comments and JavaDoc-style descriptions. The right side of the screen shows a file tree and other project details.

```
1 package Functions;
2
3 public class FunctionPoint implements java.io.Serializable, Cloneable {
4     private double x;
5     private double y;
6
7     private static final long serialVersionUID = 1L;
8
9     public FunctionPoint(double x, double y) {
10         this.x = x;
11         this.y = y;
12     }
13
14     public FunctionPoint(FunctionPoint point) {
15         this.x = point.x;
16         this.y = point.y;
17     }
18
19     public FunctionPoint() {
20         this.x = 0;
21         this.y = 0;
22     }
23
24     public double getX() {
25         return x;
26     }
27
28     public double getY() {
29         return y;
30     }
31
32     public void setX(double x) {
33         this.x = x;
34     }
35
36     public void setY(double y) {
37         this.y = y;
38     }
39
40     public String toString() {
41         return "(" + x + ";" + y + ")";
42     }
43
44     // --- ИСПРАВЛЕННЫЙ МЕТОД ---
45     public boolean equals(Object o) {
46         if (this == o) return true;
47         if (o == null || getClass() != o.getClass()) return false;
48
49         return true;
50     }
51
52     protected Object clone() throws CloneNotSupportedException {
53         return super.clone();
54     }
55 }
```

```
src > functions > . FunctionPoint.java > ...
3   public class FunctionPoint implements java.io.Serializable, Cloneable {
36     public void setY(double y) {
38       }
39
40     public String toString() {
41       return "(" + x + "; " + y + ")";
42     }
43
44     // --- ИСПРАВЛЕННЫЙ МЕТОД ---
45     public boolean equals(Object o) {
46       if (this == o) return true;
47       if (o == null || getClass() != o.getClass()) return false;
48
49       FunctionPoint that = (FunctionPoint) o;
50
51       // Задаем точность (машинный эпсилон)
52       double epsilon = 1e-9;
53
54       // Проверяем x и y с учетом погрешности
55       // Если разница между числами меньше epsilon, считаем их равными
56       boolean xEquals = Math.abs(this.x - that.x) < epsilon;
57       boolean yEquals = Math.abs(this.y - that.y) < epsilon;
58
59       return xEquals && yEquals;
60     }
61
62     public int hashCode() {
63       long xBits = Double.doubleToLongBits(x);
64       long yBits = Double.doubleToLongBits(y);
65
66       int xHash = (int) (xBits ^ (xBits >>> 32));
67       int yHash = (int) (yBits ^ (yBits >>> 32));
68
69       return 31 * xHash + yHash;
70     }
71
72     public Object clone() throws CloneNotSupportedException {
73       return super.clone();
74     }
75   }
76 }
```

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 M ✘
src > Functions > ArrayTabulatedFunction.java > ArrayTabulatedFunction > addPoint(FunctionPoint)
1 package functions;
2
3 import java.util.Arrays;
4
5 public class ArrayTabulatedFunction implements TabulatedFunction {
6     private FunctionPoint[] points;
7     private static final long serialVersionUID = 1L;
8
9     // --- Конструкторы ---
10    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
11        if (leftX >= rightX || pointsCount < 2) {
12            throw new IllegalArgumentException("Invalid arguments: leftX >= rightX or pointsCount < 2");
13        }
14        this.points = new FunctionPoint[pointsCount];
15        double step = (rightX - leftX) / (pointsCount - 1);
16        for (int i = 0; i < pointsCount; ++i) {
17            points[i] = new FunctionPoint(leftX + i * step, y: 0);
18        }
19    }
20
21    public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
22        int count = values.length;
23        if (leftX >= rightX || count < 2) {
24            throw new IllegalArgumentException("Invalid arguments: leftX >= rightX or pointsCount < 2");
25        }
26        this.points = new FunctionPoint[count];
27        double step = (rightX - leftX) / (count - 1);
28        for (int i = 0; i < count; ++i) {
29            points[i] = new FunctionPoint(leftX + i * step, values[i]);
30        }
31    }
32
33    public ArrayTabulatedFunction(FunctionPoint[] points) {
34        if (points.length < 2) {
35            throw new IllegalArgumentException("Function must have at least 2 points");
36        }
37        for (int i = 0; i < points.length - 1; ++i) {
38            if (points[i].getX() >= points[i + 1].getX()) {
39                throw new IllegalArgumentException("Points are not sorted by X");
40            }
41        }
42        this.points = new FunctionPoint[points.length];
43        for (int i = 0; i < points.length; ++i) {
44            this.points[i] = new FunctionPoint(points[i]);
45        }
46    }
47
48    // --- Основные методы ---
49
50    public double getLeftDomainBorder() {
51        return points[0].getX();
52    }
53
54    public double getRightDomainBorder() {
55        return points[points.length - 1].getX();
56    }
57
58    // --- ИСПРАВЛЕННЫЙ МЕТОД (Задание 2) ---
59    public double getFunctionValue(double x) {
60        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
61            return Double.NaN;
62        }
63
64        // Проходим по всем интервалам
65        for (int i = 0; i < points.length - 1; ++i) {
66            double x1 = points[i].getX();
67            double x2 = points[i + 1].getX();
68            double y1 = points[i].getY();
69            double y2 = points[i + 1].getY();
70
71            // Если x находится внутри текущего интервала
72            if (x >= x1 && x <= x2) {
73                double epsilon = 1e-9; // Машинный эпсилон
74
75                // Проверка точного совпадения с левой границей интервала
76                if (Math.abs(x - x1) < epsilon) {
77                    return y1;
78                }
79                // Проверка точного совпадения с правой границей интервала
80                if (Math.abs(x - x2) < epsilon) {
81                    return y2;
82                }
83
84                // Если совпадения нет, выполняем интерполяцию
85                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
86            }
87        }
88        return Double.NaN;
89    }
90
91    public int getPointsCount() {
92        return points.length;
93    }
}
Boris Suchkov (14 hours ago) | Q | Ln 170, Col 1 | Spaces: 4 | UFT-8 | CRLF | {} Java | Finish Setup | 8-9 Go Live | ⚡ Quokka | ⚡ Premier
```

```
# ArrayTabulatedFunction.java M ✘
src > Functions > ArrayTabulatedFunction.java > ArrayTabulatedFunction > ArrayTabulatedFunction(FunctionPoint[])
5 public class ArrayTabulatedFunction implements TabulatedFunction {
6
7     // --- Основные методы ---
8
9     public double getLeftDomainBorder() {
10        return points[0].getX();
11    }
12
13    public double getRightDomainBorder() {
14        return points[points.length - 1].getX();
15    }
16
17    // --- ИСПРАВЛЕННЫЙ МЕТОД (Задание 2) ---
18    public double getFunctionValue(double x) {
19        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
20            return Double.NaN;
21        }
22
23        // Проходим по всем интервалам
24        for (int i = 0; i < points.length - 1; ++i) {
25            double x1 = points[i].getX();
26            double x2 = points[i + 1].getX();
27            double y1 = points[i].getY();
28            double y2 = points[i + 1].getY();
29
30            // Если x находится внутри текущего интервала
31            if (x >= x1 && x <= x2) {
32                double epsilon = 1e-9; // Машинный эпсилон
33
34                // Проверка точного совпадения с левой границей интервала
35                if (Math.abs(x - x1) < epsilon) {
36                    return y1;
37                }
38                // Проверка точного совпадения с правой границей интервала
39                if (Math.abs(x - x2) < epsilon) {
40                    return y2;
41                }
42
43                // Если совпадения нет, выполняем интерполяцию
44                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
45            }
46        }
47        return Double.NaN;
48    }
49
50    public int getPointsCount() {
51        return points.length;
52    }
}
Boris Suchkov (14 hours ago) | Q | Ln 41, Col 10 | Spaces: 4 | UFT-8 | CRLF | {} Java | Finish Setup | 8-9 Go Live | ⚡ Quokka | ⚡ Premier
```

```
ArrayTabulatedFunction.java M ✘
src > functions > ArrayTabulatedFunction.java > ArrayTabulatedFunction > ArrayTabulatedFunction<(FunctionPoint[])
5   public class ArrayTabulatedFunction implements TabulatedFunction {
91     public int getPointsCount() {
92       return points.length;
93     }
94
95     private void checkIndex(int index) {
96       if (index < 0 || index >= points.length) {
97         throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds");
98       }
99     }
100
101    public FunctionPoint getPoint(int index) {
102      checkIndex(index);
103      return new FunctionPoint(points[index]);
104    }
105
106    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
107      checkIndex(index);
108      double newX = point.getX();
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 (newX <= leftBound || newX >= rightBound) {
113        throw new InappropriateFunctionPointException(message: "New point's X coordinate is out of the allowed interval");
114      }
115      points[index] = new FunctionPoint(point);
116    }
117
118    public double getPointX(int index) {
119      checkIndex(index);
120      return points[index].getX();
121    }
122
123    public void setPointX(int index, double x) throws InappropriateFunctionPointException {
124      checkIndex(index);
125      double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
126      double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
127
128      if (x <= leftBound || x >= rightBound) {
129        throw new InappropriateFunctionPointException(message: "New X coordinate is out of the allowed interval");
130      }
131      points[index].setX(x);
132    }
133
134    public double getPointY(int index) {
135      checkIndex(index);
136      return points[index].getY();
137    }
138
139    public void setPointY(int index, double y) {
140      checkIndex(index);
141      points[index].setY(y);
142    }
143
144    public void deletePoint(int index) {
145      checkIndex(index);
146      if (points.length < 3) {
147        throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
148      }
149      FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
150      System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, index);
151      System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
152      points = newPoints;
153    }
154
155    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
156      int insertIndex = 0;
157      while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) {
158        insertIndex++;
159      }
160
161      if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) {
162        throw new InappropriateFunctionPointException("Point with X" + point.getX() + " already exists.");
163      }
164
165      FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
166      System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, insertIndex);
167      newPoints[insertIndex] = new FunctionPoint(point);
168      System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
169      points = newPoints;
170    }
171
172    public String toString() {
173      StringBuilder sb = new StringBuilder();
174      sb.append(str: "(");
175      for (int i = 0; i < points.length; i++) {
176        sb.append(points[i].toString());
177        if (i < points.length - 1) {
178          sb.append(str: ", ");
179        }
180      }
181    }
182  }
```

```
ArrayTabulatedFunction.java M ✘
src > functions > ArrayTabulatedFunction.java > ArrayTabulatedFunction > ArrayTabulatedFunction<(FunctionPoint[])
5   public class ArrayTabulatedFunction implements TabulatedFunction {
134     public double getPointY(int index) {
135       return points[index].getY();
136     }
137
138     public void setPointY(int index, double y) {
139       checkIndex(index);
140       points[index].setY(y);
141     }
142
143     public void deletePoint(int index) {
144       checkIndex(index);
145       if (points.length < 3) {
146         throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
147       }
148       FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
149       System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, index);
150       System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
151       points = newPoints;
152     }
153
154     public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
155       int insertIndex = 0;
156       while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) {
157         insertIndex++;
158       }
159
160       if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) {
161         throw new InappropriateFunctionPointException("Point with X" + point.getX() + " already exists.");
162       }
163
164       FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
165       System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, insertIndex);
166       newPoints[insertIndex] = new FunctionPoint(point);
167       System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
168       points = newPoints;
169     }
170
171     public String toString() {
172       StringBuilder sb = new StringBuilder();
173       sb.append(str: "(");
174       for (int i = 0; i < points.length; i++) {
175         sb.append(points[i].toString());
176         if (i < points.length - 1) {
177           sb.append(str: ", ");
178         }
179       }
180     }
181   }
```

```

1  // 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 }

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 нерезультирующих операций и заметно скажется на скорости выполнения программы.

```
1  LinkedListTabulatedFunction.java M X
src > functions > & LinkedListTabulatedFunction.java > ...
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 --
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("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        for (int i = 0; i < pointsCount; i++) {
39            try {
40                this.addPoint(new FunctionPoint(leftX + i * step, y: 0));
41            } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
42        }
43    }
44
45    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
46        if (leftX >= rightX || values.length < 2) {
47            throw new IllegalArgumentException("Invalid arguments for function creation");
48        }
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        public LinkedListTabulatedFunction(FunctionPoint[] points) {
63            if (points.length < 2) {
64                throw new IllegalArgumentException("Function must have at least 2 points");
65            }
66            this.count = 0;
67            this.head = new FunctionNode();
68            this.head.prev = head;
69            this.head.next = head;
70
71            for (int i = 0; i < points.length; ++i) {
72                if (i > 0 && points[i-1].getX() > points[i].getX()) {
73                    throw new IllegalArgumentException("Points are not sorted by X");
74                }
75                try {
76                    this.addPoint(new FunctionPoint(points[i]));
77                } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
78            }
79        }
80
81    // -- Внутренние методы --
82    private FunctionNode getNodeByIndex(int index) {
83        if (index < 0 || index >= count) {
84            throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds");
85        }
86        FunctionNode current;
87        if (index < count / 2) {
88            current = head.next;
89            for (int i = 0; i < index; i++) {
90                current = current.next;
91            }
92        }
93    }
94}
```

```
1  LinkedListTabulatedFunction.java M X
src > functions > & LinkedListTabulatedFunction.java > ...
8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
45    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
46        if (leftX >= rightX || values.length < 2) {
47            throw new IllegalArgumentException("Invalid arguments for function creation");
48        }
49        this.count = 0;
50        this.head = new FunctionNode();
51        this.head.prev = head;
52        this.head.next = head;
53
54        double step = (rightX - leftX) / (values.length - 1);
55        for (int i = 0; i < values.length; i++) {
56            try {
57                this.addPoint(new FunctionPoint(leftX + i * step, values[i]));
58            } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
59        }
60
61        public LinkedListTabulatedFunction(FunctionPoint[] points) {
62            if (points.length < 2) {
63                throw new IllegalArgumentException("Function must have at least 2 points");
64            }
65            this.count = 0;
66            this.head = new FunctionNode();
67            this.head.prev = head;
68            this.head.next = head;
69
70            for (int i = 0; i < points.length; ++i) {
71                if (i > 0 && points[i-1].getX() > points[i].getX()) {
72                    throw new IllegalArgumentException("Points are not sorted by X");
73                }
74                try {
75                    this.addPoint(new FunctionPoint(points[i]));
76                } catch (InappropriateFunctionPointException e) { /* Невозможно */ }
77            }
78        }
79
80    // -- Внутренние методы --
81    private FunctionNode getNodeByIndex(int index) {
82        if (index < 0 || index >= count) {
83            throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds");
84        }
85        FunctionNode current;
86        if (index < count / 2) {
87            current = head.next;
88            for (int i = 0; i < index; i++) {
89                current = current.next;
90            }
91        }
92    }
93}
```

```
 1  LinkedListTabulatedFunction.java M X
src > functions > LinkedListTabulatedFunction.java ...
8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
9
10     private FunctionNode getNodeByIndex(int index) {
11         FunctionNode current = head.next;
12         for (int i = 0; i < index; i++) {
13             current = current.next;
14         }
15     } else {
16         current = head.prev;
17         for (int i = count - 1; i > index; i--) {
18             current = current.prev;
19         }
20     }
21     return current;
22 }
23
24     private FunctionNode addNode(FunctionNode node) {
25         FunctionNode newNode = new FunctionNode();
26         newNode.prev = node.prev;
27         newNode.next = node;
28         node.prev.next = newNode;
29         node.prev = newNode;
30         count++;
31         return newNode;
32     }
33
34     private FunctionNode deleteNode(FunctionNode node) {
35         node.prev.next = node.next;
36         node.next.prev = node.prev;
37         count--;
38         return node;
39     }
40
41 // --- ОСНОВНЫЕ МЕТОДЫ ---
42
43     public double getLeftDomainBorder() {
44         return head.next.point.getX();
45     }
46
47     public double getRightDomainBorder() {
48         return head.prev.point.getX();
49     }
50
51 // --- ИСПРАВЛЕННЫЙ МЕТОД (Задание 2) ---
52     public double getFunctionValue(double x) {
53         if (x < getLeftDomainBorder() || x > getRightDomainBorder() || count == 0) {
54             return Double.NaN;
55         }
56
57         FunctionNode current = head.next;
58         while (current != head) {
59             double epsilon = 1e-9; // Машинный эпсилон
60
61             // Проверка: если x совпадает с точкой текущего узла
62             if (Math.abs(current.point.getX() - x) < epsilon) {
63                 return current.point.getY();
64             }
65
66             // Проверяем интервал между текущим и следующим узлом
67             if (current.next != head) {
68                 double x1 = current.point.getX();
69                 double x2 = current.next.point.getX();
70
71                 // Если x строго внутри интервала
72                 if (x > x1 && x < x2) {
73                     // Проверяем правую границу на всякий случай, чтобы избежать интерполяции при x == x2
74                     if (Math.abs(x2 - x) < epsilon) {
75                         return current.next.point.getY();
76                     }
77
78                     // Если точного совпадения нет, делаем интерполяцию
79                     double y1 = current.point.getY();
80                     double y2 = current.next.point.getY();
81                     return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
82                 }
83             }
84             current = current.next;
85         }
86         return Double.NaN;
87     }
88
89     public int getPointsCount() {
90         return count;
91     }
92
93     private void checkIndex(int index) {
94         if (index < 0 || index > count) {
95             throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds [0, " + (count - 1) + "]");
96         }
97     }
98
99     public FunctionPoint getPoint(int index) {
100         checkIndex(index);
101     }
102 }
```

```
 1  LinkedListTabulatedFunction.java M X
src > functions > LinkedListTabulatedFunction.java ...
8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
9
10     public double getFunctionValue(double x) {
11
12         FunctionNode current = head.next;
13         while (current != head) {
14             double epsilon = 1e-9; // Машинный эпсилон
15
16             // Проверка: если x совпадает с точкой текущего узла
17             if (Math.abs(current.point.getX() - x) < epsilon) {
18                 return current.point.getY();
19             }
20
21             // Проверяем интервал между текущим и следующим узлом
22             if (current.next != head) {
23                 double x1 = current.point.getX();
24                 double x2 = current.next.point.getX();
25
26                 // Если x строго внутри интервала
27                 if (x > x1 && x < x2) {
28                     // Проверяем правую границу на всякий случай, чтобы избежать интерполяции при x == x2
29                     if (Math.abs(x2 - x) < epsilon) {
30                         return current.next.point.getY();
31                     }
32
33                     // Если точного совпадения нет, делаем интерполяцию
34                     double y1 = current.point.getY();
35                     double y2 = current.next.point.getY();
36                     return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
37                 }
38             }
39             current = current.next;
40         }
41         return Double.NaN;
42     }
43
44     public int getPointsCount() {
45         return count;
46     }
47
48     private void checkIndex(int index) {
49         if (index < 0 || index > count) {
50             throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds [0, " + (count - 1) + "]");
51         }
52     }
53
54     public FunctionPoint getPoint(int index) {
55         checkIndex(index);
56     }
57 }
```

```
src > functions > ↵ LinkedListTabulatedFunction.java M X
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);
225    }
226
227    public void writeExternal(ObjectOutput out) throws IOException {
228        out.writeInt(count);
229        FunctionNode current = head.next;
230        while(current != head) {
231            out.writeObject(current.point);
232            current = current.next;
233        }
234    }
235
236    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
237        int readCount = in.readInt();
238        this.head.prev = head;
239        this.head.next = head;
240        this.count = 0;
241
242        for (int i = 0; i < readCount; i++) {
243            FunctionPoint point = (FunctionPoint) in.readObject();
244            try {
245                this.addPoint(point);
246            } catch (InappropriateFunctionPointException e) {
247                throw new IOException("Failed to deserialize: " + e.getMessage());
248            }
249        }
250    }
251
252    // --- Методы Externalizable ---
253    public void writeExternal(ObjectOutput out) throws IOException {
254        out.writeInt(count);
255        FunctionNode current = head.next;
256        while(current != head) {
257            out.writeObject(current.point);
258            current = current.next;
259        }
260    }
261
262    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
263        int readCount = in.readInt();
264        this.head.prev = head;
265        this.head.next = head;
266        this.count = 0;
267
268        for (int i = 0; i < readCount; i++) {
269            FunctionPoint point = (FunctionPoint) in.readObject();
270            try {
271                this.addPoint(point);
272            } catch (InappropriateFunctionPointException e) {
273                throw new IOException("Failed to deserialize: " + e.getMessage());
274            }
275        }
276    }
277
278    // --- НОВЫЕ МЕТОДЫ ---
279}
```

```
src > functions > ↵ LinkedListTabulatedFunction.java M X
8  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
233    public void deletePoint(int index) {
234        checkIndex(index);
235        if (count < 3) {
236            throw new IllegalStateException("Cannot delete point: function must have at least 2 points remaining.");
237        }
238        deleteNode(getNodeByIndex(index));
239    }
240
241    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
242        FunctionNode current = head.next;
243        while(current != head && current.point.getX() < point.getX()) {
244            current = current.next;
245        }
246        if(current != head && current.point.getX() == point.getX()) {
247            throw new InappropriateFunctionPointException("Point with X=" + point.getX() + " already exists.");
248        }
249        addNode(current).point = new FunctionPoint(point);
250    }
251
252    // --- Методы Externalizable ---
253    public void writeExternal(ObjectOutput out) throws IOException {
254        out.writeInt(count);
255        FunctionNode current = head.next;
256        while(current != head) {
257            out.writeObject(current.point);
258            current = current.next;
259        }
260    }
261
262    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
263        int readCount = in.readInt();
264        this.head.prev = head;
265        this.head.next = head;
266        this.count = 0;
267
268        for (int i = 0; i < readCount; i++) {
269            FunctionPoint point = (FunctionPoint) in.readObject();
270            try {
271                this.addPoint(point);
272            } catch (InappropriateFunctionPointException e) {
273                throw new IOException("Failed to deserialize: " + e.getMessage());
274            }
275        }
276    }
277
278    // --- НОВЫЕ МЕТОДЫ ---
279}
```

```
 8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
269 |     // --- HOME METHODS ---
270 |
271     public String toString() {
272         StringBuilder sb = new StringBuilder();
273         sb.append(str: "(");
274         FunctionNode current = head.next;
275         while (current != head) {
276             sb.append(current.point.toString());
277             if (current.next != head) {
278                 sb.append(str: ", ");
279             }
280             current = current.next;
281         }
282         sb.append(str: ")");
283         return sb.toString();
284     }
285
286     public boolean equals(Object o) {
287         if (this == o) return true;
288         if (!(o instanceof TabulatedFunction)) return false;
289
290         TabulatedFunction that = (TabulatedFunction) o;
291
292         if (this.getPointsCount() != that.getPointsCount()) return false;
293
294         if (o instanceof LinkedListTabulatedFunction) {
295             LinkedListTabulatedFunction thatList = (LinkedListTabulatedFunction) o;
296             FunctionNode thisCurrent = this.head.next;
297             FunctionNode thatCurrent = thatList.head.next;
298
299             while (thisCurrent != this.head) {
300                 if (!thisCurrent.point.equals(thatCurrent.point)) {
301                     return false;
302                 }
303                 thisCurrent = thisCurrent.next;
304                 thatCurrent = thatCurrent.next;
305             }
306         } else {
307             for (int i = 0; i < this.getPointsCount(); i++) {
308                 if (!this.getPoint(i).equals(that.getPoint(i))) {
309                     return false;
310                 }
311             }
312         }
313
314         return true;
315     }
316
317     public int hashCode() {
318         int result = 1;
319         FunctionNode current = head.next;
320         while (current != head) {
321             result = 31 * result + current.point.hashCode();
322             current = current.next;
323         }
324         result = 31 * result + Integer.hashCode(count);
325         return result;
326     }
327
328     public Object clone() throws CloneNotSupportedException {
329         FunctionPoint[] clonedPoints = new FunctionPoint[this.count];
330         FunctionNode current = this.head.next;
331         for (int i = 0; i < this.count; i++) {
332             clonedPoints[i] = (FunctionPoint) current.point.clone();
333             current = current.next;
334         }
335         return new LinkedListTabulatedFunction(clonedPoints);
336     }
337 }
```

Screenshot of an IDE showing the implementation of the `equals` method for `LinkedListTabulatedFunction`. The code compares two instances of `LinkedListTabulatedFunction` by either traversing their linked lists point-by-point or comparing the points directly if the other instance is of type `LinkedListTabulatedFunction`. The IDE interface includes tabs for 'src > functions > LinkedListTabulatedFunction.java', status bar with file info like 'Boris Suchkov (1d hours ago)', and various toolbars.

```
 8 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
269 |     // --- HOME METHODS ---
270 |
271     public String toString() {
272         StringBuilder sb = new StringBuilder();
273         sb.append(str: "(");
274         FunctionNode current = head.next;
275         while (current != head) {
276             sb.append(current.point.toString());
277             if (current.next != head) {
278                 sb.append(str: ", ");
279             }
280             current = current.next;
281         }
282         sb.append(str: ")");
283         return sb.toString();
284     }
285
286     public boolean equals(Object o) {
287         if (this == o) return true;
288         if (!(o instanceof TabulatedFunction)) return false;
289
290         TabulatedFunction that = (TabulatedFunction) o;
291
292         if (this.getPointsCount() != that.getPointsCount()) return false;
293
294         if (o instanceof LinkedListTabulatedFunction) {
295             LinkedListTabulatedFunction thatList = (LinkedListTabulatedFunction) o;
296             FunctionNode thisCurrent = this.head.next;
297             FunctionNode thatCurrent = thatList.head.next;
298
299             while (thisCurrent != this.head) {
300                 if (!thisCurrent.point.equals(thatCurrent.point)) {
301                     return false;
302                 }
303                 thisCurrent = thisCurrent.next;
304                 thatCurrent = thatCurrent.next;
305             }
306         } else {
307             for (int i = 0; i < this.getPointsCount(); i++) {
308                 if (!this.getPoint(i).equals(that.getPoint(i))) {
309                     return false;
310                 }
311             }
312         }
313
314         return true;
315     }
316
317     public int hashCode() {
318         int result = 1;
319         FunctionNode current = head.next;
320         while (current != head) {
321             result = 31 * result + current.point.hashCode();
322             current = current.next;
323         }
324         result = 31 * result + Integer.hashCode(count);
325         return result;
326     }
327
328     public Object clone() throws CloneNotSupportedException {
329         FunctionPoint[] clonedPoints = new FunctionPoint[this.count];
330         FunctionNode current = this.head.next;
331         for (int i = 0; i < this.count; i++) {
332             clonedPoints[i] = (FunctionPoint) current.point.clone();
333             current = current.next;
334         }
335         return new LinkedListTabulatedFunction(clonedPoints);
336     }
337 }
```

Screenshot of an IDE showing the implementation of `hashCode` and `clone` methods for `LinkedListTabulatedFunction`. The `hashCode` method uses a simple rolling hash of point hash codes. The `clone` method creates a new instance with cloned points. The IDE interface is similar to the first screenshot.

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
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 `ArrayListTabulatedFunction` 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-克隆ы не изменились.

```
>Main.java U X
src > Main.java > ...
1 import functions.*;
2
3 public class Main {
4     Run|Debug
5     public static void main(String[] args) {
6
7         // --- Подготовка ---
8         FunctionPoint[] points = {
9             new FunctionPoint(x: 0, y: 0),
10            new FunctionPoint(x: 1, y: 1),
11            new FunctionPoint(x: 2, y: 4),
12            new FunctionPoint(x: 3, y: 9)
13        };
14
15        FunctionPoint[] points2 = {
16            new FunctionPoint(x: 0, y: 0),
17            new FunctionPoint(x: 1, y: 1),
18            new FunctionPoint(x: 2, y: 4),
19            new FunctionPoint(x: 3, y: 9)
20        };
21
22        FunctionPoint[] points3 = { // Различающийся
23            new FunctionPoint(x: 0, y: 0),
24            new FunctionPoint(x: 1, y: 2), // <--- Различие
25            new FunctionPoint(x: 2, y: 5),
26            new FunctionPoint(x: 3, y: 10)
27        };
28
29        // Создаем объекты двух типов
30        TabulatedFunction arrayFunc = new ArrayTabulatedFunction(points);
31        TabulatedFunction listFunc = new LinkedListTabulatedFunction(points);
32
33        System.out.println(x: "--- Задание 5.1: Проверка toString() ---");
34        System.out.println("Array: " + arrayFunc.toString());
35        System.out.println("List: " + listFunc.toString());
36
37        System.out.println(x: "\n--- Задание 5.2: Проверка equals() ---");
38        TabulatedFunction arrayFuncEq = new ArrayTabulatedFunction(points2);
39        TabulatedFunction listFuncEq = new LinkedListTabulatedFunction(points2);
40
```

```
>Main.java U X
src > Main.java > ...
3 public class Main {
4     public static void main(String[] args) {
5
6         TabulatedFunction listFuncEq = new LinkedListTabulatedFunction(points2);
7         TabulatedFunction arrayFuncDiff = new ArrayTabulatedFunction(points3);
8
9         System.out.println("arrayFunc.equals(arrayFuncEq): " + arrayFunc.equals(arrayFuncEq)); // true
10        System.out.println("listFunc.equals(listFuncEq): " + listFunc.equals(listFuncEq)); // true
11        System.out.println("arrayFunc.equals(listFuncEq): " + arrayFunc.equals(listFuncEq)); // true
12        System.out.println("listFunc.equals(arrayFuncEq): " + listFunc.equals(arrayFuncEq)); // true
13        System.out.println("arrayFunc.equals(arrayFuncDiff): " + arrayFunc.equals(arrayFuncDiff)); // false
14        System.out.println("arrayFunc.equals(null): " + arrayFunc.equals(obj: null)); // false
15
16        System.out.println(x: "\n--- Задание 5.3: Проверка hashCode() ---");
17        System.out.println("arrayFunc.hashCode(): " + arrayFunc.hashCode());
18        System.out.println("arrayFuncEq.hashCode(): " + arrayFuncEq.hashCode());
19        System.out.println("listFunc.hashCode(): " + listFunc.hashCode());
20        System.out.println("listFuncEq.hashCode(): " + listFuncEq.hashCode());
21        System.out.println("arrayFuncDiff.hashCode(): " + arrayFuncDiff.hashCode());
22
23        try {
24            // Изменяем одну точку (тут setPointX может выбросить исключение, поэтому catch нужен)
25            arrayFunc.setPointX(index: 1, x: 1.0001);
26        } catch (InappropriateFunctionPointException e) {
27            e.printStackTrace();
28        }
29        System.out.println("arrayFunc (измененный) hashCode(): " + arrayFunc.hashCode());
30
31        // Восстановим arrayFunc для теста клонирования
32        try {
33            arrayFunc.setPointX(index: 1, x: 1.0);
34        } catch (InappropriateFunctionPointException e) { e.printStackTrace(); }
35
36
37        System.out.println(x: "\n--- Задание 5.4: Проверка clone() (Глубокое клонирование) ---");
38        try {
39            // Клонируем arrayFunc
40            ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc.clone();
41
42            // 1. Идентичном ОРИГИНАЛ
43
```

The screenshot shows a Java development environment with the following details:

- Main.java** is the active file, containing the following code:

```

1  public class Main {
2      public static void main(String[] args) {
3          // Восстановим arrayFunc для теста клонирования
4          try {
5              arrayFunc.setPointX(index: 1, x: 1.0);
6          } catch (InappropriateFunctionPointException e) { e.printStackTrace(); }
7
8          System.out.println(x: "\n--- Задание 5.4: Проверка clone() (Глубокое клонирование) ---");
9          try {
10              // Клонируем arrayFunc
11              ArrayTabulatedFunction arrayClone = (ArrayTabulatedFunction) arrayFunc.clone();
12
13              // 1. Изменяем ОРИГИНАЛ
14              arrayFunc.setPointY(index: 1, y: 100.0);
15
16              // 2. Печатаем ОБА
17              System.out.println("Оригинал Array (изменён): " + arrayFunc.toString());
18              System.out.println("Клон Array (не изменён): " + arrayClone.toString());
19
20              // Клонируем listFunc
21              LinkedListTabulatedFunction listClone = (LinkedListTabulatedFunction) listFunc.clone();
22
23              // 1. Изменяем ОРИГИНАЛ
24              listFunc.setPointY(index: 1, y: 100.0);
25
26              // 2. Печатаем ОБА
27              System.out.println("Оригинал List (изменён): " + listFunc.toString());
28              System.out.println("Клон List (не изменён): " + listClone.toString());
29
30          } catch (CloneNotSupportedException e) {
31              e.printStackTrace();
32          }
33      }
34  }

```

- Execution Output:**

```

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\d095eae8d3bb290a020d048fa721cled\redhat.java\dt_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)}}

```