# Лабораторная работа №3
# Болясников Вадим Андреевич 6204-010302D

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.

## Задание 1

Я ознакомился со следующими классами исключений, которые входят в API Java:
java.lang.Exception – базовый класс для всех исключений.
java.lang.IndexOutOfBoundsException – проверка выхода за границу массива, списка и тд.
java.lang.ArrayIndexOutOfBoundsException – проверка обращения к массиву с неверном размером.
java.lang.IllegalArgumentException – проверка на правильность аргумента.
java.lang.IllegalStateException – проверка правильности состояния.

## Задание 2

В ходе выполнения второго задания я создал два класса, в моей конфигурации functions, Вот их реализация:

functions > FunctionPointIndexOutOfBoundsException.java > FunctionPointIndexOutOfBoundsException

```java
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException {

    // Конструктор по умолчанию
    public FunctionPointIndexOutOfBoundsException() {
        super();
    }

    // Конструктор с сообщением об ошибке
    public FunctionPointIndexOutOfBoundsException(String message) {
        super(message);
    }

    // Конструктор с указанием проблемного индекса
    public FunctionPointIndexOutOfBoundsException(int index) {
        super("Точка выходит за границы: " + index);
    }
}
```

Рисунок 1 – Реализация класса FunctionPointIndexOutOfBoundsException

functions > InappropriateFunctionPointException.java > Language Support for Java(TM) by Red Hat > Inapp

```java
package functions;

public class InappropriateFunctionPointException extends Exception {

    // Конструктор по умолчанию
    public InappropriateFunctionPointException() {
        super();
    }

    // Конструктор с сообщением об ошибке
    public InappropriateFunctionPointException(String message) {
        super(message);
    }

    //Конструктор с сообщением и причиной исключения

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

Рисунок 2 – Реализация класса InappropriateFunctionPointException

## Задание 3

В начале задания, я изменил оба конструктора и добавил в них проверку на то, что левая граница больше или ровна правой, а также что кол-во точек больше двух:

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

Рисунок 3 – Реализация проверок в первом конструкторе

Во втором конструкторе мы заменили pointsCounts на values.length

Далее я добавил в следующие методы исключение FunctionsPointIndexOutOfBoundsException:

getPoint(), setPoint(), getPointX(), setPointX(), getPointY(), setPointY() и deletePoint() они выбрасывают FunctionPointIndexutOfBoundsException, если переданный в метод номер выходит за границы набора точек.

```
if (index < 0 || index >= pointsCount) {
    throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс точки: " + index);}
```

Рисунок 4 – Проверка на корректность индекса

Добавил исключение InappropriateFunctionPointException в методы SetPoint() и SetPointX()

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

Рисунок 5 – Проверка что x в интервале

Также добавил такое же исключение в метод addPoint, который проверял, существует ли точка перед ее добавлением.

Также дописал методе deletePoint иключение IllegalStateException, которая проверяет, чтобы мы не удалили одну из двух точек

```
if (pointsCount < 3) {
    throw new IllegalStateException(s:"Невозможно удалить точку: должно остаться минимум 2 точки");
}
```

Рисунок 6 – Проверка на возможность удаление точки

## Задание 4 и 5

В этом задание нужно было реализовать двусвязный циклический список. У меня он состоит из двух классов: FunctionNode и LinkedListTabulatedFunction. Где первый отвечает за узлы списка, а второй за весь список. Класс LinkedListTabulatedFunction совмещает в себе сразу две функции, он описывает связные список и работу с ним, а также описывает работу с табулированной функцией и ее точками. В нем я реализовал несколько методов, таких как:

- `FunctionNode getNodeByIndex(int index)` – возвращает ссылку на объект элемента списка по его номеру
- `FunctionNode addNodeToTail()` – добавляет элемент в конец списка и возвращает ссылку на объект данного элемента
- `FunctionNode addNodeByIndex(int index)` – добавляет элемент в указанный индекс и возвращает ссылку на объект этого элемента
- `FunctionNode deleteNodeByIndex(int index)` – удаляет элемент по индексу и возвращает ссылку на объект удаленного элемент

Далее в ходе 5 задания, я реализовал в классе LinkedListTabulatedFunction такие же методы, которые есть в TabulatedFunction, но теперь мы работает с узлами и определенными геттерами и сеттерами

Весь код FunctionNode  и LinkedListTabulatedFunction

## Задание 6

В данном задание, я переименовал TabulatedFunction в  ArrayTabulatedFunction, и заместо этого в TabulatedFunction.java добавил интерфейс, теперь оба класса ArrayTabulatedFunction и LinkedListTabulatedFunction реализуют интерфейс

```java
package functions;

public interface TabulatedFunction {
    // Методы получения границ области определения
    double getLeftDomainBorder();
    double getRightDomainBorder();

    // Метод получения значения функции
    double getFunctionValue(double x);

    // Методы работы с точками
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
    double getPointX(int index);
    void setPointX(int index, double x) throws InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Рисунок 7 – Интерфейс.

## Задание 7

Проверка работы программы, путем вывода в консоль проверок:

```
[VadimBolashikov@MacBookAir Lab-3-2025 % java main.java
=== ТЕСТИРОВАНИЕ ARRAY TABULATED FUNCTION ===

1. Нормальная работа ArrayTabulatedFunction
Количество точек: 5
Область определения: [0.0, 4.0]
f(-1,0) = не определено (вне области)
f(0,0) = 0,00
f(0,5) = 0,50
f(2,0) = 4,00
f(4,0) = 16,00
f(5,0) = не определено (вне области)

2. Тестирование исключений ArrayTabulatedFunction
2.1. FunctionPointIndexOutOfBoundsException:
Выброшено исключение для индекса -1: Некорректный индекс точки: -1
Выброшено исключение для индекса 10: Некорректный индекс точки: 10
Выброшено исключение для setPointY с индексом -1: Некорректный индекс точки: -1
```

```
2.2. InappropriateFunctionPointException:
Выброшено исключение при X меньше предыдущего: X координата точки должна быть бо
льше предыдущей
Выброшено исключение при X больше следующего: X координата точки должна быть мен
ьше следующей
Выброшено исключение при добавлении точки с существующим X: Точка с таким X уже
существует

2.3. IllegalStateException:
Успешно удалена точка при pointsCount = 3
Выброшено исключение при попытке удалить точку когда pointsCount < 3: Невозможно
 удалить точку: должно остаться минимум 2 точки

Тестирование ArrayTabulatedFunction завершено

=== ТЕСТИРОВАНИЕ LINKED LIST TABULATED FUNCTION ===

1. Нормальная работа LinkedListTabulatedFunction
Количество точек: 5
Область определения: [0.0, 4.0]
f(-1,0) = не определено (вне области)
f(0,0) = 0,00
f(0,5) = 0,50
f(2,0) = 4,00
f(4,0) = 16,00
f(5,0) = не определено (вне области)

2. Тестирование исключений LinkedListTabulatedFunction
2.1. FunctionPointIndexOutOfBoundsException:
Выброшено исключение для индекса -1: Некорректный индекс: -1
Выброшено исключение для индекса 10: Некорректный индекс: 10
Выброшено исключение для setPointY с индексом -1: Некорректный индекс: -1

2.2. InappropriateFunctionPointException:
Выброшено исключение при X меньше предыдущего: X координата точки должна быть бо
льше предыдущей
Выброшено исключение при X больше следующего: X координата точки должна быть мен
ьше следующей
Выброшено исключение при добавлении точки с существующим X: Точка с таким X уже
существует

2.3. IllegalStateException:
Успешно удалена точка при pointsCount = 3
Выброшено исключение при попытке удалить точку когда pointsCount < 3: Невозможно
 удалить точку: должно остаться минимум 2 точки

Тестирование LinkedListTabulatedFunction завершено

=== ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ В КОНСТРУКТОРАХ ===
1. IllegalArgumentException - некорректные границы:
ArrayTabulatedFunction: Выброшено исключение при leftX == rightX: Левая граница
должна быть меньше правой
LinkedListTabulatedFunction: Выброшено исключение при leftX == rightX: Левая гра
ница должна быть меньше правой
ArrayTabulatedFunction: Выброшено исключение при leftX > rightX: Левая граница д
олжна быть меньше правой

2. IllegalArgumentException - недостаточное количество точек:
ArrayTabulatedFunction: Выброшено исключение при pointsCount < 2: Количество точ
ек должно быть не менее двух
LinkedListTabulatedFunction: Выброшено исключение при массиве длины < 2: Количес
тво точек должно быть не менее двух

=== ДЕТАЛЬНОЕ ТЕСТИРОВАНИЕ ОПЕРАЦИЙ С ТОЧКАМИ ===
=== ТЕСТИРОВАНИЕ ОПЕРАЦИЙ С ТОЧКАМИ В LinkedListTabulatedFunction ===
Исходная функция:
Точки функции (5): (0,00, 0,00), (1,00, 1,00), (2,00, 4,00), (3,00, 9,00), (4,00
, 16,00)

1. ТЕСТИРОВАНИЕ ДОБАВЛЕНИЯ ТОЧЕК:
Добавлена точка в середину: (0.5, 0.25)
Точки функции (6): (0,00, 0,00), (0,50, 0,25), (1,00, 1,00), (2,00, 4,00), (3,00
, 9,00), (4,00, 16,00)
Добавлена точка в начало: (-0.5, 0.25)
Точки функции (7): (-0,50, 0,25), (0,00, 0,00), (0,50, 0,25), (1,00, 1,00), (2,0
0, 4,00), (3,00, 9,00), (4,00, 16,00)
Добавлена точка в конец: (5.0, 25.0)
Точки функции (8): (-0,50, 0,25), (0,00, 0,00), (0,50, 0,25), (1,00, 1,00), (2,0
0, 4,00), (3,00, 9,00), (4,00, 16,00), (5,00, 25,00)
Корректно обработана попытка добавить точку с существующим X: Точка с таким X уж
е существует

2. ТЕСТИРОВАНИЕ УДАЛЕНИЯ ТОЧЕК:
Удаляем точку с индексом 3 (X=1.0)
Точки функции (7): (-0,50, 0,25), (0,00, 0,00), (0,50, 0,25), (2,00, 4,00), (3,0
0, 9,00), (4,00, 16,00), (5,00, 25,00)
Удаляем точку с индексом 0 (X=-0.5)
Точки функции (6): (0,00, 0,00), (0,50, 0,25), (2,00, 4,00), (3,00, 9,00), (4,00
, 16,00), (5,00, 25,00)
Удаляем точку с индексом 5 (X=5.0)
Точки функции (5): (0,00, 0,00), (0,50, 0,25), (2,00, 4,00), (3,00, 9,00), (4,00
, 16,00)
Попытка удалить точку при pointsCount = 5
Ошибка: точка была удалена, хотя должно было быть исключение
```

Рисунок 8-10 – Проверки
По ним мы видим, что программа работает корректно и мы радуемся!!

## Код FunctionNode и LinkedListTabulatedFunction

```
package functions;

// Внутренний класс элемента списка
class FunctionNode {
    // Информационное поле для хранения данных точки
    public FunctionPoint point;

    // Ссылки на предыдущий и следующий элемент
    public FunctionNode prev;
    public FunctionNode next;

    // Конструктор по умолчанию
    public FunctionNode() {
        this.point = null;
        this.prev = null;
        this.next = null;
    }

    // Конструктор с точкой
    public FunctionNode(FunctionPoint point) {
        this.point = point;
        this.prev = null;
        this.next = null;
    }

    // Конструктор с точкой и ссылками
    public FunctionNode(FunctionPoint point, FunctionNode prev, FunctionNode next) {
        this.point = point;
        this.prev = prev;
        this.next = next;
    }
}



package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction{
    // Голова списка (не содержит данных, всегда существует)
    private FunctionNode head;

    // Количество значащих элементов (без головы)
    private int pointsCount;
```

```
    // Вспомогательные поля для оптимизации доступа
    private FunctionNode lastAccessedNode;
    private int lastAccessedIndex;

    // Конструктор по умолчанию (пустой список)
    public LinkedListTabulatedFunction() {
        // Создаем голову, которая ссылается сама на себя
        head = new FunctionNode();
        head.prev = head;
        head.next = head;
        pointsCount = 0;
        lastAccessedNode = head;
        lastAccessedIndex = -1;
    }

    // Конструктор с параметрами (равномерная сетка)
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
        this(); // Вызываем конструктор по умолчанию для инициализации головы

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

        // Создаем точки равномерной сетки
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            addNodeToTail().point = new FunctionPoint(x, 0);
        }
    }

    // Конструктор с параметрами (массив значений)
    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
        this(); // Вызываем конструктор по умолчанию для инициализации головы

        // Проверка условий конструктора
        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
двух");
        }

        // Создаем точки с заданными значениями
        double step = (rightX - leftX) / (values.length - 1);
```

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

    // Метод для получения узла по индексу с оптимизацией доступа
    private FunctionNode getNodeByIndex(int index) {
        // Проверка корректности индекса
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс: " +
index);
        }

        // Оптимизация: если запрашиваем тот же элемент, что и в прошлый раз
        if (lastAccessedIndex == index) {
            return lastAccessedNode;
        }

        // Оптимизация: если запрашиваем следующий элемент
        if (lastAccessedIndex != -1 && lastAccessedIndex == index - 1) {
            lastAccessedNode = lastAccessedNode.next;
            lastAccessedIndex = index;
            return lastAccessedNode;
        }

        // Оптимизация: если запрашиваем предыдущий элемент
        if (lastAccessedIndex != -1 && lastAccessedIndex == index + 1) {
            lastAccessedNode = lastAccessedNode.prev;
            lastAccessedIndex = index;
            return lastAccessedNode;
        }

        // Поиск с ближайшего конца
        FunctionNode current;
        int currentIndex;

        if (index < pointsCount / 2) {
            // Ищем с начала
            current = head.next;
            currentIndex = 0;
            while (currentIndex < index) {
                current = current.next;
                currentIndex++;
            }
        } else {
            // Ищем с конца
            current = head.prev;
            currentIndex = pointsCount - 1;
            while (currentIndex > index) {
                current = current.prev;
                currentIndex--;
            }
```

```
        }

        // Сохраняем для будущей оптимизации
        lastAccessedNode = current;
        lastAccessedIndex = index;

        return current;
    }

    // Метод для добавления узла в конец списка
    private FunctionNode addNodeToTail() {
        FunctionNode newNode = new FunctionNode();

        // Вставляем перед головой (что эквивалентно концу списка)
        newNode.prev = head.prev;
        newNode.next = head;

        head.prev.next = newNode;
        head.prev = newNode;

        pointsCount++;
        lastAccessedNode = newNode;
        lastAccessedIndex = pointsCount - 1;

        return newNode;
    }

    // Метод для добавления узла по индексу
    private FunctionNode addNodeByIndex(int index) {
        if (index < 0 || index > pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс для
вставки: " + index);
        }

        if (index == pointsCount) {
            // Вставка в конец
            return addNodeToTail();
        }

        // Находим узел, перед которым нужно вставить новый
        FunctionNode nextNode = getNodeByIndex(index);
        FunctionNode prevNode = nextNode.prev;

        // Создаем новый узел
        FunctionNode newNode = new FunctionNode();
        newNode.prev = prevNode;
        newNode.next = nextNode;

        // Обновляем ссылки соседних узлов
        prevNode.next = newNode;
        nextNode.prev = newNode;

        pointsCount++;
```

```
        lastAccessedNode = newNode;
        lastAccessedIndex = index;

        return newNode;
    }

    // Метод для удаления узла по индексу
    private FunctionNode deleteNodeByIndex(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс для
удаления: " + index);
        }

        if (pointsCount < 3) {
            throw new IllegalStateException("Невозможно удалить точку: должно остаться
минимум 2 точки");
        }

        // Находим удаляемый узел
        FunctionNode nodeToDelete = getNodeByIndex(index);

        // Обновляем ссылки соседних узлов
        nodeToDelete.prev.next = nodeToDelete.next;
        nodeToDelete.next.prev = nodeToDelete.prev;

        pointsCount--;

        // Сбрасываем кэш, если удалили кэшированный элемент
        if (lastAccessedIndex == index) {
            lastAccessedNode = head;
            lastAccessedIndex = -1;
        } else if (lastAccessedIndex > index) {
            lastAccessedIndex--;
        }

        return nodeToDelete;
    }

    // Методы табулированной функции (аналогичные TabulatedFunction)

    public double getLeftDomainBorder() {
        if (pointsCount == 0) {
            return Double.NaN;
        }
        return head.next.point.getX();
    }

    public double getRightDomainBorder() {
        if (pointsCount == 0) {
            return Double.NaN;
        }
        return head.prev.point.getX();
    }
```

```
    public double getFunctionValue(double x) {
        if (pointsCount == 0 || x < getLeftDomainBorder() || x >
getRightDomainBorder()) {
            return Double.NaN;
        }

        FunctionNode current = head.next;

        // Проверяем первую точку
        if (x == current.point.getX()) {
            return current.point.getY();
        }

        // Ищем интервал
        for (int i = 0; i < pointsCount - 1; i++) {
            double curX = current.point.getX();
            double nextX = current.next.point.getX();

            if (x == nextX) {
                return current.next.point.getY();
            }

            if (x > curX && x < nextX) {
                return linearInterpolation(current.point, current.next.point, x);
            }

            current = current.next;
        }

        return Double.NaN;
    }
    private double linearInterpolation(FunctionPoint p1, FunctionPoint p2, double x) {
        double x1 = p1.getX();
        double y1 = p1.getY();
        double x2 = p2.getX();
        double y2 = p2.getY();

        return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
    }

    public int getPointsCount() {
        return pointsCount;
    }

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

    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
        FunctionNode node = getNodeByIndex(index);
```

```
        // Проверка упорядоченности точек
        if (index > 0 && point.getX() <= getNodeByIndex(index - 1).point.getX()) {
            throw new InappropriateFunctionPointException("X координата точки должна
быть больше предыдущей");
        }
        if (index < pointsCount - 1 && point.getX() >= getNodeByIndex(index +
1).point.getX()) {
            throw new InappropriateFunctionPointException("X координата точки должна
быть меньше следующей");
        }

        node.point = new FunctionPoint(point);
    }

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

    public void setPointX(int index, double x) throws
InappropriateFunctionPointException {
        FunctionNode node = getNodeByIndex(index);

        // Проверка упорядоченности точек
        if (index > 0 && x <= getNodeByIndex(index - 1).point.getX()) {
            throw new InappropriateFunctionPointException("X координата должна быть
больше предыдущей");
        }
        if (index < pointsCount - 1 && x >= getNodeByIndex(index + 1).point.getX()) {
            throw new InappropriateFunctionPointException("X координата должна быть
меньше следующей");
        }

        node.point.setX(x);
    }

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

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

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

    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
        double newX = point.getX();

        // Проверяем, не существует ли уже точка с таким X
        FunctionNode current = head.next;
```

```
        for (int i = 0; i < pointsCount; i++) {
            if (current.point.getX() == newX) {
                throw new InappropriateFunctionPointException("Точка с таким X уже
существует");
            }
            current = current.next;
        }

        // Находим позицию для вставки
        int insertIndex = 0;
        current = head.next;
        while (insertIndex < pointsCount && current.point.getX() < newX) {
            current = current.next;
            insertIndex++;
        }

        // Вставляем новую точку
        FunctionNode newNode = addNodeByIndex(insertIndex);
        newNode.point = new FunctionPoint(point);
    }


}
```