

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)

Институт информатики и кибернетики

Кафедра информационных систем и технологий

**ОТЧЕТ  
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

*«Реализация табулированных функций на основе связанного  
списка и обработка исключений» по курсу Объектно-  
ориентированное программирование*

Выполнил студент:  
Мацюк А.А.  
гр. 6203-010302

## Задание №1

Самара 2025

Я продолжил работу с пакетом functions. Создал новые классы исключений для обработки ошибок, возникающих при работе с табулированными функциями:

- InappropriateFunctionPointException – для случаев нарушения порядка точек по оси X или попытки добавить точку с уже существующим X.
- FunctionPointIndexOutOfBoundsException – для ситуаций выхода за границы индексов при обращении к точкам функции.

Эти классы наследуются от стандартных исключений Java и позволяют точно идентифицировать источник ошибки.

```
public InappropriateFunctionPointException() { super(); }

public InappropriateFunctionPointException(String message) { super(message); }

public InappropriateFunctionPointException(String message, Throwable cause) { super(message, cause); }

public InappropriateFunctionPointException(Throwable cause) { super(cause); }

}

public FunctionPointIndexOutOfBoundsException() { super(); }

public FunctionPointIndexOutOfBoundsException(String message) { super(message); }

public FunctionPointIndexOutOfBoundsException(String message, Throwable cause) { no usages
    super(message);
    this.initCause(cause);
}
|
```

## Задание №2

Разработал иерархию классов для представления табулированных функций:

- TabulatedFunction – интерфейс, определяющий общий контракт для всех табулированных функций.
- ArrayTabulatedFunction – реализация функции на основе массива точек.
- LinkedListTabulatedFunction – реализация функции на основе двусвязного списка.

Обе реализации поддерживают одинаковый набор операций, что обеспечивает полиморфизм и возможность взаимозаменяемости.

```

public interface TabulatedFunction { 10 usages 2 implementations
    int getPointsCount(); 4 usages 2 implementations
    double getPointX(int index); 6 usages 2 implementations
    double getPointY(int index); 2 usages 2 implementations
    void setPointX(int index, double x) throws InappropriateFunctionPointException; 3 usages 2 implementations
    void setPointY(int index, double y); 2 usages 2 implementations
    double getLeftDomainBorder(); 3 usages 2 implementations
    double getRightDomainBorder(); 3 usages 2 implementations
    double getFunctionValue(double x); 5 usages 2 implementations
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; 6 usages 2 implementations
    void deletePoint(int index); 5 usages 2 implementations
    FunctionPoint getPoint(int index); 2 usages 2 implementations
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; 2 usages 2 implementations
}

public class ArrayTabulatedFunction implements TabulatedFunction { 5 usages
    private FunctionPoint[] points; 31 usages
    private int pointsCount; 23 usages
    private static final double Epsilon = 1e-10; 10 usages

    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) { 4 usages
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

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

        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            this.points[i] = new FunctionPoint(x, 0.0);
        }
    }

    public ArrayTabulatedFunction(double leftX, double rightX, double[] values){ 1 usage
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }
    }

    public class LinkedListTabulatedFunction implements TabulatedFunction { 4 usages
        private class FunctionNode { 22 usages
            FunctionPoint point; 19 usages
            FunctionNode prev; 11 usages
            FunctionNode next; 20 usages

            FunctionNode(FunctionPoint point) { this.point = point; }
        }

        private FunctionNode head; 18 usages
        private int pointsCount; 15 usages
        private static final double Epsilon = 1e-10; 10 usages

        public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) { 3 usages
            if (leftX >= rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой");
            if (pointsCount < 2) throw new IllegalArgumentException("Количество точек должно быть не менее 2");

            initHead();
            double xStep = (rightX - leftX) / (pointsCount - 1);
            for (int i = 0; i < pointsCount; i++) {
                double x = leftX + i * xStep;
                addNodeToTail().point = new FunctionPoint(x, 0);
            }
            this.pointsCount = pointsCount;
        }

        public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) { 1 usage
            if (leftX >= rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой");
            if (values.length < 2) throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }
    }
}

```

### Задание №3

В классе ArrayTabulatedFunction реализовал методы:

- Конструкторы с проверкой входных данных (границы, количество точек).
- Методы доступа к точкам с валидацией индексов.
- Метод getFunctionValue() с линейной интерполяцией.
- Методы addPoint(), deletePoint(), setPoint() с обработкой исключений.
- Автоматическое расширение массива при добавлении точек.

```
public class ArrayTabulatedFunction implements TabulatedFunction { 5 usages

    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {...}

    public ArrayTabulatedFunction(double leftX, double rightX, double[] values){...}

    public double getLeftDomainBorder() { return points[0].getX(); }

    public double getRightDomainBorder() { return points[pointsCount - 1].getX(); }

    public double getFunctionValue(double x){...}

    public int getPointsCount() { return pointsCount; }

    public FunctionPoint getPoint(int index){...}

    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {...}

    public double getPointX(int index){...}

    public void setPointX(int index, double x) throws InappropriateFunctionPointException {...}

    public double getPointY(int index){...}

    public void setPointY(int index, double y){...}

    public void deletePoint(int index){...}

    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {...}
}
```

### Задание №4

В классе LinkedListTabulatedFunction реализовал структуру двусвязного списка. Это обеспечивает:

- Эффективное добавление и удаление точек.
- Автоматическое поддержание порядка по X.
- Использование внутреннего класса FunctionNode для инкапсуляции узлов списка.
- Методы для навигации по списку: getNodeByIndex(), addNodeByIndex(), deleteNodeByIndex().

```
public class LinkedListTabulatedFunction implements TabulatedFunction { 4 usages
    private class FunctionNode { 22 usages
        FunctionPoint point; 19 usages
        FunctionNode prev; 11 usages
        FunctionNode next; 20 usages

        FunctionNode(FunctionPoint point) { this.point = point; }
    }

    private FunctionNode head; 18 usages
    private int pointsCount; 15 usages
    private static final double Epsilon = 1e-10; 10 usages

    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {...}

    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {...}

    public LinkedListTabulatedFunction(FunctionPoint[] points) {...}

    private void initHead() {...}

    private FunctionNode addNodeToTail() {...}

    private FunctionNode addNodeByIndex(int index) {...}

    private FunctionNode deleteNodeByIndex(int index) {...}

    private FunctionNode getNodeByIndex(int index) {...}

public class LinkedListTabulatedFunction implements TabulatedFunction { 4 usages
    public int getPointsCount() { return pointsCount; }

    public double getPointX(int index) { return getNodeByIndex(index).point.getX(); }

    public double getPointY(int index) { return getNodeByIndex(index).point.getY(); }

    public void setPointX(int index, double x) throws InappropriateFunctionPointException {...}

    public void setPointY(int index, double y) { getNodeByIndex(index).point.setY(y); }

    public double getLeftDomainBorder() { return head.next.point.getX(); }

    public double getRightDomainBorder() { return head.prev.point.getX(); }

    public double getFunctionValue(double x) {...}

    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {...}

    public void deletePoint(int index) { deleteNodeByIndex(index); }

    public FunctionPoint getPoint(int index) { return new FunctionPoint(getNodeByIndex(index).point); }

    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {...}
```

Создал класс Main для тестирования обеих реализаций. В нём реализованы:

- Метод testFunction() – тестирование основных операций для переданной функции.
- Метод testExceptions() – проверка обработки исключений в различных сценариях.
- Демонстрация работы интерполяции, добавления, удаления и изменения точек.
- Сравнение работы ArrayTabulatedFunction и LinkedListTabulatedFunction.

### **Задание №6**

В ходе тестирования проверял следующие сценарии:

- Корректность интерполяции внутри и вне области определения.
- Обработка некорректных входных данных в конструкторах.
- Проверка граничных значений индексов.
- Попытки нарушения порядка точек по X.
- Удаление точек при минимальном количестве.
- Добавление дублирующихся точек.

Все исключения корректно обрабатываются, программа не завершается аварийно.

Ниже прикрепил скриншоты консольного вывода запуска main:

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-  
==== ТЕСТИРОВАНИЕ ARRAY ====  
Функция: ArrayTabulatedFunction  
Все точки функции:  
x = -3.0  y = 9.0  
x = -2.0  y = 4.0  
x = -1.0  y = 1.0  
x = 0.0   y = 0.0  
x = 1.0   y = 1.0  
x = 2.0   y = 4.0  
x = 3.0   y = 9.0  
Границы: -3.0 - 3.0  
f(-2) = 4.0  
f(-4) = NaN  
Проверка интерполяции:  
f(-2.5) = 6.5  
f(0.5) = 0.5  
f(2.5) = 6.5  
Количество точек: 7  
Точка (-2.5, 6) успешно добавлена  
Точка с индексом 1 успешно изменена  
Точка с индексом 3 удалена  
Функция после операций:  
x = -3.0  y = 9.0  
x = -2.1  y = 6.2  
x = -2.0  y = 4.0  
x = 0.0   y = 0.0  
x = 1.0   y = 1.0  
x = 2.0   y = 4.0
```

```
x = 3.0  y = 9.0
-----
== ТЕСТИРОВАНИЕ LINKEDLIST ==
Функция: LinkedListTabulatedFunction
Все точки функции:
x = -3.0  y = 9.0
x = -2.0  y = 4.0
x = -1.0  y = 1.0
x = 0.0   y = 0.0
x = 1.0   y = 1.0
x = 2.0   y = 4.0
x = 3.0   y = 9.0
Границы: -3.0 - 3.0
f(-2) = 4.0
f(-4) = NaN
Проверка интерполяции:
f(-2.5) = 6.5
f(0.5) = 0.5
f(2.5) = 6.5
Количество точек: 7
Точка (-2.5, 6) успешно добавлена
Точка с индексом 1 успешно изменена
Точка с индексом 3 удалена
Функция после операций:
x = -3.0  y = 9.0
x = -2.1  y = 6.2
x = -2.0  y = 4.0
x = 0.0   y = 0.0
```

```
== ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ ==
```

1. Тестирование некорректных конструкторов:

Ошибка: Левая граница должна быть меньше правой

Ошибка: Количество точек должно быть не менее 2

2. Тестирование выхода за границы индексов:

Ошибка индекса: Индекс выходит за границы: -1

Ошибка индекса: Индекс выходит за границы: 10

3. Тестирование нарушения упорядоченности:

Ошибка порядка: X точки должен быть больше предыдущей точки

4. Тестирование добавления точки с существующим X:

Ошибка дублирования: Точка с таким X уже существует

5. Тестирование удаления при малом количестве точек:

Ошибка удаления: Нельзя удалить точку: должно остаться минимум 2 точки

6. Тестирование методов setPoint:

Ошибка установки: X точки должен быть меньше следующей точки

7. Тестирование граничных случаев:

Ошибка: Индекс выходит за границы: -5

Ошибка: Индекс выходит за границы: 15

8. Тестирование с LinkedListTabulatedFunction:

Ошибка в LinkedList: Точка с таким X уже существует

7. Тестирование граничных случаев:

Ошибка: Индекс выходит за границы: -5

Ошибка: Индекс выходит за границы: 15

8. Тестирование с LinkedListTabulatedFunction:

Ошибка в LinkedList: Точка с таким X уже существует

9. Дополнительные тесты исключений:

Ошибка setPointY: Индекс выходит за границы: -1

Ошибка удаления последних точек: Нельзя удалить точку: должно остаться минимум 2 точки

10. Тестирование корректных операций:

Неожиданная ошибка: Точка с таким X уже существует

```
Process finished with exit code 0
```