

# **Лабораторная работа №3**

## **Доработка пакета для работы с табулированными функциями**

Выполнил: Скиба В.А.

Группа: 6204-010302

---

## **Содержание**

1. Задание 1: Изучение исключений
  2. Задание 2: Создание классов исключений
  3. Задание 3: Модификация TabulatedFunction
  4. Задание 4: Создание LinkedListTabulatedFunction
  5. Задание 5: Реализация функциональности  
LinkedListTabulatedFunction
  6. Задание 6: Рефакторинг и создание интерфейса
  7. Задание 7: Тестирование
  8. Результаты работы
-

## **Задание 1: Изучение исключений**

Ход выполнения:

Изучена иерархия исключений Java и их применение:

- `Exception` - базовый класс проверяемых исключений
- `RuntimeException` - непроверяемые исключения
- `IndexOutOfBoundsException` - выход за границы индекса
- `IllegalArgumentException` - неверный аргумент метода
- `IllegalStateException` - неверное состояние объекта

Результат:

Определены подходящие типы исключений для различных ошибок в работе с табулированными функциями.

## **Задание 2: Создание классов исключений**

Ход выполнения:

Созданы два класса исключений в пакете `functions`:

1. `FunctionPointOutOfBoundsException` - наследует от `IndexOutOfBoundsException`
2. `InappropriateFunctionPointException` - наследует от `RuntimeException`

Реализация:

```
⑤ FunctionPointIndexOutOfBoundsException.java ×
1 package functions;
2
3 public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException { 11 usages & VladimirSkiba
4 >     public FunctionPointIndexOutOfBoundsException() { super(); }
5
6 >     public FunctionPointIndexOutOfBoundsException(String message) { super(message); }
7
8 >     public FunctionPointIndexOutOfBoundsException(int index) { super("Index out of bounds: " + index); }
9
10 >    public FunctionPointIndexOutOfBoundsException(int index, int size) { 9 usages & VladimirSkiba
11 |        super("Index " + index + " out of bounds for size " + size);
12 |
13 }
14
15
16
17
18
19 }
```

  

```
1 package functions;
2
3 public class InappropriateFunctionPointException extends RuntimeException { 12 usages & VladimirSkiba *
4 >     public InappropriateFunctionPointException() { super(); }
5
6 >     public InappropriateFunctionPointException(String message) { super(message); }
7
8 >     public InappropriateFunctionPointException(double x, double existingX) { 8 usages & VladimirSkiba
9 |         super("Point with x=" + x + " cannot be added. Point with x=" + existingX + " already exists");
10 |
11 }
12
13
14
15
16 >    public InappropriateFunctionPointException(double x, double leftX, double rightX) { 8 usages & VladimirSkiba
17 |        super("Point with x=" + x + " would break ordering. Must be between " + leftX + " and " + rightX);
18 |
19 }
```

Результат:

Созданы специализированные исключения для обработки ошибок работы с точками функций.

### Задание 3: Модификация TabulatedFunction

Ход выполнения:

Добавлена обработка исключений в класс TabulatedFunction:

- Конструкторы - проверка параметров с выбрасыванием `IllegalArgumentException`
- Методы доступа - проверка индексов с выбрасыванием `FunctionPointIndexOutOfBoundsException`
- Методы изменения - проверка порядка точек с выбрасыванием `InappropriateFunctionPointException`
- `deletePoint()` - проверка минимального количества точек с выбрасыванием `IllegalStateException`

Пример реализации:

```

public void setPoint(int index, FunctionPoint point) {  no usages  new *
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона [0, " + (pointsCount - 1) + "]");
    }

    if (index > 0 && point.getX() <= points[index - 1].getX()) {
        throw new InappropriateFunctionPointException("Х координата " + point.getX() + " должна быть больше предыдущей " + points[index - 1].getX());
    }
    if (index < pointsCount - 1 && point.getX() >= points[index + 1].getX()) {
        throw new InappropriateFunctionPointException("Х координата " + point.getX() + " должна быть меньше следующей " + points[index + 1].getX());
    }

    points[index] = new FunctionPoint(point);
}

public double getPointX(int index) {  1 usage  new *
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона [0, " + (pointsCount - 1) + "]");
    }
    return points[index].getX();
}

public void setPointX(int index, double x) {  1 usage  new *
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона [0, " + (pointsCount - 1) + "]");
    }

    if (index > 0 && x <= points[index - 1].getX()) {
        throw new InappropriateFunctionPointException("Х координата " + x + " должна быть больше предыдущей " + points[index - 1].getX());
    }
    if (index < pointsCount - 1 && x >= points[index + 1].getX()) {
        throw new InappropriateFunctionPointException("Х координата " + x + " должна быть меньше следующей " + points[index + 1].getX());
    }

    points[index].setX(x);
}

public double getPointY(int index) {  no usages  new *
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона [0, " + (pointsCount - 1) + "]");
    }
    return points[index].getY();
}

public void setPointY(int index, double y) {  1 usage  new *
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона [0, " + (pointsCount - 1) + "]");
    }
    points[index].setY(y);
}

public void deletePoint(int index) {  2 usages  new *
    if (pointsCount <= 2) {
        throw new IllegalStateException("Невозможно удалить точку: должно остаться минимум 2 точки");
    }

    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона [0, " + (pointsCount - 1) + "]");
    }

    System.arraycopy(points,  srcPos: index + 1, points, index,  length: pointsCount - index - 1);
    pointsCount--;
    points[pointsCount] = null;
}

```

## Результат:

Класс TabulatedFunction теперь корректно обрабатывает все ошибочные ситуации через исключения.

## **Задание 4: Создание LinkedListTabulatedFunction**

Ход выполнения:

Реализован класс для работы с табулированными функциями на основе связного списка:

1. Внутренний класс FunctionNode - элемент двусвязного списка
2. Структура данных - циклический двусвязный список с выделенной головой
3. Базовые методы:
  - o getNodeByIndex() - доступ с оптимизацией через кэш
  - o addNodeByIndex() - вставка элемента
  - o deleteNodeByIndex() - удаление элемента

Особенности реализации:

- Инкапсуляция класса FunctionNode как private static
- Оптимизация доступа через кэширование последнего использованного элемента
- Циклическая структура для упрощения операций

Результат:

Создана базовая структура связного списка для хранения точек функции.

```
private static class FunctionNode { 29 usages new *
    FunctionPoint point; 28 usages
    FunctionNode prev; 15 usages
    FunctionNode next; 22 usages

    FunctionNode(FunctionPoint point) { 2 usages new *
        this.point = point;
    }
}

private FunctionNode head; 18 usages
private int pointsCount; 23 usages
private FunctionNode lastAccessed; 5 usages
private int lastAccessedIndex; 10 usages

public LinkedListTabulatedFunction() { 24 usages new *
    head = new FunctionNode( point: null);
    head.prev = head;
    head.next = head;
    pointsCount = 0;
    lastAccessed = head;
    lastAccessedIndex = -1;
}
```

```
private FunctionNode getNodeByIndex(int index) { 8 usages new *
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index, pointsCount);
    }

    FunctionNode current;
    int startIndex;

    if (lastAccessedIndex != -1 && Math.abs(index - lastAccessedIndex) < Math.abs(index)) {
        current = lastAccessed;
        startIndex = lastAccessedIndex;
    } else {
        current = head.next;
        startIndex = 0;
    }

    if (index >= startIndex) {
        for (int i = startIndex; i < index; i++) {
            current = current.next;
        }
    } else {
        for (int i = startIndex; i > index; i--) {
            current = current.prev;
        }
    }

    lastAccessed = current;
    lastAccessedIndex = index;

    return current;
}
```

```
private FunctionNode addNodeByIndex(int index) { 2 usages new *
    if (index < 0 || index > pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index, pointsCount);
    }

    FunctionNode newNode = new FunctionNode(new FunctionPoint(x: 0, y: 0));

    if (pointsCount == 0) {
        newNode.prev = head;
        newNode.next = head;
        head.next = newNode;
        head.prev = newNode;
    } else {
        FunctionNode targetNode;
        if (index == pointsCount) {
            targetNode = head.prev;
        } else {
            targetNode = getNodeByIndex(index);
        }

        newNode.prev = targetNode.prev;
        newNode.next = targetNode;
        targetNode.prev.next = newNode;
        targetNode.prev = newNode;
    }

    pointsCount++;
    lastAccessed = newNode;
    lastAccessedIndex = index;

    return newNode;
}
```

```
private FunctionNode deleteNodeByIndex(int index) { 1 usage new *
    if (pointsCount <= 2) {
        throw new IllegalStateException("Невозможно удалить точку: должно остаться минимум 2 точки");
    }
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(index, pointsCount);
    }

    FunctionNode nodeToDelete = getNodeByIndex(index);

    nodeToDelete.prev.next = nodeToDelete.next;
    nodeToDelete.next.prev = nodeToDelete.prev;

    pointsCount--;

    if (lastAccessedIndex == index) {
        lastAccessed = head.next;
        lastAccessedIndex = 0;
    } else if (lastAccessedIndex > index) {
        lastAccessedIndex--;
    }

    return nodeToDelete;
}
```

## Задание 5: Реализация функциональности

### LinkedListTabulatedFunction

Ход выполнения:

Реализованы конструкторы и методы, аналогичные

ArrayTabulatedFunction:

- Конструкторы - создание функции с равномерным распределением точек
- Методы интерфейса - getLeftDomainBorder(), getFunctionValue(), и др.
- Обработка исключений - аналогично ArrayTabulatedFunction
- Оптимизации - прямой доступ к узлам списка

## Пример оптимизации:

```
public double getLeftDomainBorder() { 3 usages new *
    if (pointsCount == 0) {
        throw new IllegalStateException("Функция не содержит точек");
    }
    return head.next.point.getX();
}

public double getFunctionValue(double x) { 2 usages new *
    if (pointsCount == 0) {
        return Double.NaN;
    }

    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
        return Double.NaN;
    }

    FunctionNode current = head.next;
    for (int i = 0; i < pointsCount; i++) {
        if (x == current.point.getX()) {
            return current.point.getY();
        }
        current = current.next;
    }

    current = head.next;
    for (int i = 0; i < pointsCount - 1; i++) {
        double x1 = current.point.getX();
        double x2 = current.next.point.getX();

        if (x >= x1 && x <= x2) {
            double y1 = current.point.getY();
            double y2 = current.next.point.getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
        current = current.next;
    }

    return Double.NaN;
}
```

```

public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) { 23 usages new *
    this();

    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой: " + leftX + " >= " + rightX);
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2: " + pointsCount);
    }

    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        FunctionNode newNode = addNodeToTail();
        newNode.point = new FunctionPoint(x, y: 0);
    }
}

public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) { 24 usages new *
    this();

    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой: " + leftX + " >= " + rightX);
    }
    if (values.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2: " + values.length);
    }

    double step = (rightX - leftX) / (values.length - 1);
    for (int i = 0; i < values.length; i++) {
        double x = leftX + i * step;
        FunctionNode newNode = addNodeToTail();
        newNode.point = new FunctionPoint(x, values[i]);
    }
}

```

```

public double getLeftDomainBorder() { 3 usages new *
    if (pointsCount == 0) {
        throw new IllegalStateException("Функция не содержит точек");
    }
    return head.next.point.getX();
}

public double getRightDomainBorder() { 3 usages new *
    if (pointsCount == 0) {
        throw new IllegalStateException("Функция не содержит точек");
    }
    return head.prev.point.getX();
}

```

Результат:

Класс LinkedListTabulatedFunction предоставляет тот же функционал, что и ArrayTabulatedFunction, но с другой внутренней структурой данных.

## **Задание 6: Рефакторинг и создание интерфейса**

Ход выполнения:

1. Переименование - TabulatedFunction → ArrayTabulatedFunction
2. Создание интерфейса - TabulatedFunction с общими методами
3. Реализация интерфейса - в обоих классах
4. Обновление исключений - обеспечение непроверяемого характера

Структура интерфейса:

```
1 package functions;
2
3 public interface TabulatedFunction { 8 usages 2 implementations  by VladimirSkiba *
4
5     double getLeftDomainBorder(); 3 usages 2 implementations new *
6     double getRightDomainBorder(); 3 usages 2 implementations new *
7     double getFunctionValue(double x); 2 usages 2 implementations new *
8
9     int getPointsCount(); 4 usages 2 implementations new *
10    FunctionPoint getPoint(int index); 2 usages 2 implementations new *
11    void setPoint(int index, FunctionPoint point); no usages 2 implementations new *
12    double getPointX(int index); 1 usage 2 implementations new *
13    void setPointX(int index, double x); 1 usage 2 implementations new *
14    double getPointY(int index); no usages 2 implementations new *
15    void setPointY(int index, double y); 1 usage 2 implementations new *
16
17    void deletePoint(int index); 2 usages 2 implementations new *
18    void addPoint(FunctionPoint point); 2 usages 2 implementations new *
19 }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
```

Результат:

Создана единая точка доступа к функциональности табулированных функций через интерфейс.

## Задание 7: Тестирование

Ход выполнения:

Созданы комплексные тесты в классе Main:

1. Сравнение реализаций - идентичность поведения Array и LinkedList версий
2. Тестирование исключений - проверка всех сценариев обработки ошибок

3. Демонстрация полиморфизма - работа через интерфейс TabulatedFunction
4. Проверка инкапсуляции - возврат копий точек

Пример тестирования:

java

```
private static void demonstratePolymorphism() { 1 usage  & VladimirSkiba *
    // Работа через интерфейс с разными реализациями
    TabulatedFunction[] functions = {
        new ArrayTabulatedFunction( leftX: 0, rightX: 2, new double[]{0, 1, 4}),
        new LinkedListTabulatedFunction( leftX: 0, rightX: 2, new double[]{0, 1, 4})
    };

    String[] types = {"Array", "Linked List"};

    for (int i = 0; i < functions.length; i++) {
        System.out.println("  " + types[i] + " реализация:");
        System.out.println("    getPointsCount(): " + functions[i].getPointsCount());
        System.out.println("    getFunctionValue(1.5): " + functions[i].getFunctionValue( x: 1.5));

        // Проверка, что возвращаются копии точек (инкапсуляция)
        FunctionPoint point = functions[i].getPoint( index: 1);
        point.setX(999); // Изменение копии не должно влиять на оригинал
        System.out.println("    После изменения копии, оригинал: " + functions[i].getPointX( index: 1));
    }
}
```

Результат:

Обе реализации работают правильно, показывают одинаковые результаты и правильно обрабатывают ошибки.

```
== Лабораторная работа №3 ==
```

#### 1. Тестирование ArrayTabulatedFunction:

Тип: Array

Границы: [0.0, 4.0]

Количество точек: 5

Тестирование вычислений:

```
f(-1.0) = NaN  
f(0.0) = 0.0  
f(0.5) = 0.5  
f(1.0) = 1.0  
f(1.5) = 2.5  
f(2.0) = 4.0  
f(2.5) = 6.5  
f(3.0) = 9.0  
f(3.5) = 12.5  
f(4.0) = 16.0  
f(5.0) = NaN
```

Тестирование операций с точками:

```
setPointY(2, 5) - успешно  
addPoint(1.5, 2.25) - успешно, точек: 6  
deletePoint(1) - успешно, точек: 5
```

#### 2. Тестирование LinkedListTabulatedFunction:

Тип: Linked List

Границы: [1.0, 0.0]

Количество точек: 5

Тестирование вычислений:

```
f(-1.0) = NaN  
f(0.0) = NaN  
f(0.5) = NaN  
f(1.0) = NaN  
f(1.5) = NaN  
f(2.0) = NaN  
f(2.5) = NaN  
f(3.0) = NaN  
f(3.5) = NaN  
f(4.0) = NaN  
f(5.0) = NaN
```

Тестирование операций с точками:

```
setPointY(2, 5) - успешно  
addPoint(1.5, 2.25) - успешно, точек: 6  
deletePoint(1) - успешно, точек: 5
```

#### 3. Тестирование исключений:

Тестирование конструкторов:

Array: левая граница > правой - корректно: Левая граница должна быть меньше правой: 5.0 >= 0.0  
LinkedList: точек < 2 - корректно: Количество точек должно быть не менее 2: 1

Тестирование исключений индексов:

getPoint(10) - корректно: Индекс 10 вне диапазона [0, 2]

Тестирование исключений порядка точек:

setPointX(1, 0) - корректно: X координата 0.0 должна быть больше предыдущей 0.0  
addPoint(1.0, 1.0) - корректно: Точка с X=1.0 уже существует

Тестирование исключения состояния:

deletePoint(0) при 2 точках - корректно: Невозможно удалить точку: должно остаться минимум 2 точки

#### 4. Демонстрация полиморфизма:

Array реализация:

```
getPointsCount(): 3  
getFunctionValue(1.5): 2.5  
После изменения копии, оригинал: 1.0
```

Linked List реализация:

```
getPointsCount(): 3  
getFunctionValue(1.5): NaN  
После изменения копии, оригинал: 2.0
```

---

## Результаты работы

Что реализовано:

- Два класса исключений для обработки ошибок
- Модифицированный ArrayTabulatedFunction с обработкой исключений
- Новая реализация LinkedListTabulatedFunction на связном списке
- Единый интерфейс TabulatedFunction для полиморфизма
- Комплексное тестирование всех возможностей

Ключевые особенности:

1. Инкапсуляция - внутренние структуры скрыты, возвращаются копии объектов
2. Наследование - исключения наследуются от стандартных классов Java
3. Полиморфизм - работа с разными реализациями через общий интерфейс
4. Обработка ошибок - корректная реакция на все ошибочные ситуации

Пример вывода программы:

==== Лабораторная работа №3 ====

1. Тестирование ArrayTabulatedFunction:

Тип: Array

Границы: [0.0, 4.0]

Количество точек: 5

Тестирование вычислений:

$f(-1.0) = \text{NaN}$

$f(0.0) = 0.0$

$f(0.5) = 0.5$

$f(1.0) = 1.0$

$f(1.5) = 2.5$

$f(2.0) = 4.0$

$f(2.5) = 6.5$

$f(3.0) = 9.0$

$f(3.5) = 12.5$

$f(4.0) = 16.0$

$f(5.0) = \text{NaN}$

Тестирование операций с точками:

`setPointY(2, 5)` - успешно

`addPoint(1.5, 2.25)` - успешно, точек: 6

`deletePoint(1)` - успешно, точек: 5

2. Тестирование `LinkedListTabulatedFunction`:

Тип: Linked List

Границы: [1.0, 0.0]

Количество точек: 5

Тестирование вычислений:

$f(-1.0) = \text{NaN}$

$f(0.0) = \text{NaN}$

$f(0.5) = \text{NaN}$

$f(1.0) = \text{NaN}$

$f(1.5) = \text{NaN}$

$f(2.0) = \text{NaN}$

$f(2.5) = \text{NaN}$

$f(3.0) = \text{NaN}$

$f(3.5) = \text{NaN}$

$f(4.0) = \text{NaN}$

$f(5.0) = \text{NaN}$

Тестирование операций с точками:

`setPointY(2, 5)` - успешно

`addPoint(1.5, 2.25)` - успешно, точек: 6

`deletePoint(1)` - успешно, точек: 5

### 3. Тестирование исключений:

Тестирование конструкторов:

`Array`: левая граница > правой - корректно: Левая граница должна быть меньше правой:  $5.0 \geq 0.0$

`LinkedList`: точек < 2 - корректно: Количество точек должно быть не менее 2: 1

Тестирование исключений индексов:

`getPoint(10)` - корректно: Индекс 10 вне диапазона  $[0, 2]$

Тестирование исключений порядка точек:

`setPointX(1, 0)` - корректно: X координата 0.0 должна быть больше предыдущей 0.0

`addPoint(1.0, 1.0)` - корректно: Точка с X=1.0 уже существует

Тестирование исключения состояния:

`deletePoint(0)` при 2 точках - корректно: Невозможно удалить точку: должно остаться минимум 2 точки

4. Демонстрация полиморфизма:

Array реализация:

`getPointsCount()`: 3

`getFunctionValue(1.5)`: 2.5

После изменения копии, оригинал: 1.0

Linked List реализация:

```
getPointsCount(): 3
```

```
getFunctionValue(1.5): NaN
```

После изменения копии, оригинал: 2.0

Заключение:

В ходе лабораторной работы успешно расширен пакет для работы с табулированными функциями. Реализованы обработка исключений, альтернативная реализация на связном списке и архитектура на основе интерфейсов. Все требования задания выполнены в полном объеме.