

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)

ОТЧЕТ ПО  
ЛАБОРАТОРНОЙ РАБОТЕ № 3

**«Работа с исключениями и  
интерфейсами в наборе классов  
табулированной функции»**

по курсу  
Объектно-ориентированное программирование

Выполнила: Егорова-Екимкова Яна,  
студент группы 6203-010302D

# Оглавление

## Задание №1

Перед началом практической части лабораторной работы мной были изучены классы исключений Java:

- Exception – базовый класс для проверяемых исключений.
- IndexOutOfBoundsException – выход за границы коллекции.
- ArrayIndexOutOfBoundsException – специализированное исключение для массивов.
- IllegalArgumentException – неверный аргумент метода.
- IllegalStateException – некорректное состояние объекта.

## Задание №2

Для этого задания в пакете functions мной были созданы классы исключений:

- FunctionPointIndexOutOfBoundsException – наследует IndexOutOfBoundsException, выбрасывается при недопустимом индексе точки.
- InappropriateFunctionPointException – наследует Exception, предотвращает нарушение порядка точек по оси X.

```
④ InappropriateFunctionPointException.java × ④ FunctionPointIndexOutOfBoundsException.java
1 package functions;
2
3 public class InappropriateFunctionPointException extends RuntimeException { 14 usages new *
4     public InappropriateFunctionPointException(String message) { super(message); } 13 usages new *
5 }
6
```

Рисунок 1

```
④ InappropriateFunctionPointException.java × ④ FunctionPointIndexOutOfBoundsException.java
1 package functions;
2
3 public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException 13 usages new *
4 {
5     public FunctionPointIndexOutOfBoundsException() { super(); } 3 usages new *
6     public FunctionPointIndexOutOfBoundsException(String message) { super(message); } 14 usages new *
7 }
8
```

Рисунок 2

## Задание №3

Задание 3 требовало модификации класса TabulatedFunction (далее переименован в ArrayTabulatedFunction) для обработки исключений:

- Конструкторы выбрасывают IllegalArgumentException при:
  - leftX  $\geq$  rightX (некорректные границы).
  - pointsCount  $< 2$  (недостаточное количество точек).
- Методы работы с точками (getPoint, setPoint, deletePoint и др.) выбрасывают FunctionPointIndexOutOfBoundsException при неверном индексе.
- Методы изменения X (setPoint, setPointX, addPoint) проверяют упорядоченность и выбрасывают InappropriateFunctionPointException при:
  - Нарушении порядка соседних точек.
  - Дублировании абсцисс.
- Метод deletePoint выбрасывает IllegalStateException при попытке удаления точки, если точек меньше трёх.

```
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount){ 7 usages new*
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Left border must be less than right border");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Points count must be more than 2");
    }

    this.pointsCount = pointsCount;
    this.array = new FunctionPoint[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);

    for (int i = 0; i<pointsCount; i++) {
        double x = leftX + step*i;
        array[i] = new FunctionPoint(x, y: 0.0);
    }
}
```

Рисунок 3

```
public ArrayTabulatedFunction(double leftX, double rightX, double[] values){ 3 usages new*
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Left border must be less than right border");
    }
    if (values.length < 2) {
        throw new IllegalArgumentException("Points count must be more than 2");
    }
    this.pointsCount = values.length;
    this.array = new FunctionPoint[values.length];
    double step = (rightX - leftX) / (values.length - 1);

    for (int i = 0; i < values.length; i++) {
        double x = leftX + step * i;
        array[i] = new FunctionPoint(x, values[i]);
    }
}
```

Рисунок 4

```

public FunctionPoint getPoint(int index) { 1 usage new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }
    return new FunctionPoint(array[index]);
}

public void setPoint(int index, FunctionPoint point) { 1 usage new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }

    if (index > 0 && point.getX() <= array[index-1].getX()) {
        throw new InappropriateFunctionPointException("X must be greater than previous point's X");
    }
    if (index < pointsCount - 1 && point.getX() >= array[index+1].getX()) {
        throw new InappropriateFunctionPointException("X must be less than next point's X");
    }

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

```

*Pucynok 5*

```

public void setPointX(int index, double x) { 1 usage new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }

    if (index > 0 && x <= array[index-1].getX()) {
        throw new InappropriateFunctionPointException("X must be greater than previous point's X");
    }
    if (index < pointsCount - 1 && x >= array[index+1].getX()) {
        throw new InappropriateFunctionPointException("X must be less than next point's X");
    }

    array[index].setX(x);
}

public void setPointY(int index, double y) { 1 usage new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }
    array[index].setY(y);
}

public double getPointX(int index) { 4 usages new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }
    return array[index].getX();
}

public double getPointY(int index) { 1 no usages new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }
    return array[index].getY();
}

```

*Pucynok 6*

```

public void deletePoint(int index) { 2 usages new *
    if (pointsCount < 3) {
        throw new IllegalStateException("Points count must be more than two");
    }
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }

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

public void addPoint(FunctionPoint point){ 2 usages new *
    FunctionPoint newPoint = new FunctionPoint(point);
    int newIndex = 0;
    while (newIndex<pointsCount && array[newIndex].getX()< newPoint.getX()){
        newIndex++;
    }
    if (newIndex< pointsCount && array[newIndex].getX() == newPoint.getX()){
        throw new InappropriateFunctionPointException("This X is already here");
    }
    if (pointsCount >= array.length) {
        FunctionPoint[] newArray = new FunctionPoint[array.length * 2];
        System.arraycopy(array,  srcPos: 0, newArray,  destPos: 0, pointsCount);
        array = newArray;
    }
    for (int i = pointsCount; i>newIndex; i--) {
        array[i] = array[i - 1];
    }
    array[newIndex] = newPoint;
    pointsCount++;
}

```

Рисунок 7

## Задание №4

По заданию 4 я реализовала класс `LinkedListTabulatedFunction` для хранения точек в двусвязном циклическом списке с выделенной головой:

- Внутренний класс `FunctionNode`:
  - Содержит поля: `FunctionPoint point`, `FunctionNode prev`, `FunctionNode next`.
  - Инкапсулирован как `private static`, что исключает доступ извне
- Оптимизации в `LinkedListTabulatedFunction`:
  - Поля `lastNode` и `lastIndex` кэшируют последний доступный узел для ускорения последовательных операций.
  - Метод `getNodeByIndex` выбирает кратчайший путь (с головы или хвоста).

- Реализованы методы работы со списком:
  - `addNodeToTail` – добавление в конец.
  - `addNodeByIndex` – вставка по индексу.
  - `deleteNodeByIndex` – удаление по индексу.

```
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction { 4 usages new *

  private static class FunctionNode { 39 usages new *

    // узлы
    private FunctionNode next; 31 usages
    private FunctionNode prev; 29 usages

    // информац. поле
    private FunctionPoint point; 32 usages

    // конструкторы
    private FunctionNode(FunctionPoint point){...}

    private FunctionNode(FunctionPoint point, FunctionNode prev, FunctionNode next){...}

    private FunctionNode(FunctionNode part){...}

    // геттеры для точки и узлов
    public FunctionPoint getPoint() { return point; } no usages new *
    public FunctionNode getNext(){ return next; } no usages new *
    public FunctionNode getPrev(){ return prev; } no usages new *

    //сеттеры для точки и узлов
    public void setPoint(FunctionPoint point){ this.point = point; } no usages new *
    public void setNext(FunctionNode next){ this.next = next; } no usages new *
    public void setPrev(FunctionNode prev){ this.prev = prev; } no usages new *

  }

}
```

*Рисунок 8*

```
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction { 4 usages new *

>  private static class FunctionNode {...}

  private FunctionNode head = new FunctionNode(new FunctionPoint( x: 0, y: 0)); //голова списка 33 usages
  private int pointsCount; 14 usages
  private int lastIndex; 11 usages
  private FunctionNode lastNode; 14 usages
```

*Рисунок 9*

```

// достать узел из списка по индексу
private FunctionNode getNodeByIndex(int index){ 9 usages new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }
    if (lastNode != null && lastIndex == index ){
        return lastNode;
    }

    FunctionNode temp;
    if (index > pointsCount / 2){
        temp = head.prev;
        for (int i = pointsCount -1; i > index; i--){
            temp = temp.prev;
        }
    } else {
        temp = head.next;
        for (int i = 0; i < index; i++){
            temp = temp.next;
        }
    }

    lastIndex = index;
    lastNode = temp;
    return lastNode;
}

```

Рисунок 10

```

private FunctionNode addNodeToTail(FunctionPoint point){  no usages new*
    FunctionNode newNode = new FunctionNode(point);
    newNode.prev = head.prev;
    head.prev.next = newNode;
    newNode.next = head;
    head.prev = newNode;
    pointsCount++;
    return newNode;
}

// добавить узел по индексу
private FunctionNode addNodeByIndex(int index, FunctionPoint point){  1 usage new*
    if (index > pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }
    FunctionNode newNode = new FunctionNode(point);
    FunctionNode temp = getNodeByIndex(index);
    newNode.prev = temp.prev;
    newNode.next = temp;
    temp.prev.next = newNode;
    temp.prev = newNode;
    pointsCount++;

    return newNode;
}

// удалить узел по индексу
private FunctionNode deleteNodeByIndex(int index){  1 usage new*
    if (index >= pointsCount || index < 0) {
        throw new FunctionPointIndexOutOfBoundsException("Index must not be beyond borders");
    }
    FunctionNode temp = getNodeByIndex(index);
    temp.prev.next = temp.next;
    temp.next.prev = temp.prev;
    return temp;
}

```

Рисунок 11

## Задание №5

Для задания 5 класс LinkedListTabulatedFunction был дополнен конструкторами и методами из TabulatedFunction:

- Конструкторы:
  - Равномерное создание точек по границам leftX, rightX и количеству pointsCount.
  - Инициализация через массив значений values.
- Оптимизированные методы:
  - getFunctionValue – поиск интервала начинается с lastNode.
  - addPoint – поиск позиции для вставки также стартует от lastNode.
- Исключения соответствуют логике ArrayTabulatedFunction.

```
// создание табулированной функции
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) { 11 usages new *
    if (leftX >= rightX) throw new IllegalArgumentException("Left border must be less than right border");
    if (pointsCount < 2) throw new IllegalArgumentException("Points count must be more than 2");

    this.pointsCount = pointsCount;
    double step = (rightX - leftX) / (pointsCount - 1);

    this.head = new FunctionNode(new FunctionPoint(x: 0, y: 0));
    head.next = head;
    head.prev = head;

    FunctionNode current = head;
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + step * i;
        FunctionNode newNode = new FunctionNode(new FunctionPoint(x, y: 0));

        newNode.prev = current;
        newNode.next = head;
        current.next = newNode;
        head.prev = newNode;
        current = newNode;
    }

    this.lastIndex = 0;
    this.lastNode = head.next;
}
```

Рисунок 12

```

// создание табулированной функции со значениями игрек из массива
public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) { 15 usages new*
    if (leftX >= rightX) throw new IllegalArgumentException("Left border must be less than right border");
    if (values.length < 2) throw new IllegalArgumentException("Points count must be more than 2");

    this.pointsCount = values.length;
    double step = (rightX - leftX) / (values.length - 1);

    this.head = new FunctionNode(new FunctionPoint(x: 0, y: 0));
    head.next = head;
    head.prev = head;

    FunctionNode current = head;
    for (int i = 0; i < values.length; i++) {
        double x = leftX + step * i;
        FunctionNode newNode = new FunctionNode(new FunctionPoint(x, values[i]));

        newNode.prev = current;
        newNode.next = head;
        current.next = newNode;
        head.prev = newNode;
        current = newNode;
    }

    this.lastIndex = 0;
    this.lastNode = head.next;
}

```

Рисунок 13

```

// вычисление значения функции в точке x
public double getFunctionValue(double x) { 2 usages new*
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
        return Double.NaN;
    }

    FunctionNode current = lastNode;
    int startIndex = lastIndex;

    // Оптимизация: определяем направление поиска от lastNode
    if (x < current.point.getX()) {
        while (current != head && current.point.getX() > x) {
            current = current.prev;
            startIndex--;
        }
    } else if (x > current.point.getX()) {
        while (current != head.prev && current.next.point.getX() < x) {
            current = current.next;
            startIndex++;
        }
    }
}

// совпадение
if (Math.abs(current.point.getX() - x) < 1e-10) {
    lastIndex = startIndex;
    lastNode = current;
    return current.point.getY();
}

```

Рисунок 14

```

// линейная интерполяция
FunctionNode next = current.next;
if (next == head) next = head.next;

double slope = (next.point.getY() - current.point.getY()) /
    (next.point.getX() - current.point.getX());
double result = current.point.getY() + slope * (x - current.point.getX());

lastIndex = startIndex;
lastNode = current;
return result;
}

```

Рисунок 15

```

// добавление новой точки (оптимизированная версия с использованием lastNode)
public void addPoint(FunctionPoint point) { 2 usages new *
    int newIndex = 0;
    FunctionNode current = (lastNode != null) ? lastNode : head.next;
    int currentIndex = (lastNode != null) ? lastIndex : 0;

    // Оптимизация: поиск позиции начинается с lastNode
    while (current != head && current.point.getX() < point.getX()) {
        current = current.next;
        currentIndex++;
    }

    if (current != head && Math.abs(current.point.getX() - point.getX()) < 1e-10) {
        throw new InappropriateFunctionPointException("This X is already here");
    }

    addNodeByIndex(currentIndex, point);
    lastIndex = currentIndex;
    lastNode = getNodeByIndex(currentIndex);
}

```

Рисунок 16

## Задание №6

Задание 6 требовало переименовать класс TabulatedFunction в ArrayTabulatedFunction.

Создан **интерфейс** TabulatedFunction, объединяющий общие методы:

- getPointsCount, getLeftDomainBorder, getFunctionValue.
- Методы управления точками: getPoint, setPoint, addPoint, deletePoint.

Оба класса (ArrayTabulatedFunction и LinkedListTabulatedFunction) реализуют интерфейс, что позволяет использовать полиморфизм.

```

package functions;

public interface TabulatedFunction { 5 usages 2 implementations ▾ Егорова-Екимкова Яна *

    // получение количества точек табуляции
    int getPointsCount(); 3 usages 2 implementations new *

    // получение левой границы области определения
    double getLeftDomainBorder(); 2 usages 2 implementations new *

    // получение правой границы области определения
    double getRightDomainBorder(); 2 usages 2 implementations new *

    // получение значения функции в точке x
    double getFunctionValue(double x); 2 usages 2 implementations new *

    // получение точки по индексу
    FunctionPoint getPoint(int index); 1 usage 2 implementations new *

    // замена точки по индексу
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; 1 usage 2 implementations new *

    // получение координаты X точки по индексу
    double getPointX(int index); 4 usages 2 implementations new *

    // установка координаты X точки по индексу
    void setPointX(int index, double x) throws InappropriateFunctionPointException; 1 usage 2 implementations new *

    // получение координаты Y точки по индексу
    double getPointY(int index); no usages 2 implementations new *

    // установка координаты Y точки по индексу
    void setPointY(int index, double y); 1 usage 2 implementations new *
}

```

Рисунок 17

```

    // удаление точки по индексу
    void deletePoint(int index); 2 usages 2 implementations new *

    // добавление точки
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; 2 usages 2 implementations new *
}

```

Рисунок 18

## Задание №7

По заданию 7 нужно протестировать интерфейс TabulatedFunction в классе Main:

- Проверены исключения:
  - FunctionPointIndexOutOfBoundsException – неверный индекс.
  - InappropriateFunctionPointException – нарушение порядка X.
  - IllegalArgumentException – ошибки в конструкторах.
  - IllegalStateException – удаление при недостаточном количестве точек.

Вывод консоли после работы программы представлен на рис.24-

```

import functions.*;

public class Main { * Егорова-Екимкова Яна *
    public static void main(String[] args) { new *
        // объявляем переменную типа интерфейса
        TabulatedFunction function;

        System.out.println("Testing ArrayTabulatedFunction");
        function = new ArrayTabulatedFunction( leftX: 0, rightX: 10, pointsCount: 5);
        testFunction(function);

        System.out.println("\nTesting LinkedListTabulatedFunction");
        function = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, new double[]{1, 2, 3, 4, 5});
        testFunction(function);
    }
}

```

Рисунок 19

```

public static void testFunction(TabulatedFunction function) { 2 usages Егорова-Екимкова Яна *
    System.out.println("Domain: [" +
        function.getLeftDomainBorder() + ", " +
        function.getRightDomainBorder() + "]");
    System.out.println("Points count: " + function.getPointsCount());

    System.out.println("Value at point 5: " + function.getFunctionValue( x: 5));

    // выход за границы области определения
    try {
        System.out.println("Value beyond left border: " + function.getFunctionValue( x: -1));
    } catch (Exception e) {
        System.out.println("Expected behavior: " + e.getMessage());
    }

    // неправильный индекс при получении точки
    try {
        FunctionPoint point = function.getPoint( index: 10);
        System.out.println("Point: " + point);
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Caught exception: " + e.getMessage());
    }

    // неправильный индекс при установке точки
    try {
        function.setPoint( index: 10, new FunctionPoint( x: 5, y: 10));
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Caught exception: " + e.getMessage());
    }
}

```

Рисунок 20

```

// нарушение порядка X при установке точки
try {
    function.setPointX( index: 2, x: function.getPointX( index: 1) - 1); // X меньше предыдущего
} catch (InappropriateFunctionPointException e) {
    System.out.println("Caught exception: " + e.getMessage());
}

// нарушение порядка X при добавлении точки
try {
    function.addPoint(new FunctionPoint(function.getPointX( index: 1), y: 100)); // Дублирование X
} catch (InappropriateFunctionPointException e) {
    System.out.println("Caught exception: " + e.getMessage());
}

// удаление точки при недостаточном количестве
try {
    // создаем функцию с минимальным количеством точек
    TabulatedFunction smallFunction;
    if (function instanceof ArrayTabulatedFunction) {
        smallFunction = new ArrayTabulatedFunction( leftX: 0, rightX: 2, pointsCount: 2);
    } else {
        smallFunction = new LinkedListTabulatedFunction( leftX: 0, rightX: 2, new double[]{1, 2});
    }
    smallFunction.deletePoint( index: 0);
} catch (IllegalStateException e) {
    System.out.println("Caught exception: " + e.getMessage());
}

```

Рисунок 21

```

// некорректные границы в конструкторе
try {
    if (function instanceof ArrayTabulatedFunction) {
        new ArrayTabulatedFunction( leftX: 10, rightX: 5, pointsCount: 3);
    } else {
        new LinkedListTabulatedFunction( leftX: 10, rightX: 5, new double[]{1, 2, 3});
    }
} catch (IllegalArgumentException e) {
    System.out.println("Caught constructor exception: " + e.getMessage());
}

// недостаточное количество точек в конструкторе
try {
    if (function instanceof ArrayTabulatedFunction) {
        new ArrayTabulatedFunction( leftX: 0, rightX: 5, pointsCount: 1);
    } else {
        new LinkedListTabulatedFunction( leftX: 0, rightX: 5, new double[]{1});
    }
} catch (IllegalArgumentException e) {
    System.out.println("Caught constructor exception: " + e.getMessage());
}

```

Рисунок 22

```

System.out.println("\nCorrect operations");
try {
    // добавление точки в правильном порядке
    double newX = (function.getPointX(index: 1) + function.getPointX(index: 2)) / 2;
    function.addPoint(new FunctionPoint(newX, y: 50));
    System.out.println("Point added successfully");

    // изменение Y
    function.setPointY(index: 2, y: 100);
    System.out.println("Y changed successfully");

    // удаление точки
    if (function.getPointsCount() > 3) {
        function.deletePoint(index: 1);
        System.out.println("Point deleted successfully");
    }

} catch (Exception e) {
    System.out.println("Unexpected exception: " + e.getMessage());
}

System.out.println("Final points count: " + function.getPointsCount());
}

```

Рисунок 23

```

C:\Users\sweet\.jdks\openjdk-24.0.2+12-54\bin\java.exe "-javaagent:C:\Program Files\Java\Testing\junit5\junit5-agent.jar" -Dfile.encoding=UTF-8 -jar C:\Users\sweet\IdeaProjects\Java\src\main\java\com\olegjavastudy\arraytabulatedfunction\src\main\java\com\olegjavastudy\arraytabulatedfunction\ArrayTabulatedFunction.jar
testing ArrayTabulatedFunction
domain: [0.0, 10.0]
pointscount: 5
initial state:
current points (5):
[0] x=0,00, y=0,00
[1] x=2,50, y=0,00
[2] x=5,00, y=0,00
[3] x=7,50, y=0,00
[4] x=10,00, y=0,00

value at point 5: 0.0
value beyond left border: NaN
caught exception: Index must not be beyond borders
caught exception: Index must not be beyond borders
caught exception: X must be greater than previous point's X
caught exception: This X is already here

```

Рисунок 24

```
correct operations
before correct operations:
current points (5):
[0] x=0,00, y=0,00
[1] x=2,50, y=0,00
[2] x=5,00, y=0,00
[3] x=7,50, y=0,00
[4] x=10,00, y=0,00

adding point at x=3.75
point added successfully
after adding point:
current points (6):
[0] x=0,00, y=0,00
[1] x=2,50, y=0,00
[2] x=3,75, y=50,00
[3] x=5,00, y=0,00
[4] x=7,50, y=0,00
[5] x=10,00, y=0,00
```

*Pucynok 25*

```
changing Y at index 2 to 100
y changed successfully
after changing y:
current points (6):
[0] x=0,00, y=0,00
[1] x=2,50, y=0,00
[2] x=3,75, y=100,00
[3] x=5,00, y=0,00
[4] x=7,50, y=0,00
[5] x=10,00, y=0,00

deleting point at index 1
point deleted successfully
final state after deletion:
current points (5):
[0] x=0,00, y=0,00
[1] x=3,75, y=100,00
[2] x=5,00, y=0,00
[3] x=7,50, y=0,00
[4] x=10,00, y=0,00
```

*Рисунок 26*

```
testing valid X change
changing X at index 1 to 4.375
x changed successfully
after valid x change:
current points (5):
[0] x=0,00, y=0,00
[1] x=4,38, y=100,00
[2] x=5,00, y=0,00
[3] x=7,50, y=0,00
[4] x=10,00, y=0,00

final points count: 5
small function before deletion attempt:
current points (2):
[0] x=0,00, y=0,00
[1] x=2,00, y=0,00

caught exception: Points count must be more than two
caught constructor exception: Left border must be less than right border
caught constructor exception: Points count must be more than 2
```

*Рисунок 27*

```
testing LinkedListTabulatedFunction
domain: [0.0, 10.0]
pointscount: 5
initial state:
current points (5):
[0] x=0,00, y=1,00
[1] x=2,50, y=2,00
[2] x=5,00, y=3,00
[3] x=7,50, y=4,00
[4] x=10,00, y=5,00

value at point 5: 3.0
value beyond left border: NaN
caught exception: Index must not be beyond borders
caught exception: Index must not be beyond borders
caught exception: X must be greater than previous point's X
caught exception: This X is already here

correct operations
before correct operations:
current points (5):
[0] x=0,00, y=1,00
[1] x=2,50, y=2,00
[2] x=5,00, y=3,00
[3] x=7,50, y=4,00
[4] x=10,00, y=5,00
```

Рисунок 28

```
testing valid X change
changing X at index 1 to 4.375
x changed successfully
after valid x change:
current points (5):
[0] x=0,00, y=1,00
[1] x=4,38, y=100,00
[2] x=5,00, y=3,00
[3] x=7,50, y=4,00
[4] x=10,00, y=5,00

final points count: 5
small function before deletion attempt:
current points (2):
[0] x=0,00, y=1,00
[1] x=2,00, y=2,00

caught exception: Points count must be more than two
caught constructor exception: Left border must be less than right border
caught constructor exception: Points count must be more than 2

Process finished with exit code 0
```

*Рисунок 29*