

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
Кафедра технической кибернетики

Отчет по лабораторной работе № 3

Дисциплина: «ООП»

Тема «Исключения и интерфейсы»

Выполнил: Сучков Борис Антонович

Группа: 6201-120303D

Самара 2025

Задание на лабораторную работу

Дополнить пакет для работы с функциями одной переменной, заданными в табличной форме, добавив классы исключений, новый класс функций и базовый интерфейс.

В ходе выполнения работы запрещено использовать классы из пакета `java.util`.

Задания

Задание 1

Ознакомиться (изучить документацию) со следующими классами исключений, входящих в API Java:

• `java.lang.Exception` • `java.lang.IndexOutOfBoundsException` • `java.lang.ArrayIndexOutOfBoundsException` • `java.lang.IllegalArgumentException` • `java.lang.IllegalStateException`

Задание 2

В пакете `functions` создать два класса исключений:

- `FunctionPointIndexOutOfBoundsException` – исключение выхода за границы набора точек при обращении к ним по номеру, наследует от класса `IndexOutOfBoundsException`;
- `InappropriateFunctionPointException` – исключение, выбрасываемое при попытке добавления или изменения точки функции несоответствующим образом, наследует от класса `Exception`.

При создании классов исключений настоятельно рекомендуется использовать специализированные средства среды разработки.

Задание 3

В разработанный ранее класс `TabulatedFunction` внести изменения, обеспечивающие выбрасывание исключений методами класса.

- Оба конструктора класса должны выбрасывать исключение `IllegalArgumentException`, если левая граница области определения больше или равна правой, а также если предлагаемое количество точек меньше двух. Это обеспечит создание объекта только при корректных параметрах.
-

Методы `getPoint()`, `setPoint()`, `getPointX()`, `setPointX()`, `getPointY()`, `setPointY()` и `deletePoint()` должны выбрасывать исключение `FunctionPointIndexOutOfBoundsException`,

если переданный в метод номер выходит за границы набора точек. Это обеспечит корректность обращений к точкам функции.

- Методы `setPoint()` и `setPointX()` должны выбрасывать исключение `InappropriateFunctionPointException` в том случае, если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции. Метод `addPoint()` также должен выбрасывать исключение `InappropriateFunctionPointException`, если в наборе точек функции есть точка, абсцисса которой совпадает с абсциссой добавляемой точки. Это обеспечит сохранение упорядоченности точек функции.

- Метод `deletePoint()` должен выбрасывать исключение `IllegalStateException`, если на момент удаления точки количество точек в наборе менее трех. Это обеспечит невозможность получения функции с некорректным количеством точек.

Задание 4

В пакете `functions` создать класс `LinkedListTabulatedFunction`, объект которого также должен описывать табулированную функцию. Отличие этого класса должно заключаться в том, что для хранения набора точек в нем должен использоваться не массив, а динамическая структура – связный список.

Важно понимать, что представляет собой связный список в объектноориентированном программировании. Действительно, в процедурных языках список обычно представляет собой набор записей (или структур) в памяти, некоторые элементы которых являются указателями на соседние элементы списка, а также набор процедур по работе со списком. В Java, во-первых, нет указателей и адресной арифметики, вместо этого будут использоваться ссылки. Во-вторых, вместо записей будут использоваться объекты. В-третьих, процедуры по работе со списком будут заменены на методы класса.

Особенность заключается в том, что для полноценного описания связного списка потребуется реализовать два класса. Ответственность первого из них – элемент списка и его связи, объекты этого класса являются объектами элементов списка, хранящими данные и ссылки на соседние элементы. Ответственность второго – список в целом и операции с ним, именно в нем реализуются методы по манипулированию списком, а его объект – это объект списка целиком.

Таким образом, в первом классе должны присутствовать информационное поле и поля связи. А во втором классе должна храниться ссылка на объект головы списка и должны быть описаны методы для работы со списком, причем публичные методы не должны получать или возвращать ссылки на объекты элементов списка, т.к. это будет нарушением инкапсуляции. Можно сказать, что между этими классами имеется отношение композиции: объекты элементов списка являются частями объекта списка и не существуют вне его.

При написании «внешнего» класса следует учесть, что инкапсуляция работы со списком в отдельный объект позволяет несколько изменить алгоритмы по работе со списком по сравнению с обычными алгоритмами в процедурном исполнении. Так, не имеет смысла подсчитывать каждый раз длину списка (ведь никто кроме самого объекта списка не может его изменить), вместо этого проще хранить ее как поле «внешнего» объекта. Также очевидно, что часто работа с элементами списка ведется последовательно или в соседних элементах, поэтому пробегать каждый раз от головы при доступе к элементу списка неразумно, проще хранить ссылку и номер элемента, к которому было последнее обращение, и в некоторых случаях двигаться от него.

В качестве структуры списка в настоящей работе требуется использовать двусвязный циклический список с выделенной головой. Первое означает, что объект элемента списка хранит две ссылки – на предыдущий элемент и на следующий элемент. Второе означает, что голова списка не используется для хранения данных и присутствует в списке всегда (пустой список представляет собой объект головы, ссылающийся сам на себя и как на предыдущий элемент, и как на следующий).

Класс `LinkedListTabulatedFunction` совмещает в себе две функции: с одной стороны, он будет описывать связный список и работу с ним, а с другой стороны, он будет описывать работу с табулированной функцией и ее точками. Для реализации первой функции требуется выполнить следующие действия.

1. Описать класс элементов списка `FunctionNode`, содержащий информационное поле для хранения данных типа `FunctionPoint`, а также поля для хранения ссылок на предыдущий и следующий элемент. Выберите и обоснуйте место описания класса и его видимость. Также выберите и обоснуйте реализацию инкапсуляции в этом классе.
2. Описать класс `LinkedListTabulatedFunction` объектов списка, содержащий поле ссылки на объект головы, а также иные вспомогательные поля.
3. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode getNodeByIndex(int index)`, возвращающий ссылку на объект элемента списка по его номеру. Нумерация значащих элементов (голова списка в данном случае к ним не относится) должна начинаться с 0. Метод должен обеспечивать оптимизацию доступа к элементам списка.
4. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode addNodeToTail()`, добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента.
5. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode addNodeByIndex(int index)`, добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента.

6. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode deleteNodeByIndex(int index)`, удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента.

Задание 5

Для обеспечения второй функции класса `LinkedListTabulatedFunction` реализовать в классе конструкторы и методы, аналогичные конструкторам и методам класса `TabulatedFunction`. Конструкторы должны иметь те же параметры, методы должны иметь те же сигнатуры. Также должны выбрасываться те же виды исключений в тех же случаях.

С одной стороны, при написании методов следует использовать уже написанные методы, отвечающие за работу со связным списком. С другой стороны, инкапсуляция в одном классе и списка, и табулированной функции позволяет в некоторых методах, относящихся к табулированной функции, значительно оптимизировать работу за счет возможности обращаться к элементам списка напрямую. Определите такие методы и оптимизируйте их.

Задание 6

Класс `TabulatedFunction` переименовать в класс `ArrayTabulatedFunction`. Создать интерфейс `TabulatedFunction`, содержащий объявления общих методов классов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`.

Сделать так, чтобы оба класса функций реализовывали созданный интерфейс.

Теперь суть работы с табулированными функциями заключена в типе интерфейса, а в классах заключена только реализация этой работы.

Задание 7

Проверить работу написанных классов.

Для этого в созданном ранее классе `Main`, содержащем точку входа программы, добавить проверку для случаев, в которых объект табулированной функции должен выбрасывать исключения.

Ссылочную переменную для работы с объектом функции следует объявить типа `TabulatedFunction`, а при создании объекта следует указать реальный класс. Тогда проверка работы класса `LinkedListTabulatedFunction` может быть произведена путем простой замены класса, объект которого создается.

ЗАДАНИЕ 1

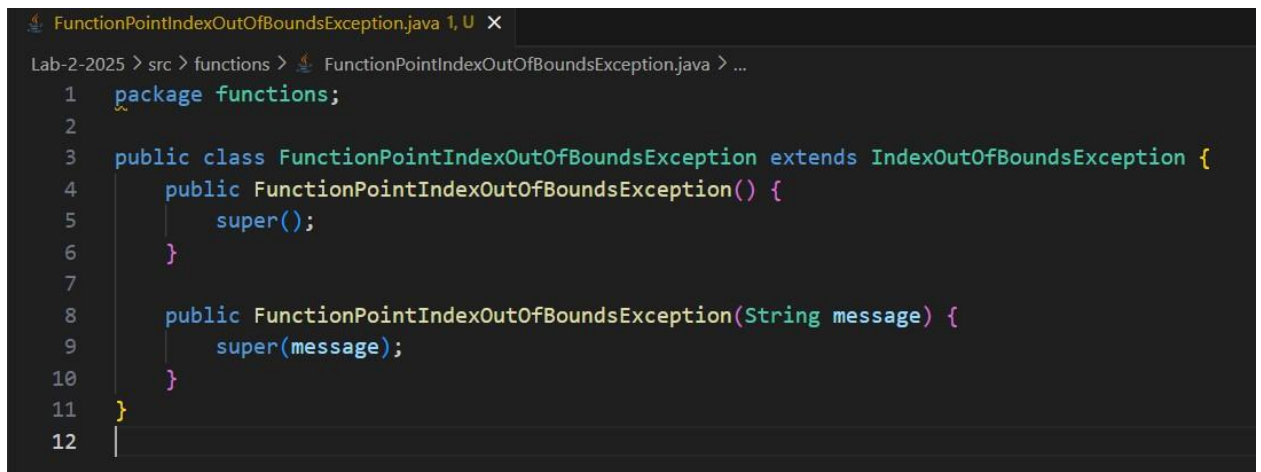
Я ознакомился (изучил документацию) со следующими классами исключений, входящих в API Java:

- java.lang.Exception • java.lang.IndexOutOfBoundsException • java.lang.ArrayIndexOutOfBoundsException • java.lang.IllegalArgumentException • java.lang.IllegalStateException

ЗАДАНИЕ 2

В пакете functions я создал два класса исключений:

- FunctionPointIndexOutOfBoundsException – исключение выхода за границы набора точек при обращении к ним по номеру, наследует от класса IndexOutOfBoundsException;
- InappropriateFunctionPointException – исключение, выбрасываемое при попытке добавления или изменения точки функции несоответствующим образом, наследует от класса Exception.



```
FunctionPointIndexOutOfBoundsException.java 1, U X
Lab-2-2025 > src > functions > FunctionPointIndexOutOfBoundsException.java > ...
1 package functions;
2
3 public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException {
4     public FunctionPointIndexOutOfBoundsException() {
5         super();
6     }
7
8     public FunctionPointIndexOutOfBoundsException(String message) {
9         super(message);
10    }
11 }
12
```

```
InappropriateFunctionPointException.java 1, U X
Lab-2-2025 > src > functions > InappropriateFunctionPointException.java > ...
1 package functions;
2
3 public class InappropriateFunctionPointException extends Exception {
4     public InappropriateFunctionPointException() {
5         super();
6     }
7
8     public InappropriateFunctionPointException(String message) {
9         super(message);
10    }
11 }
12
```

ЗАДАНИЕ 3

В разработанный ранее класс `TabulatedFunction` я внёс изменения, обеспечивающие выбрасывание исключений методами класса.

- Оба конструктора класса выбрасывают исключение `IllegalArgumentException`, если левая граница области определения больше или равна правой, а также если предлагаемое количество точек меньше двух. Это обеспечивает создание объекта только при корректных параметрах.
 - Методы `getPoint()`, `setPoint()`, `getPointX()`, `setPointX()`, `getPointY()`, `setPointY()` и `deletePoint()` выбрасывают исключение `FunctionPointIndexOutOfBoundsException`, если переданный в метод номер выходит за границы набора точек. Это обеспечивает корректность обращений к точкам функции.
 - Методы `setPoint()` и `setPointX()` выбрасывают исключение `InappropriateFunctionPointException` в том случае, если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции. Метод `addPoint()` также выбрасывает исключение `InappropriateFunctionPointException`, если в наборе точек функции есть точка, абсцисса которой совпадает с абсциссой добавляемой точки. Это обеспечивает сохранение упорядоченности точек функции.
- Метод `deletePoint()` выбрасывает исключение `IllegalStateException`, если на момент удаления точки количество точек в наборе менее трех. Это обеспечивает невозможность получения функции с некорректным количеством точек.

```

1 // TabulatedFunction.h
2 #ifndef TABULATEDFUNCTION_H
3 #define TABULATEDFUNCTION_H
4
5 #include <vector>
6 #include <stdexcept>
7 #include <string>
8
9 namespace functions {
10
11     class TabulatedFunction {
12     private:
13         struct FunctionPoint {
14             double x;
15             double y;
16         };
17         std::vector<FunctionPoint> points;
18
19     public:
20         // Constructors
21         TabulatedFunction(double leftX, double rightX, int pointCount) {
22             // Initialize function with points
23             if (leftX >= rightX || pointCount < 2) {
24                 throw std::invalid_argument("Invalid arguments for function creation");
25             }
26             this->points = new FunctionPoint[pointCount];
27             double step = (rightX - leftX) / (pointCount - 1);
28             for (int i = 0; i < pointCount; ++i) {
29                 points[i] = new FunctionPoint(leftX + i * step, 0);
30             }
31         }
32
33         TabulatedFunction(double leftX, double rightX, double values) {
34             int count = values.length;
35             // Initialize function with values
36             if (leftX >= rightX || count < 2) {
37                 throw std::invalid_argument("Invalid arguments for function creation");
38             }
39             this->points = new FunctionPoint[count];
40             double step = (rightX - leftX) / (count - 1);
41             for (int i = 0; i < count; ++i) {
42                 points[i] = new FunctionPoint(leftX + i * step, values[i]);
43             }
44         }
45
46         // Accessors
47         public: double getLeftX() const {
48             return points[0].x;
49         }
50
51         public: double getRightX() const {
52             return points[points.length - 1].x;
53         }
54
55         public: double getFunctionValue(double x) {
56             if (x < getLeftX() || x > getRightX()) {
57                 return Double.NaN;
58             }
59             for (int i = 0; i < points.length - 1; ++i) {
60                 if (points[i].x <= x && x <= points[i + 1].x) {
61                     double x1 = points[i].x;
62                     double y1 = points[i].y;
63                     double x2 = points[i + 1].x;
64                     double y2 = points[i + 1].y;
65                     return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
66                 }
67             }
68             return Double.NaN;
69         }
70
71         // Methods
72         public: int getPointCount() {
73             return points.length;
74         }
75
76     private: void checkIndex(int index) {
77         if (index < 0 || index >= points.length) {
78             throw std::out_of_range("Index is out of bounds");
79         }
80     }
81
82     public: FunctionPoint getPoint(int index) {
83         checkIndex(index);
84         return new FunctionPoint(points[index]);
85     }
86
87     public: void setPoint(int index, FunctionPoint point) throws IllegalArgumentException {
88         checkIndex(index);
89         double newX = point.x;
90         double leftBound = (index > 0) ? points[index - 1].x : Double.NEGATIVE_INFINITY;
91         double rightBound = (index < points.length - 1) ? points[index + 1].x : Double.POSITIVE_INFINITY;
92         if (newX < leftBound || newX > rightBound) {
93             throw new IllegalArgumentException("X coordinate is out of the allowed interval");
94         }
95         points[index] = new FunctionPoint(point);
96     }
97
98     public: double getPointX(int index) {
99         checkIndex(index);
100         return points[index].x;
101     }
102
103     public: void setPointX(int index, double x) throws IllegalArgumentException {
104         checkIndex(index);
105         double newX = x;
106         double leftBound = (index > 0) ? points[index - 1].x : Double.NEGATIVE_INFINITY;
107         double rightBound = (index < points.length - 1) ? points[index + 1].x : Double.POSITIVE_INFINITY;
108         if (newX < leftBound || newX > rightBound) {
109             throw new IllegalArgumentException("X coordinate is out of the allowed interval");
110         }
111         points[index].setX(x);
112     }
113
114     public: double getPointY(int index) {
115         checkIndex(index);
116         return points[index].y;
117     }
118
119     public: void setPointY(int index, double y) {
120         checkIndex(index);
121         points[index].setY(y);
122     }
123
124     public: void deletePoint(int index) {
125         checkIndex(index);
126         // Remove element from array
127         if (points.length < 2) {
128             throw new IllegalArgumentException("Cannot delete point, function must have at least 2 points");
129         }
130         FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
131         System.arraycopy(points, 0, newPoints, 0, index);
132         System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
133         points = newPoints;
134     }
135
136     public: void addPoint(FunctionPoint point) throws IllegalArgumentException {
137         int insertIndex = 0;
138         while (insertIndex < points.length && points[insertIndex].x < point.x) {
139             insertIndex++;
140         }
141         // Remove element to prevent
142         if (insertIndex < points.length && points[insertIndex].x == point.x) {
143             throw new IllegalArgumentException("Point with such X already exists");
144         }
145         FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
146         System.arraycopy(points, 0, newPoints, 0, insertIndex);
147         newPoints[insertIndex] = new FunctionPoint(point);
148         System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
149         points = newPoints;
150     }
151 }
152
153 }

```

ЗАДАНИЯ 4 И 5

В пакете functions реализован класс `LinkedListTabulatedFunction`, объект которого описывает табулированную функцию. Для хранения набора точек используется динамическая структура данных – двусвязный циклический список с выделенной головой.

Внутри класса `LinkedListTabulatedFunction` реализован вложенный (inner) класс `FunctionNode`, который инкапсулирует данные одного узла списка и хранит ссылки на соседние элементы. Класс содержит поля: `FunctionPoint point` для хранения данных точки функции (значения x и y), `FunctionNode next` – ссылку на следующий элемент списка, и `FunctionNode prev` – ссылку на предыдущий элемент списка. Класс `FunctionNode` объявлен как `private static`, что полностью скрывает его реализацию от внешнего кода и обеспечивает инкапсуляцию. Работа с элементами осуществляется исключительно через методы внешнего класса `LinkedListTabulatedFunction`.

Класс `LinkedListTabulatedFunction` содержит следующие основные поля: `FunctionNode head` – ссылка на объект головы списка (не хранящий данных), при этом в пустом списке `head.next = head` и `head.prev = head`; `int count` – текущее количество элементов (точек) в списке; `FunctionNode current` и `int currentIndex` – поля для оптимизации доступа к элементам списка, позволяющие не начинать поиск всегда от головы.

Реализован метод `FunctionNode getNodeByIndex(int index)`, возвращающий ссылку на узел списка по индексу. Нумерация начинается с 0, голова списка не учитывается. Метод использует оптимизированный алгоритм доступа: если запрошенный индекс ближе к текущему (`currentIndex`), движение начинается от текущего узла; если ближе к голове или хвосту – поиск начинается с соответствующей стороны. Это уменьшает количество переходов между элементами и ускоряет доступ.

Реализован метод `FunctionNode addNodeToTail()`, добавляющий новый элемент в конец списка. Метод создает новый узел, вставляет его перед головой (`head`), обновляет ссылки `prev` и `next`, увеличивает счётчик элементов `count` и возвращает ссылку на созданный узел.

Реализован метод `FunctionNode addNodeByIndex(int index)`, вставляющий новый элемент на указанную позицию. Алгоритм работы следующий: находится узел, который должен следовать после вставляемого, создается новый узел, обновляются связи `prev` и `next` у соседних узлов, увеличивается `count` и возвращается ссылка на вставленный элемент. Метод корректно обрабатывает граничные случаи – вставку в начало и конец списка.

Реализован метод `FunctionNode deleteNodeByIndex(int index)`, удаляющий элемент по индексу. Метод находит удаляемый узел с помощью `getNodeByIndex(index)`, перестраивает связи `prev` и `next` у соседних узлов, уменьшает счётчик `count` и возвращает ссылку на удалённый узел.

Используется двусвязный циклический список с выделенной головой, что упрощает работу с граничными элементами (нет необходимости проверять `null`). Поддерживается полная инкапсуляция – внешние пользователи не имеют доступа к элементам списка напрямую. Хранение `count`, `current` и `currentIndex` обеспечивает высокую эффективность операций добавления, удаления и обращения к элементам. Вся логика манипуляций со списком инкапсулирована внутри класса `LinkedListTabulatedFunction`.

Для обеспечения второй функции класса `LinkedListTabulatedFunction` я реализовал в классе конструкторы и методы, аналогичные конструкторам и методам класса `TabulatedFunction`. Конструкторы имеют те же параметры,

методы имеют те же сигнатуры. Также выбрасываются те же виды исключений в тех же случаях.

С одной стороны, при написании методов используются уже написанные методы, отвечающие за работу со связным списком. С другой стороны, инкапсуляция в одном классе и списка, и табулированной функции позволяет в некоторых методах, относящихся к табулированной функции, значительно оптимизировать работу за счёт возможности обращаться к элементам списка напрямую. Я определил такие методы и оптимизировал их.

```
1 // LinkedListFunction.java v. 0.8
2 package function;
3
4 public class LinkedListFunction {
5     // constructor and init check
6     // no private, but the constructor is private, so it's not visible
7     // no constructor
8     private class FunctionNode {
9         FunctionPoint point;
10        FunctionNode next;
11    }
12
13    private final FunctionNode head; // linked list, no other nodes
14    private int count; // number of nodes, 0 means empty list
15
16    // constructor
17    public LinkedListFunction(double left, double right, int pointCount) {
18        if (left > right || pointCount < 2) {
19            throw new IllegalArgumentException("Invalid arguments for function creation");
20        }
21        this.count = pointCount;
22        this.head = new FunctionNode();
23        this.head.prev = null;
24        this.head.next = null;
25
26        double step = (right - left) / (pointCount - 1);
27        for (int i = 0; i < pointCount; i++) {
28            addNodeAtLeft(i * step, 0);
29        }
30    }
31
32    public LinkedListFunction(double left, double right, double[] values) {
33        if (left > right || values.length < 2) {
34            throw new IllegalArgumentException("Invalid arguments for function creation");
35        }
36        this.count = values.length;
37        this.head = new FunctionNode();
38        this.head.prev = null;
39        this.head.next = null;
40
41        double step = (right - left) / (values.length - 1);
42        for (int i = 0; i < values.length; i++) {
43            addNodeAtLeft(i * step, values[i]);
44        }
45    }
46
47    // methods for getting (getLeft, getRight, getPoint)
48    private FunctionNode getHeadNode(int index) {
49        FunctionNode current = head;
50        for (int i = 0; i < index; i++) {
51            current = current.next;
52        }
53        return current;
54    }
55
56    private FunctionNode addNodeAtLeft(int index) {
57        FunctionNode newNode = new FunctionNode();
58        newNode.prev = head;
59        newNode.next = head;
60        head.prev = newNode;
61        head.next = newNode;
62        return newNode;
63    }
64
65    private FunctionNode addNodeAtRight(int index) {
66        FunctionNode newNode = new FunctionNode();
67        FunctionNode current = getHeadNode(index);
68        newNode.prev = current;
69        newNode.next = null;
70        current.next = newNode;
71    }
72
73    // methods for getting (getLeft, getRight, getPoint)
74    public double getLeft() {
75        return head.prev.point.getX();
76    }
77
78    public double getRight() {
79        return head.next.point.getX();
80    }
81
82    public double getFunctionValue(double x) {
83        if (x < getLeft() || x > getRight()) {
84            return Double.NaN;
85        }
86        FunctionNode current = head;
87        for (int i = 0; i < count; i++) {
88            if (current.point.getX() <= x && x < current.next.point.getX()) {
89                double x1 = current.point.getX();
90                double y1 = current.point.getY();
91                double x2 = current.next.point.getX();
92                double y2 = current.next.point.getY();
93                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
94            }
95            current = current.next;
96        }
97        return Double.NaN;
98    }
99
100    public int getPointsCount() {
101        return count;
102    }
103
104    private void checkIndex(int index) {
105        if (index < 0 || index > count) {
106            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
107        }
108    }
109
110    public FunctionPoint getPoint(int index) {
111        checkIndex(index);
112        return new FunctionPoint(getHeadNode(index).point);
113    }
114
115    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
116        checkIndex(index);
117        FunctionNode node = getHeadNode(index);
118        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
119        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
120        if (point.getX() < leftBound || point.getX() > rightBound) {
121            throw new InappropriateFunctionPointException("X is out of interval");
122        }
123        node.point = new FunctionPoint(point);
124    }
125
126    // methods for getting (getLeft, getRight, getPoint)
127    public double getLeft() {
128        return head.prev.point.getX();
129    }
130
131    public double getRight() {
132        return head.next.point.getX();
133    }
134
135    public double getFunctionValue(double x) {
136        if (x < getLeft() || x > getRight()) {
137            return Double.NaN;
138        }
139        FunctionNode current = head;
140        for (int i = 0; i < count; i++) {
141            if (current.point.getX() <= x && x < current.next.point.getX()) {
142                double x1 = current.point.getX();
143                double y1 = current.point.getY();
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 is out of bounds");
160        }
161    }
162
163    public FunctionPoint getPoint(int index) {
164        checkIndex(index);
165        return new FunctionPoint(getHeadNode(index).point);
166    }
167
168    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
169        checkIndex(index);
170        FunctionNode node = getHeadNode(index);
171        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
172        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
173        if (point.getX() < leftBound || point.getX() > rightBound) {
174            throw new InappropriateFunctionPointException("X is out of interval");
175        }
176        node.point = new FunctionPoint(point);
177    }
178
179    // methods for getting (getLeft, getRight, getPoint)
180    public double getLeft() {
181        return head.prev.point.getX();
182    }
183
184    public double getRight() {
185        return head.next.point.getX();
186    }
187
188    public double getFunctionValue(double x) {
189        if (x < getLeft() || x > getRight()) {
190            return Double.NaN;
191        }
192        FunctionNode current = head;
193        for (int i = 0; i < count; i++) {
194            if (current.point.getX() <= x && x < current.next.point.getX()) {
195                double x1 = current.point.getX();
196                double y1 = current.point.getY();
197                double x2 = current.next.point.getX();
198                double y2 = current.next.point.getY();
199                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
200            }
201            current = current.next;
202        }
203        return Double.NaN;
204    }
205
206    public int getPointsCount() {
207        return count;
208    }
209
210    private void checkIndex(int index) {
211        if (index < 0 || index > count) {
212            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
213        }
214    }
215
216    public FunctionPoint getPoint(int index) {
217        checkIndex(index);
218        return new FunctionPoint(getHeadNode(index).point);
219    }
220
221    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
222        checkIndex(index);
223        FunctionNode node = getHeadNode(index);
224        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
225        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
226        if (point.getX() < leftBound || point.getX() > rightBound) {
227            throw new InappropriateFunctionPointException("X is out of interval");
228        }
229        node.point = new FunctionPoint(point);
230    }
231
232    // methods for getting (getLeft, getRight, getPoint)
233    public double getLeft() {
234        return head.prev.point.getX();
235    }
236
237    public double getRight() {
238        return head.next.point.getX();
239    }
240
241    public double getFunctionValue(double x) {
242        if (x < getLeft() || x > getRight()) {
243            return Double.NaN;
244        }
245        FunctionNode current = head;
246        for (int i = 0; i < count; i++) {
247            if (current.point.getX() <= x && x < current.next.point.getX()) {
248                double x1 = current.point.getX();
249                double y1 = current.point.getY();
250                double x2 = current.next.point.getX();
251                double y2 = current.next.point.getY();
252                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
253            }
254            current = current.next;
255        }
256        return Double.NaN;
257    }
258
259    public int getPointsCount() {
260        return count;
261    }
262
263    private void checkIndex(int index) {
264        if (index < 0 || index > count) {
265            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
266        }
267    }
268
269    public FunctionPoint getPoint(int index) {
270        checkIndex(index);
271        return new FunctionPoint(getHeadNode(index).point);
272    }
273
274    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
275        checkIndex(index);
276        FunctionNode node = getHeadNode(index);
277        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
278        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
279        if (point.getX() < leftBound || point.getX() > rightBound) {
280            throw new InappropriateFunctionPointException("X is out of interval");
281        }
282        node.point = new FunctionPoint(point);
283    }
284
285    // methods for getting (getLeft, getRight, getPoint)
286    public double getLeft() {
287        return head.prev.point.getX();
288    }
289
290    public double getRight() {
291        return head.next.point.getX();
292    }
293
294    public double getFunctionValue(double x) {
295        if (x < getLeft() || x > getRight()) {
296            return Double.NaN;
297        }
298        FunctionNode current = head;
299        for (int i = 0; i < count; i++) {
300            if (current.point.getX() <= x && x < current.next.point.getX()) {
301                double x1 = current.point.getX();
302                double y1 = current.point.getY();
303                double x2 = current.next.point.getX();
304                double y2 = current.next.point.getY();
305                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
306            }
307            current = current.next;
308        }
309        return Double.NaN;
310    }
311
312    public int getPointsCount() {
313        return count;
314    }
315
316    private void checkIndex(int index) {
317        if (index < 0 || index > count) {
318            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
319        }
320    }
321
322    public FunctionPoint getPoint(int index) {
323        checkIndex(index);
324        return new FunctionPoint(getHeadNode(index).point);
325    }
326
327    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
328        checkIndex(index);
329        FunctionNode node = getHeadNode(index);
330        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
331        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
332        if (point.getX() < leftBound || point.getX() > rightBound) {
333            throw new InappropriateFunctionPointException("X is out of interval");
334        }
335        node.point = new FunctionPoint(point);
336    }
337
338    // methods for getting (getLeft, getRight, getPoint)
339    public double getLeft() {
340        return head.prev.point.getX();
341    }
342
343    public double getRight() {
344        return head.next.point.getX();
345    }
346
347    public double getFunctionValue(double x) {
348        if (x < getLeft() || x > getRight()) {
349            return Double.NaN;
350        }
351        FunctionNode current = head;
352        for (int i = 0; i < count; i++) {
353            if (current.point.getX() <= x && x < current.next.point.getX()) {
354                double x1 = current.point.getX();
355                double y1 = current.point.getY();
356                double x2 = current.next.point.getX();
357                double y2 = current.next.point.getY();
358                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
359            }
360            current = current.next;
361        }
362        return Double.NaN;
363    }
364
365    public int getPointsCount() {
366        return count;
367    }
368
369    private void checkIndex(int index) {
370        if (index < 0 || index > count) {
371            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
372        }
373    }
374
375    public FunctionPoint getPoint(int index) {
376        checkIndex(index);
377        return new FunctionPoint(getHeadNode(index).point);
378    }
379
380    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
381        checkIndex(index);
382        FunctionNode node = getHeadNode(index);
383        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
384        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
385        if (point.getX() < leftBound || point.getX() > rightBound) {
386            throw new InappropriateFunctionPointException("X is out of interval");
387        }
388        node.point = new FunctionPoint(point);
389    }
390
391    // methods for getting (getLeft, getRight, getPoint)
392    public double getLeft() {
393        return head.prev.point.getX();
394    }
395
396    public double getRight() {
397        return head.next.point.getX();
398    }
399
400    public double getFunctionValue(double x) {
401        if (x < getLeft() || x > getRight()) {
402            return Double.NaN;
403        }
404        FunctionNode current = head;
405        for (int i = 0; i < count; i++) {
406            if (current.point.getX() <= x && x < current.next.point.getX()) {
407                double x1 = current.point.getX();
408                double y1 = current.point.getY();
409                double x2 = current.next.point.getX();
410                double y2 = current.next.point.getY();
411                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
412            }
413            current = current.next;
414        }
415        return Double.NaN;
416    }
417
418    public int getPointsCount() {
419        return count;
420    }
421
422    private void checkIndex(int index) {
423        if (index < 0 || index > count) {
424            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
425        }
426    }
427
428    public FunctionPoint getPoint(int index) {
429        checkIndex(index);
430        return new FunctionPoint(getHeadNode(index).point);
431    }
432
433    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
434        checkIndex(index);
435        FunctionNode node = getHeadNode(index);
436        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
437        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
438        if (point.getX() < leftBound || point.getX() > rightBound) {
439            throw new InappropriateFunctionPointException("X is out of interval");
440        }
441        node.point = new FunctionPoint(point);
442    }
443
444    // methods for getting (getLeft, getRight, getPoint)
445    public double getLeft() {
446        return head.prev.point.getX();
447    }
448
449    public double getRight() {
450        return head.next.point.getX();
451    }
452
453    public double getFunctionValue(double x) {
454        if (x < getLeft() || x > getRight()) {
455            return Double.NaN;
456        }
457        FunctionNode current = head;
458        for (int i = 0; i < count; i++) {
459            if (current.point.getX() <= x && x < current.next.point.getX()) {
460                double x1 = current.point.getX();
461                double y1 = current.point.getY();
462                double x2 = current.next.point.getX();
463                double y2 = current.next.point.getY();
464                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
465            }
466            current = current.next;
467        }
468        return Double.NaN;
469    }
470
471    public int getPointsCount() {
472        return count;
473    }
474
475    private void checkIndex(int index) {
476        if (index < 0 || index > count) {
477            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
478        }
479    }
480
481    public FunctionPoint getPoint(int index) {
482        checkIndex(index);
483        return new FunctionPoint(getHeadNode(index).point);
484    }
485
486    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
487        checkIndex(index);
488        FunctionNode node = getHeadNode(index);
489        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
490        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
491        if (point.getX() < leftBound || point.getX() > rightBound) {
492            throw new InappropriateFunctionPointException("X is out of interval");
493        }
494        node.point = new FunctionPoint(point);
495    }
496
497    // methods for getting (getLeft, getRight, getPoint)
498    public double getLeft() {
499        return head.prev.point.getX();
500    }
501
502    public double getRight() {
503        return head.next.point.getX();
504    }
505
506    public double getFunctionValue(double x) {
507        if (x < getLeft() || x > getRight()) {
508            return Double.NaN;
509        }
510        FunctionNode current = head;
511        for (int i = 0; i < count; i++) {
512            if (current.point.getX() <= x && x < current.next.point.getX()) {
513                double x1 = current.point.getX();
514                double y1 = current.point.getY();
515                double x2 = current.next.point.getX();
516                double y2 = current.next.point.getY();
517                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
518            }
519            current = current.next;
520        }
521        return Double.NaN;
522    }
523
524    public int getPointsCount() {
525        return count;
526    }
527
528    private void checkIndex(int index) {
529        if (index < 0 || index > count) {
530            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
531        }
532    }
533
534    public FunctionPoint getPoint(int index) {
535        checkIndex(index);
536        return new FunctionPoint(getHeadNode(index).point);
537    }
538
539    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
540        checkIndex(index);
541        FunctionNode node = getHeadNode(index);
542        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
543        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
544        if (point.getX() < leftBound || point.getX() > rightBound) {
545            throw new InappropriateFunctionPointException("X is out of interval");
546        }
547        node.point = new FunctionPoint(point);
548    }
549
550    // methods for getting (getLeft, getRight, getPoint)
551    public double getLeft() {
552        return head.prev.point.getX();
553    }
554
555    public double getRight() {
556        return head.next.point.getX();
557    }
558
559    public double getFunctionValue(double x) {
560        if (x < getLeft() || x > getRight()) {
561            return Double.NaN;
562        }
563        FunctionNode current = head;
564        for (int i = 0; i < count; i++) {
565            if (current.point.getX() <= x && x < current.next.point.getX()) {
566                double x1 = current.point.getX();
567                double y1 = current.point.getY();
568                double x2 = current.next.point.getX();
569                double y2 = current.next.point.getY();
570                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
571            }
572            current = current.next;
573        }
574        return Double.NaN;
575    }
576
577    public int getPointsCount() {
578        return count;
579    }
580
581    private void checkIndex(int index) {
582        if (index < 0 || index > count) {
583            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
584        }
585    }
586
587    public FunctionPoint getPoint(int index) {
588        checkIndex(index);
589        return new FunctionPoint(getHeadNode(index).point);
590    }
591
592    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
593        checkIndex(index);
594        FunctionNode node = getHeadNode(index);
595        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
596        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
597        if (point.getX() < leftBound || point.getX() > rightBound) {
598            throw new InappropriateFunctionPointException("X is out of interval");
599        }
600        node.point = new FunctionPoint(point);
601    }
602
603    // methods for getting (getLeft, getRight, getPoint)
604    public double getLeft() {
605        return head.prev.point.getX();
606    }
607
608    public double getRight() {
609        return head.next.point.getX();
610    }
611
612    public double getFunctionValue(double x) {
613        if (x < getLeft() || x > getRight()) {
614            return Double.NaN;
615        }
616        FunctionNode current = head;
617        for (int i = 0; i < count; i++) {
618            if (current.point.getX() <= x && x < current.next.point.getX()) {
619                double x1 = current.point.getX();
620                double y1 = current.point.getY();
621                double x2 = current.next.point.getX();
622                double y2 = current.next.point.getY();
623                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
624            }
625            current = current.next;
626        }
627        return Double.NaN;
628    }
629
630    public int getPointsCount() {
631        return count;
632    }
633
634    private void checkIndex(int index) {
635        if (index < 0 || index > count) {
636            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
637        }
638    }
639
640    public FunctionPoint getPoint(int index) {
641        checkIndex(index);
642        return new FunctionPoint(getHeadNode(index).point);
643    }
644
645    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
646        checkIndex(index);
647        FunctionNode node = getHeadNode(index);
648        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
649        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
650        if (point.getX() < leftBound || point.getX() > rightBound) {
651            throw new InappropriateFunctionPointException("X is out of interval");
652        }
653        node.point = new FunctionPoint(point);
654    }
655
656    // methods for getting (getLeft, getRight, getPoint)
657    public double getLeft() {
658        return head.prev.point.getX();
659    }
660
661    public double getRight() {
662        return head.next.point.getX();
663    }
664
665    public double getFunctionValue(double x) {
666        if (x < getLeft() || x > getRight()) {
667            return Double.NaN;
668        }
669        FunctionNode current = head;
670        for (int i = 0; i < count; i++) {
671            if (current.point.getX() <= x && x < current.next.point.getX()) {
672                double x1 = current.point.getX();
673                double y1 = current.point.getY();
674                double x2 = current.next.point.getX();
675                double y2 = current.next.point.getY();
676                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
677            }
678            current = current.next;
679        }
680        return Double.NaN;
681    }
682
683    public int getPointsCount() {
684        return count;
685    }
686
687    private void checkIndex(int index) {
688        if (index < 0 || index > count) {
689            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
690        }
691    }
692
693    public FunctionPoint getPoint(int index) {
694        checkIndex(index);
695        return new FunctionPoint(getHeadNode(index).point);
696    }
697
698    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
699        checkIndex(index);
700        FunctionNode node = getHeadNode(index);
701        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
702        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
703        if (point.getX() < leftBound || point.getX() > rightBound) {
704            throw new InappropriateFunctionPointException("X is out of interval");
705        }
706        node.point = new FunctionPoint(point);
707    }
708
709    // methods for getting (getLeft, getRight, getPoint)
710    public double getLeft() {
711        return head.prev.point.getX();
712    }
713
714    public double getRight() {
715        return head.next.point.getX();
716    }
717
718    public double getFunctionValue(double x) {
719        if (x < getLeft() || x > getRight()) {
720            return Double.NaN;
721        }
722        FunctionNode current = head;
723        for (int i = 0; i < count; i++) {
724            if (current.point.getX() <= x && x < current.next.point.getX()) {
725                double x1 = current.point.getX();
726                double y1 = current.point.getY();
727                double x2 = current.next.point.getX();
728                double y2 = current.next.point.getY();
729                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
730            }
731            current = current.next;
732        }
733        return Double.NaN;
734    }
735
736    public int getPointsCount() {
737        return count;
738    }
739
740    private void checkIndex(int index) {
741        if (index < 0 || index > count) {
742            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
743        }
744    }
745
746    public FunctionPoint getPoint(int index) {
747        checkIndex(index);
748        return new FunctionPoint(getHeadNode(index).point);
749    }
750
751    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
752        checkIndex(index);
753        FunctionNode node = getHeadNode(index);
754        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
755        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
756        if (point.getX() < leftBound || point.getX() > rightBound) {
757            throw new InappropriateFunctionPointException("X is out of interval");
758        }
759        node.point = new FunctionPoint(point);
760    }
761
762    // methods for getting (getLeft, getRight, getPoint)
763    public double getLeft() {
764        return head.prev.point.getX();
765    }
766
767    public double getRight() {
768        return head.next.point.getX();
769    }
770
771    public double getFunctionValue(double x) {
772        if (x < getLeft() || x > getRight()) {
773            return Double.NaN;
774        }
775        FunctionNode current = head;
776        for (int i = 0; i < count; i++) {
777            if (current.point.getX() <= x && x < current.next.point.getX()) {
778                double x1 = current.point.getX();
779                double y1 = current.point.getY();
780                double x2 = current.next.point.getX();
781                double y2 = current.next.point.getY();
782                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
783            }
784            current = current.next;
785        }
786        return Double.NaN;
787    }
788
789    public int getPointsCount() {
790        return count;
791    }
792
793    private void checkIndex(int index) {
794        if (index < 0 || index > count) {
795            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
796        }
797    }
798
799    public FunctionPoint getPoint(int index) {
800        checkIndex(index);
801        return new FunctionPoint(getHeadNode(index).point);
802    }
803
804    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
805        checkIndex(index);
806        FunctionNode node = getHeadNode(index);
807        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
808        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
809        if (point.getX() < leftBound || point.getX() > rightBound) {
810            throw new InappropriateFunctionPointException("X is out of interval");
811        }
812        node.point = new FunctionPoint(point);
813    }
814
815    // methods for getting (getLeft, getRight, getPoint)
816    public double getLeft() {
817        return head.prev.point.getX();
818    }
819
820    public double getRight() {
821        return head.next.point.getX();
822    }
823
824    public double getFunctionValue(double x) {
825        if (x < getLeft() || x > getRight()) {
826            return Double.NaN;
827        }
828        FunctionNode current = head;
829        for (int i = 0; i < count; i++) {
830            if (current.point.getX() <= x && x < current.next.point.getX()) {
831                double x1 = current.point.getX();
832                double y1 = current.point.getY();
833                double x2 = current.next.point.getX();
834                double y2 = current.next.point.getY();
835                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
836            }
837            current = current.next;
838        }
839        return Double.NaN;
840    }
841
842    public int getPointsCount() {
843        return count;
844    }
845
846    private void checkIndex(int index) {
847        if (index < 0 || index > count) {
848            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
849        }
850    }
851
852    public FunctionPoint getPoint(int index) {
853        checkIndex(index);
854        return new FunctionPoint(getHeadNode(index).point);
855    }
856
857    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
858        checkIndex(index);
859        FunctionNode node = getHeadNode(index);
860        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
861        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
862        if (point.getX() < leftBound || point.getX() > rightBound) {
863            throw new InappropriateFunctionPointException("X is out of interval");
864        }
865        node.point = new FunctionPoint(point);
866    }
867
868    // methods for getting (getLeft, getRight, getPoint)
869    public double getLeft() {
870        return head.prev.point.getX();
871    }
872
873    public double getRight() {
874        return head.next.point.getX();
875    }
876
877    public double getFunctionValue(double x) {
878        if (x < getLeft() || x > getRight()) {
879            return Double.NaN;
880        }
881        FunctionNode current = head;
882        for (int i = 0; i < count; i++) {
883            if (current.point.getX() <= x && x < current.next.point.getX()) {
884                double x1 = current.point.getX();
885                double y1 = current.point.getY();
886                double x2 = current.next.point.getX();
887                double y2 = current.next.point.getY();
888                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
889            }
890            current = current.next;
891        }
892        return Double.NaN;
893    }
894
895    public int getPointsCount() {
896        return count;
897    }
898
899    private void checkIndex(int index) {
900        if (index < 0 || index > count) {
901            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
902        }
903    }
904
905    public FunctionPoint getPoint(int index) {
906        checkIndex(index);
907        return new FunctionPoint(getHeadNode(index).point);
908    }
909
910    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
911        checkIndex(index);
912        FunctionNode node = getHeadNode(index);
913        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
914        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
915        if (point.getX() < leftBound || point.getX() > rightBound) {
916            throw new InappropriateFunctionPointException("X is out of interval");
917        }
918        node.point = new FunctionPoint(point);
919    }
920
921    // methods for getting (getLeft, getRight, getPoint)
922    public double getLeft() {
923        return head.prev.point.getX();
924    }
925
926    public double getRight() {
927        return head.next.point.getX();
928    }
929
930    public double getFunctionValue(double x) {
931        if (x < getLeft() || x > getRight()) {
932            return Double.NaN;
933        }
934        FunctionNode current = head;
935        for (int i = 0; i < count; i++) {
936            if (current.point.getX() <= x && x < current.next.point.getX()) {
937                double x1 = current.point.getX();
938                double y1 = current.point.getY();
939                double x2 = current.next.point.getX();
940                double y2 = current.next.point.getY();
941                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
942            }
943            current = current.next;
944        }
945        return Double.NaN;
946    }
947
948    public int getPointsCount() {
949        return count;
950    }
951
952    private void checkIndex(int index) {
953        if (index < 0 || index > count) {
954            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
955        }
956    }
957
958    public FunctionPoint getPoint(int index) {
959        checkIndex(index);
960        return new FunctionPoint(getHeadNode(index).point);
961    }
962
963    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
964        checkIndex(index);
965        FunctionNode node = getHeadNode(index);
966        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
967        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
968        if (point.getX() < leftBound || point.getX() > rightBound) {
969            throw new InappropriateFunctionPointException("X is out of interval");
970        }
971        node.point = new FunctionPoint(point);
972    }
973
974    // methods for getting (getLeft, getRight, getPoint)
975    public double getLeft() {
976        return head.prev.point.getX();
977    }
978
979    public double getRight() {
980        return head.next.point.getX();
981    }
982
983    public double getFunctionValue(double x) {
984        if (x < getLeft() || x > getRight()) {
985            return Double.NaN;
986        }
987        FunctionNode current = head;
988        for (int i = 0; i < count; i++) {
989            if (current.point.getX() <= x && x < current.next.point.getX()) {
990                double x1 = current.point.getX();
991                double y1 = current.point.getY();
992                double x2 = current.next.point.getX();
993                double y2 = current.next.point.getY();
994                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
995            }
996            current = current.next;
997        }
998        return Double.NaN;
999    }
1000
1001    public int getPointsCount() {
1002        return count;
1003    }
1004
1005    private void checkIndex(int index) {
1006        if (index < 0 || index > count) {
1007            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
1008        }
1009    }
1010
1011    public FunctionPoint getPoint(int index) {
1012        checkIndex(index);
1013        return new FunctionPoint(getHeadNode(index).point);
1014    }
1015
1016    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
1017       
```

```

125
126
127 public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
128     checkIndex(index);
129     double newX = point.getX();
130     FunctionNode node = getNodeByIndex(index);
131     double leftBound = node.prev == head ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
132     double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
133     if (newX < leftBound || newX > rightBound) {
134         throw new InappropriateFunctionPointException("x is out of interval");
135     }
136     node.point = new FunctionPoint(point);
137 }
138
139 public double getPointX(int index) {
140     checkIndex(index);
141     return getNodeByIndex(index).point.getX();
142 }
143
144 public void setPointX(int index, double x) throws InappropriateFunctionPointException {
145     checkIndex(index);
146     FunctionNode node = getNodeByIndex(index);
147     double leftBound = node.prev == head ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
148     double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
149     if (x < leftBound || x > rightBound) {
150         throw new InappropriateFunctionPointException("x is out of interval");
151     }
152     node.point.setX(x);
153 }
154
155 public double getPointY(int index) {
156     checkIndex(index);
157     return getNodeByIndex(index).point.getY();
158 }
159
160 public void setPointY(int index, double y) {
161     checkIndex(index);
162     getNodeByIndex(index).point.setY(y);
163 }
164
165 public void deletePoint(int index) {
166     checkIndex(index);
167     if (count < 3) {
168         throw new IllegalStateException("Cannot delete point, less than 3 points in function");
169     }
170     deleteNodeByIndex(index);
171     count--;
172 }
173
174 public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
175     FunctionNode current = head.next;
176     int index = 0;
177     while (current != head && current.point.getX() < point.getX()) {
178         current = current.next;
179         index++;
180     }
181     if (current != head && current.point.getX() == point.getX()) {
182         throw new InappropriateFunctionPointException("Point with such x already exists");
183     }
184     addNodeByIndex(index, point = new FunctionPoint(point));
185     count++;
186 }
187
188 }
189
190

```

ЗАДАНИЕ 6

Класс `TabulatedFunction` я переименовал в класс `ArrayTabulatedFunction`. Создал интерфейс `TabulatedFunction`, содержащий объявления общих методов классов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`. Сделал так, чтобы оба класса функций реализовывали созданный интерфейс.

Теперь суть работы с табулированными функциями заключена в типе интерфейса, а в классах заключена только реализация этой работы.

```

LinkedListTabulatedFunction.java U   ArrayTabulatedFunction.java 1, U   TabulatedFunction.java 1, M X
Lab-2-2025 > src > functions > TabulatedFunction.java > ...
1  package functions;
2
3  public interface TabulatedFunction {
4      double getLeftDomainBorder();
5      double getRightDomainBorder();
6      double getFunctionValue(double x);
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

```

ЗАДАНИЕ 7

Я проверить работу написанных классов.

Для этого в созданном ранее классе Main, содержащем точку входа программы, я добавлю проверку для случаев, в которых объект табулированной функции должен выбрасывать исключения.

Ссылочную переменную для работы с объектом функции я объявил типа TabulatedFunction, а при создании объекта указал реальный класс. Теперь проверка работы класса LinkedListTabulatedFunction может быть произведена путем простой замены класса, объект которого создается.

```
src > Main.java > Main > testFunction(TabulatedFunction)
1 import functions.*;
2
3 public class Main {
4     Run | Debug
5     public static void main(String[] args) {
6         double[] values = { 0, 1, 4, 9, 16 };
7
8         // ТЕСТИРОВАНИЕ ArrayTabulatedFunction
9         System.out.println("ТЕСТИРОВАНИЕ ArrayTabulatedFunction");
10        TabulatedFunction arrayFunction = new ArrayTabulatedFunction(leftX:0, rightX:4, values);
11        testFunction(arrayFunction); // Общая проверка методов
12        testExceptions(arrayFunction); // Проверка исключений
13
14        // ТЕСТИРОВАНИЕ LinkedListTabulatedFunction
15        System.out.println("ТЕСТИРОВАНИЕ LinkedListTabulatedFunction");
16        TabulatedFunction listFunction = new LinkedListTabulatedFunction(leftX:0, rightX:4, values);
17        testFunction(listFunction); // Общая проверка методов
18        testExceptions(listFunction); // Проверка исключений
19    }
20
21    // Метод тестирует основные операции с функцией: получение значения, изменение, добавление и удаление точек.
22    public static void testFunction(TabulatedFunction func) {
23        System.out.println("Исходная функция (" + func.getClass().getSimpleName() + ")");
24        printFunction(func);
25
26        System.out.println("Проверка getFunctionValue");
27        System.out.println("Значение в точке f(2,5) = " + func.getFunctionValue(x:2.5));
28
29        System.out.println("Проверка setPointY");
30        System.out.println("Изменяем Y в точке [ ] индексом 2 на 5,0");
31        func.setPointY(index:2, y:5.0);
32        printFunction(func);
33        // Чтобы не запутаться, возвращаем обратно
34        func.setPointY(index:2, y:4.0);
35
36        System.out.println("Проверка deletePoint");
37        System.out.println("Удаляем точку [ ] индексом 1");
38        func.deletePoint(index:1);
39        printFunction(func);
40
41        System.out.println("Проверка addPoint");
42        System.out.println("Добавляем точку (1.0; 1.0) обратно");
43        try {
44            func.addPoint(new FunctionPoint(x:1.0, y:1.0));
45        } catch (InappropriateFunctionPointException e) {
46            System.out.println("удалось добавить точку: " + e.getMessage());
47        }
48        printFunction(func);
49
50        // Этот метод тестирует выбрасывание всех необходимых исключений.
51        public static void testExceptions(TabulatedFunction func) {
52            System.out.println("Тестирование исключений для " + func.getClass().getSimpleName());
53
54            // 1. Выход за границы индекса
55            try {
56                System.out.println("Попытка получить точку [ ] индексом 10 !!!!!");
57                func.getPoint(index:10);
58            }
```

```

1  Main.java X
2  public class Main {
3      public static void testExceptions(TabulatedFunction func) {
4          // 1. Выход за границы индекса
5          try {
6              System.out.println("Попытка получить точку с индексом 10 !!!!!");
7              func.getPoint(10);
8          } catch (FunctionPointIndexOutOfBoundsException e) {
9              System.out.println("Перехвачено: " + e.getClass().getSimpleName());
10         }
11
12         // 2. Некорректный X при изменении
13         try {
14             System.out.println("Попытка изменить X точки 2 на 0,5 (неверно) !!!!!");
15             func.setPointX(2, 0.5);
16         } catch (InappropriateFunctionPointException e) {
17             System.out.println("Перехвачено: " + e.getClass().getSimpleName());
18         }
19
20         // 3. Добавление точки с существующим X
21         try {
22             System.out.println("Попытка добавить точку (3.0; 9.0) !!!!!");
23             func.addPoint(new FunctionPoint(3.0, 9.0));
24         } catch (InappropriateFunctionPointException e) {
25             System.out.println("Перехвачено: " + e.getClass().getSimpleName());
26         }
27
28         // 4. Удаление из маленькой функции
29         try {
30             System.out.println("Попытка удалить точку из функции с 2 точками !!!!!");
31             // Создаем временный объект того же класса, что и тестируемый
32             TabulatedFunction smallFunc;
33             if (func instanceof ArrayTabulatedFunction) {
34                 smallFunc = new ArrayTabulatedFunction(1, 1, new double[] {0, 1});
35             } else {
36                 smallFunc = new LinkedListTabulatedFunction(1, 1, new double[] {0, 1});
37             }
38             smallFunc.deletePoint(0);
39         } catch (IllegalStateException e) {
40             System.out.println("Перехвачено: " + e.getClass().getSimpleName());
41         }
42     }
43
44     // Дополнительный метод для красивой печати
45     public static void printFunction(TabulatedFunction func) {
46         System.out.println("Функция из " + func.getPointsCount() + " точек:");
47         for (int i = 0; i < func.getPointsCount(); i++) {
48             FunctionPoint p = func.getPoint(i);
49             System.out.println("Точка " + i + ": (" + p.getX() + "; " + p.getY() + ")");
50         }
51     }
52 }
53

```

```

PS C:\projects\Lab-3-2025> c:; cd 'c:\projects\Lab-3-2025'; & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:58188' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\29d5783e413a3470974e7b186a7b3547\redhat.java\jdt_ws\Lab-3-2025_426b55b8\bin' 'Main'
ТЕСТИРОВАНИЕ ArrayTabulatedFunction
Исходная функция (ArrayTabulatedFunction)
Функция из 5 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 4.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)

Проверка getFunctionValue
Значение в точке f(2,5) = 6.5

Проверка setPointY
Изменяем Y в точке с индексом 2 на 5,0
Функция из 5 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 5.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)

Проверка deletePoint
Удаляем точку с индексом 1
Функция из 4 точек:
Точка 0: (0.0; 0.0)
Точка 1: (2.0; 4.0)
Точка 2: (3.0; 9.0)
Точка 3: (4.0; 16.0)

Проверка addPoint
Добавляем точку (1.0; 1.0) обратно
Функция из 5 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 4.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)

```

```
,address=localhost:58188' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\29d5783e413a3470974e7b186a7b3547\redhat.java\jdt_ws\Lab-3-2025_426b55b8\bin' 'Main'
ТЕСТИРОВАНИЕ ArrayTabulatedFunction
Точка 4: (4.0; 16.0)

    Тестирование исключений для ArrayTabulatedFunction
Попытка получить точку с индексом 10 !!!!!
Перехвачено: FunctionPointIndexOutOfBoundsException
Попытка изменить X точки 2 на 0,5 (неверно) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка добавить точку (3.0; 9.0) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка удалить точку из функции с 2 точками !!!!!
Перехвачено: IllegalStateException

    ТЕСТИРОВАНИЕ LinkedListTabulatedFunction
Исходная функция (LinkedListTabulatedFunction)
Функция из 5 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 4.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)

    Проверка getFunctionValue
Значение в точке f(2,5) = 6.5

    Проверка setPointY
Изменяем Y в точке с индексом 2 на 5,0
Функция из 5 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 5.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)

    Проверка deletePoint
Удаляем точку с индексом 1
Функция из 4 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 5.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)

    Проверка addPoint
Добавляем точку (1.0; 1.0) обратно
Функция из 5 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 4.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)

    Тестирование исключений для LinkedListTabulatedFunction
Попытка получить точку с индексом 10 !!!!!
Перехвачено: FunctionPointIndexOutOfBoundsException
Попытка изменить X точки 2 на 0,5 (неверно) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка добавить точку (3.0; 9.0) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка удалить точку из функции с 2 точками !!!!!
Перехвачено: IllegalStateException
PS C:\projects\Lab-3-2025> c;; cd 'c:\projects\Lab-3-2025'; & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y',address=localhost:58188' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\29d5783e413a3470974e7b186a7b3547\redhat.java\jdt_ws\Lab-3-2025_426b55b8\bin' 'Main'
```