

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

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 объектно-ориентированном 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.java
2 package function;
3
4 public class TabulatedFunction {
5     private FunctionPoint[] points;
6
7     // Constructors & Methods
8     public TabulatedFunction(double leftx, double rightx, int pointCount) {
9         // Initialize members attributes
10         if (leftx >= rightx || pointCount < 2) {
11             throw new IllegalArgumentException("Invalid arguments for function creation");
12         }
13         this.points = new FunctionPoint[pointCount];
14         double step = (rightx - leftx) / (pointCount - 1);
15         for (int i = 0; i < pointCount; i++) {
16             points[i] = new FunctionPoint(leftx + i * step, 0);
17         }
18     }
19
20     public TabulatedFunction(double leftx, double rightx, double[] values) {
21         int count = values.length;
22         // Initialize members attributes
23         if (leftx >= rightx || count < 2) {
24             throw new IllegalArgumentException("Invalid arguments for function creation");
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     // Getter methods
34     public double getLeftBoundOrder() {
35         return points[0].getX();
36     }
37
38     public double getRightBoundOrder() {
39         return points[points.length - 1].getX();
40     }
41
42     public double getFunctionValue(double x) {
43         if (x < getLeftBoundOrder() || x > getRightBoundOrder()) {
44             return Double.NaN;
45         }
46         for (int i = 0; i < points.length - 1; i++) {
47             if (points[i].getX() >= x && x <= points[i + 1].getX()) {
48                 double y1 = points[i].getY();
49                 double y2 = points[i + 1].getY();
50                 double x2 = points[i + 1].getX();
51                 double y3 = points[i + 1].getY();
52                 return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
53             }
54         }
55         return Double.NaN;
56     }
57
58     // Method for finding c through c derivative
59     public int getPointCount() {
60         return points.length;
61     }
62
63     private void checkIndex(int index) {
64         if (index < 0 || index >= points.length) {
65             throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
66         }
67     }
68
69     public FunctionPoint getPoint(int index) {
70         checkIndex(index);
71         return new FunctionPoint(points[index]);
72     }
73
74     public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
75         checkIndex(index);
76         double mod = point.getX();
77         double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
78         double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
79         if (mod < leftBound || mod > rightBound) {
80             throw new InappropriateFunctionPointException("x coordinate is out of the allowed interval");
81         }
82         points[index] = new FunctionPoint(point);
83     }
84 }
```


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 // 2020-10-10 by Andreas J. Gschwendtner andreas.gschwendtner@gmail.com | https://github.com/andreasg
2 // project: https://github.com/andreasg/functional-iterators
3
4 public class ListOfIntsIterableFunction {
5     // constructor taking zero or more Integers
6     // the private, but the public iterator can generate more or zero Integer elements.
7     private final List<Integer> values;
8
9     // private constructor
10    public ListOfIntsIterableFunction(
11        Function<Integer, Boolean>
12        Predicate<Integer> pred,
13        Function<Integer, Integer>
14        Function<Integer, Integer> map,
15        Function<Integer, Integer>
16        Function<Integer, Integer> next) {
17    }
18
19    private final Function<Integer, Boolean> isFinite; // always correct, no repeat needed
20    private int count; // known number of items, infinite on certain values but not
21
22    // constructor
23    public ListOfIntsIterableFunction(
24        Predicate<Integer> pred,
25        double leftX, rightX, int
26        leftStep, rightStep, int
27        leftStepCount, rightStepCount) {
28        if (leftX > rightX) {
29            throw new IllegalArgumentException("Invalid arguments for Function creation");
30        }
31        this.count = rightStepCount;
32        this.left = leftX;
33        this.right = rightX;
34        this.leftStep = leftStep;
35        this.rightStep = rightStep;
36        this.leftStepCount = leftStepCount;
37        this.rightStepCount = rightStepCount;
38        this.isFinite = true;
39    }
40
41    public double step = (rightX - leftX) / (rightStepCount - 1);
42    for (int i = 0; i < leftStepCount; i++) {
43        addValueToTail().point = new Function<Integer, Integer>() {
44            @Override public Integer apply(Integer i) {
45                return leftX + i * step;
46            }
47        };
48    }
49
50    public ListOfIntsIterableFunction(
51        Predicate<Integer> pred,
52        double leftX, double rightX, double[] values) {
53        if (leftX > rightX) {
54            throw new IllegalArgumentException("Invalid arguments for Function creation");
55        }
56        this.count = values.length;
57        this.left = leftX;
58        this.right = rightX;
59        this.leftStep = leftX;
60        this.rightStep = rightX;
61        this.leftStepCount = 1;
62        this.rightStepCount = 1;
63        this.isFinite = true;
64    }
65
66    public double step = (rightX - leftX) / (values.length - 1);
67    for (int i = 0; i < values.length; i++) {
68        addValueToTail().point = new Function<Integer, Integer>() {
69            @Override public Integer apply(Integer i) {
70                return leftX + i * step + values[i];
71            }
72        };
73    }
74
75    // Method for adding or updating (overwriting) values
76
77    private Function<Integer, Integer> addValueToTail() {
78        Function<Integer, Integer> current = head.next;
79        // add
80        // add
81        // add
82        // add
83        // add
84        // add
85        // add
86        // add
87        // add
88        // add
89        // add
90        // add
91        // add
92        // add
93        // add
94        // add
95        // add
96        // add
97        // add
98        // add
99        // add
100        // add
101        // add
102        // add
103        // add
104        // add
105        // add
106        // add
107        // add
108        // add
109        // add
110        // add
111        // add
112        // add
113        // add
114        // add
115        // add
116        // add
117        // add
118        // add
119        // add
120        // add
121        // add
122        // add
123        // add
124        // add
125        // add
126        // add
127        // add
128        // add
129        // add
130        // add
131        // add
132        // add
133        // add
134        // add
135        // add
136        // add
137        // add
138        // add
139        // add
140        // add
141        // add
142        // add
143        // add
144        // add
145        // add
146        // add
147        // add
148        // add
149        // add
150        // add
151        // add
152        // add
153        // add
154        // add
155        // add
156        // add
157        // add
158        // add
159        // add
160        // add
161        // add
162        // add
163        // add
164        // add
165        // add
166        // add
167        // add
168        // add
169        // add
170        // add
171        // add
172        // add
173        // add
174        // add
175        // add
176        // add
177        // add
178        // add
179        // add
180        // add
181        // add
182        // add
183        // add
184        // add
185        // add
186        // add
187        // add
188        // add
189        // add
190        // add
191        // add
192        // add
193        // add
194        // add
195        // add
196        // add
197        // add
198        // add
199        // add
200        // add
201        // add
202        // add
203        // add
204        // add
205        // add
206        // add
207        // add
208        // add
209        // add
210        // add
211        // add
212        // add
213        // add
214        // add
215        // add
216        // add
217        // add
218        // add
219        // add
220        // add
221        // add
222        // add
223        // add
224        // add
225        // add
226        // add
227        // add
228        // add
229        // add
230        // add
231        // add
232        // add
233        // add
234        // add
235        // add
236        // add
237        // add
238        // add
239        // add
240        // add
241        // add
242        // add
243        // add
244        // add
245        // add
246        // add
247        // add
248        // add
249        // add
250        // add
251        // add
252        // add
253        // add
254        // add
255        // add
256        // add
257        // add
258        // add
259        // add
260        // add
261        // add
262        // add
263        // add
264        // add
265        // add
266        // add
267        // add
268        // add
269        // add
270        // add
271        // add
272        // add
273        // add
274        // add
275        // add
276        // add
277        // add
278        // add
279        // add
280        // add
281        // add
282        // add
283        // add
284        // add
285        // add
286        // add
287        // add
288        // add
289        // add
290        // add
291        // add
292        // add
293        // add
294        // add
295        // add
296        // add
297        // add
298        // add
299        // add
300        // add
301        // add
302        // add
303        // add
304        // add
305        // add
306        // add
307        // add
308        // add
309        // add
310        // add
311        // add
312        // add
313        // add
314        // add
315        // add
316        // add
317        // add
318        // add
319        // add
320        // add
321        // add
322        // add
323        // add
324        // add
325        // add
326        // add
327        // add
328        // add
329        // add
330        // add
331        // add
332        // add
333        // add
334        // add
335        // add
336        // add
337        // add
338        // add
339        // add
340        // add
341        // add
342        // add
343        // add
344        // add
345        // add
346        // add
347        // add
348        // add
349        // add
350        // add
351        // add
352        // add
353        // add
354        // add
355        // add
356        // add
357        // add
358        // add
359        // add
360        // add
361        // add
362        // add
363        // add
364        // add
365        // add
366        // add
367        // add
368        // add
369        // add
370        // add
371        // add
372        // add
373        // add
374        // add
375        // add
376        // add
377        // add
378        // add
379        // add
380        // add
381        // add
382        // add
383        // add
384        // add
385        // add
386        // add
387        // add
388        // add
389        // add
390        // add
391        // add
392        // add
393        // add
394        // add
395        // add
396        // add
397        // add
398        // add
399        // add
400        // add
401        // add
402        // add
403        // add
404        // add
405        // add
406        // add
407        // add
408        // add
409        // add
410        // add
411        // add
412        // add
413        // add
414        // add
415        // add
416        // add
417        // add
418        // add
419        // add
420        // add
421        // add
422        // add
423        // add
424        // add
425        // add
426        // add
427        // add
428        // add
429        // add
430        // add
431        // add
432        // add
433        // add
434        // add
435        // add
436        // add
437        // add
438        // add
439        // add
440        // add
441        // add
442        // add
443        // add
444        // add
445        // add
446        // add
447        // add
448        // add
449        // add
450        // add
451        // add
452        // add
453        // add
454        // add
455        // add
456        // add
457        // add
458        // add
459        // add
460        // add
461        // add
462        // add
463        // add
464        // add
465        // add
466        // add
467        // add
468        // add
469        // add
470        // add
471        // add
472        // add
473        // add
474        // add
475        // add
476        // add
477        // add
478        // add
479        // add
480        // add
481        // add
482        // add
483        // add
484        // add
485        // add
486        // add
487        // add
488        // add
489        // add
490        // add
491        // add
492        // add
493        // add
494        // add
495        // add
496        // add
497        // add
498        // add
499        // add
500        // add
501        // add
502        // add
503        // add
504        // add
505        // add
506        // add
507        // add
508        // add
509        // add
510        // add
511        // add
512        // add
513        // add
514        // add
515        // add
516        // add
517        // add
518        // add
519        // add
520        // add
521        // add
522        // add
523        // add
524        // add
525        // add
526        // add
527        // add
528
```

[illegible]

```

125 }
126
127 public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
128     checkIndex(index);
129     double head = 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 (head < leftBound || head > 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 }

```

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.

```

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.

```
... Main.java X
src > / Main.java > ...
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         // Используем тип интерфейса
9         TabulatedFunction function = new ArrayTabulatedFunction(leftX:0, rightX:4, values);
10        // TabulatedFunction function = new LinkedListTabulatedFunction(0, 4, values); // - для другой реализации
11
12        System.out.println(" Исходная функция (" + function.getClass().getSimpleName() + ")");
13        printFunction(function);
14
15        System.out.println("\n    Тестирование исключений");
16
17        // 1. Выход за границы индекса
18        try {
19            System.out.println("Попытка получить точку [ ] индексом 10 !!!!!");
20            function.getPoint(index:10);
21        } catch (FunctionPointIndexOutOfBoundsException e) {
22            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
23        }
24
25        // 2. Некорректный X при изменении
26        try {
27            System.out.println("Попытка изменить X точки 2 на 0,5 (неверно !!!!!");
28            function.setPointX(index:2, x:0.5);
29        } catch (InappropriateFunctionPointException e) {
30            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
31        }
32
33        // 3. Добавление точки с существующим X
34        try {
35            System.out.println("Попытка добавить точку (3, 9) !!!!!");
36            function.addPoint(new FunctionPoint(x:3.0, y:9.0));
37        } catch (InappropriateFunctionPointException e) {
38            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
39        }
40
41        // 4. Удаление из маленькой функции
42        try {
43            System.out.println("Попытка удалить точку из функции [ ] 2 точками !!!!!");
44            TabulatedFunction smallFunc = new ArrayTabulatedFunction(leftX:0, rightX:1, new double[]{0, 1});
45            smallFunc.deletePoint(index:0);
46        } catch (IllegalStateException e) {
47            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
48        }
49
50        // 5. Неверные аргументы конструктора
51        try {
52            System.out.println("Попытка создать функцию [ ] 1 точкой !!!!!");
53            new ArrayTabulatedFunction(leftX:0, rightX:1, pointsCount:1);
54        } catch (IllegalArgumentException e) {
55            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
56        }
57
58        public static void printFunction(TabulatedFunction func) {
59            System.out.println("Функция из " + func.getPointsCount() + " точек:");
60            for (int i = 0; i < func.getPointsCount(); i++) {
61                FunctionPoint p = func.getPoint(i);
62                System.out.println("Точка " + i + ": (" + p.getX() + "; " + p.getY() + ")");
63            }
64        }
65    }
66 }
```

```
PS C:\projects\Lab-2-2025> c::; cd 'c:\projects\Lab-2-2025'; & 'C:\Program Files\Java\jdk-24\bin\java
t:61198' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\wo
_40b67d19\bin' 'Main'
    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)

    Тестирование исключений
Попытка получить точку с индексом 10 !!!!!
Перехвачено: FunctionPointIndexOutOfBoundsException
Попытка изменить X точки 2 на 0,5 (неверно) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка добавить точку (3, 9) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка удалить точку из функции с 2 точками !!!!!
Перехвачено: IllegalStateException
Попытка создать функцию с 1 точкой !!!!!
Перехвачено: IllegalArgumentException
PS C:\projects\Lab-2-2025>
```