Лабораторная работа №3

Валиневич Владислав Александрович

группа: 6204-010302D

Цель работы: Дополнить пакет для работы с функциями одной переменной, заданными в табличной форме, добавив классы исключений, новый класс функций и базовый интерфейс.

Задание 1:
Изучил следующие классы исключений:

java.lang.Exception — базовый класс для проверяемых исключений.

java.lang.IndexOutOfBoundsException — исключение выхода за границы индекса.

java.lang.ArrayIndexOutOfBoundsException — подкласс *IndexOutOfBoundsException* для массивов.

java.lang.IllegalArgumentException — исключение для недопустимых аргументов.

java.lang.IllegalStateException — исключение для недопустимого состояния объекта.

Задание 2:
Создал два класса исключений в пакете functions:
FunctionPointIndexOutOfBoundsException — наследуется от *IndexOutOfBoundsException*.

InappropriateFunctionPointException — наследуется от *Exception*.

```
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
13
```

```java
package functions;

public class InappropriateFunctionPointException extends Exception {
    public InappropriateFunctionPointException() {
        super();
    }

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

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

Задание 3:

В разработанный ранее класс TabulatedFunction внес изменения, обеспечивающие генерацию исключений методами класса.

Получим: Конструкторы выбрасывают IllegalArgumentException при недопустимых параметрах.

Методы работы с точками выбрасывают FunctionPointIndexOutOfBoundsException при неверном индексе.

Методы setPoint(), setPointX(), addPoint() выбрасывают InappropriateFunctionPointException при нарушении порядка точек.

Метод deletePoint() выбрасывает IllegalStateException при попытке удалить точку, если точек меньше трёх.

```java
package functions;


public class TabulatedFunction {
    private FunctionPoint[] points; //массив точек
    private static final double EPSILON = 1e-10;

    // Конструктор 1:
    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Кол-во точек меньше 2");
        }

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

        // Заполняем массив точками с координатой x и с y=0
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            points[i] = new FunctionPoint(x, 0);
        }
    }

    // Конструктор 2:
    public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException("Кол-во точек меньше 2");
        }

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

        //заполняем массив заданным значением y
        for (int i = 0; i < values.length; i++) {
            double x = leftX + i * step;
            points[i] = new FunctionPoint(x, values[i]);
        }
    }

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

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

//Вычисляем значение функции
    @Override
    public double getFunctionValue(double x) {

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


        for (int i = 0; i < points.length - 1; i++) {
```

```
        for (int i = 0; i < points.length - 1; i++) {
            double x1 = points[i].getX();
            double x2 = points[i + 1].getX();
            double y1 = points[i].getY();
            double y2 = points[i + 1].getY();


            if (Math.abs(x - x1) < EPSILON) {
                return y1;
            }
            if (Math.abs(x - x2) < EPSILON) {
                return y2;
            }


            if (x > x1 && x < x2) {
                return y1 + (y2 - y1) / (x2 - x1) * (x - x1);
            }
        }


        return points[points.length - 1].getY();
    }

 // Количество точек
    @Override
    public int getPointsCount() {
        return points.length;

    }


    @Override
    public FunctionPoint getPoint(int index){
        if (index < 0 || index >= points.length) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс  : " + index);
        }
        return new FunctionPoint(points[index]);
    }

//Копируем точку
    @Override
    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
        if (index < 0 || index >= points.length) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс  : " + index);
        }

        if (index > 0 && point.getX() <= points[index - 1].getX()) {
            throw new InappropriateFunctionPointException("Координата точки X должна быть больше, чем координата предыдущей точки X");
        }
        if (index < points.length - 1 && point.getX() >= points[index + 1].getX()) {
            throw new InappropriateFunctionPointException("Координата точки X должна быть меньше координаты X следующей точки");
        }
        points[index] = new FunctionPoint(point);
    }
    @Override
    public double getPointX(int index) throws FunctionPointIndexOutOfBoundsException {
        if (index < 0 || index >= points.length){
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс  : " + index);
        }
        return points[index].getX();
    }
    @Override
    public void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
```

```java
    @Override
    public void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
        if (index < 0 || index >= points.length) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс  : " + index);
        }

        if (index > 0 && x <= points[index - 1].getX()) {
            throw new InappropriateFunctionPointException("Координата точки X должна быть больше, чем координата предыдущей точки X");
        }
        if (index < points.length - 1 && x >= points[index + 1].getX()) {
            throw new InappropriateFunctionPointException("Координата точки X должна быть меньше координаты X следующей точки");
        }
        points[index].setX(x);
    }

    @Override
    public double getPointY(int index) {
        if (index < 0 || index >= points.length) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс  : " + index);
        }
        return points[index].getY();
    }
    @Override
    public void setPointY(int index, double y) {
        if (index < 0 || index >= points.length) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс  : " + index);
        }
        points[index].setY(y);
    }

// Удаление точки
    @Override
    public void deletePoint(int index) {
        if (index < 0 || index >= points.length) {
            throw new FunctionPointIndexOutOfBoundsException("Некорректный индекс  : " + index);
        }

        if (points.length <= 2) {
            throw new IllegalStateException("Невозможно удалить точку, требуется минимум 2");
        }

        FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
        System.arraycopy(points, 0, newPoints, 0, index);
        System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
        points = newPoints;
    }

// Добавление точки
    @Override
    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
        FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
        int i = 0;

        while (i < points.length && point.getX() > points[i].getX()) {
            i++;
        }

        // Если точка с таким X уже существует - выбрасываем исключение
        if (i < points.length && Math.abs(point.getX() - points[i].getX()) < EPSILON) {
            throw new InappropriateFunctionPointException("Такая точка с координатой X уже существует");
        }

        // Копируем старый массив, вставляя новую точку в нужную позицию
        System.arraycopy(points, 0, newPoints, 0, i);
        newPoints[i] = new FunctionPoint(point);
```

Задание 4:

Создадим класс *LinkedListTabulatedFunction*, использующий двусвязный циклический список с выделенной головой. Он состоит из двух частей, в первой части мы реализуем методы для работы с ним:

Методы работы со списком:

getNodeByIndex(int index) — доступ к узлу по индексу с оптимизацией.

addNodeToTail() — добавление узла в конец списка.

addNodeByIndex(int index) — вставка узла по индексу.

deleteNodeByIndex(int index) — удаление узла по индексу.

```
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction {
    private static class FunctionNode {
        private FunctionPoint point;
        private FunctionNode prev;
        private FunctionNode next;

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

    private FunctionNode head;
    private int pointsCount;
    private static final double EPSILON = 1e-9;

    private void initializeList() {
        head = new FunctionNode(null);
        head.prev = head;
        head.next = head;
        pointsCount = 0;
    }

    private FunctionNode getNodeByIndex(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный индекс");
        }

        FunctionNode currentNode;


        if (index < pointsCount - index) {

            currentNode = head.next;
            for (int i = 0; i < index; i++) {
                currentNode = currentNode.next;
            }
        } else {

            currentNode = head.prev;
            for (int i = pointsCount - 1; i > index; i--) {
                currentNode = currentNode.prev;
            }
        }

        return currentNode;
    }

    private FunctionNode addNodeToTail() {
        FunctionNode newNode = new FunctionNode(null);
        FunctionNode tail = head.prev;

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

        pointsCount++;
        return newNode;
    }

    private FunctionNode addNodeByIndex(int index) {
        if (index < 0 || index > pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный индекс");
```

```
private FunctionNode addNodeByIndex(int index) {
    if (index < 0 || index > pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Неверный индекс");
    }

    if (index == pointsCount) {

        return addNodeToTail();
    }

    FunctionNode currentNode = getNodeByIndex(index);
    FunctionNode prevNode = currentNode.prev;
    FunctionNode newNode = new FunctionNode(null);

    prevNode.next = newNode;
    newNode.prev = prevNode;
    newNode.next = currentNode;
    currentNode.prev = newNode;

    pointsCount++;
    return newNode;
}

private FunctionNode deleteNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Неверный индекс");
    }

    if (pointsCount <= 2) {
        throw new IllegalStateException("Невозможно удалить точку, требуется минимум 2");
    }

    FunctionNode nodeToDelete = getNodeByIndex(index);
    FunctionNode prevNode = nodeToDelete.prev;
    FunctionNode nextNode = nodeToDelete.next;

    prevNode.next = nextNode;
    nextNode.prev = prevNode;

    pointsCount--;
    return nodeToDelete;
}
```

Задание 5:

Для реализации второй части необходимо создать в классе конструкторы и методы, аналогичные конструкторам и методам класса TabulatedFunction.
Конструкторы:

```java
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Кол-во точек меньше 2");
    }

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

    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        addNodeToTail().point = new FunctionPoint(x, 0);
    }
}

public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }
    if (values.length < 2) {
        throw new IllegalArgumentException("Кол-во точек меньше 2");
    }

    initializeList();
    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]);
    }
}
```

Методы получения значений функции, работы с точками, добавления и удаления точек:

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

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


@Override
public double getFunctionValue(double x) {
    if (x < getLeftDomainBorder() || x > getRightDomainBorder())
        return Double.NaN;

    FunctionNode cur = head.next;
    while (cur.next != head) {
        double x1 = cur.point.getX();
        double x2 = cur.next.point.getX();


        if (Math.abs(x - x1) < EPSILON)
            return cur.point.getY();
        if (Math.abs(x - x2) < EPSILON)
            return cur.next.point.getY();

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

@Override
public int getPointsCount() {
    return pointsCount;
}

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

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

    if (index > 0 && point.getX() <= getNodeByIndex(index - 1).point.getX()) {
        throw new InappropriateFunctionPointException("Координата точки X должна быть больше, чем координата предыдущей точки X");
    }
    if (index < pointsCount - 1 && point.getX() >= getNodeByIndex(index + 1).point.getX()) {
```

```
    if (index < pointsCount - 1 && point.getX() >= getNodeByIndex(index + 1).point.getX()) {
        throw new InappropriateFunctionPointException("Координата точки X должна быть меньше координаты X следующей точки");
    }

    node.point = new FunctionPoint(point);
}

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

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


    if (index > 0 && x <= getNodeByIndex(index - 1).point.getX()) {
        throw new InappropriateFunctionPointException("Координата точки X должна быть больше, чем координата предыдущей точки X");
    }
    if (index < pointsCount - 1 && x >= getNodeByIndex(index + 1).point.getX()) {
        throw new InappropriateFunctionPointException("Координата точки X должна быть меньше координаты X следующей точки");
    }

    node.point.setX(x);
}

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

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

@Override
public void deletePoint(int index) {
    if (pointsCount <= 2) throw new IllegalStateException("Точек меньше или равно 2");
    deleteNodeByIndex(index);
}

@Override
public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {

    int insertIndex = 0;
    FunctionNode current = head.next;

    while (current != head && point.getX() > current.point.getX()) {
        current = current.next;
        insertIndex++;
    }


    if (current != head && Math.abs(point.getX() - current.point.getX()) < EPSILON) {
        throw new InappropriateFunctionPointException("Такая точка с координатой X уже существует");
    }


    FunctionNode newNode = addNodeByIndex(insertIndex);
    newNode.point = new FunctionPoint(point);
}
```

Задание 6:

Переименуем класс TabulatedFunction в класс ArrayTabulatedFunction.

Создадим интерфейс TabulatedFunction, содержащий объявления общих методов классов ArrayTabulatedFunction и LinkedListTabulatedFunction.

```java
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction {
    private static class FunctionNode {
        private FunctionPoint point;
        private FunctionNode prev;
        private FunctionNode next;

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

    private FunctionNode head;
    private int pointsCount;
    private static final double EPSILON = 1e-9;

    private void initializeList() {
```

```java
package functions;


public class ArrayTabulatedFunction implements TabulatedFunction {
    private FunctionPoint[] points; //массив точек
    private static final double EPSILON = 1e-10;

    // Конструктор 1:
    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше пр
```

```java
package functions;

public interface TabulatedFunction {

    // Область определения
    double getLeftDomainBorder();
    double getRightDomainBorder();

    // Вычисление значения функции
    double getFunctionValue(double x);

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

    // Строковое представление
    String toString();
}
```

Задание 7:
Создадим класс main

```java
import functions.*;

public class main {
    public static void main(String[] args) {
        System.out.println("=== Тестирование ArrayTabulatedFunction ===");
        testArrayTabulatedFunction();

        System.out.println("\n=== Тестирование LinkedListTabulatedFunction ===");
        testLinkedListTabulatedFunction();

        System.out.println("\n=== Тестирование исключений ===");
        testExceptions();
    }

    private static void testArrayTabulatedFunction() {
        try {
            // Создадим функцию параболы y = x^2 через массив значений Y
            double[] values = { 0.0, 1.0, 4.0, 9.0, 16.0 }; // y = x^2 для x = 0, 1, 2, 3, 4
            TabulatedFunction arrayFunc = new ArrayTabulatedFunction(0.0, 4.0, values);

            System.out.println("ArrayTabulatedFunction (парабола y = x^2):");
            System.out.println("Область определения: [" + arrayFunc.getLeftDomainBorder() + ", "
                    + arrayFunc.getRightDomainBorder() + "]");
            System.out.println("Количество точек: " + arrayFunc.getPointsCount());

            // Выведем все точки
            for (int i = 0; i < arrayFunc.getPointsCount(); i++) {
                FunctionPoint point = arrayFunc.getPoint(i);
                System.out.println("Точка " + i + ": (" + point.getX() + ", " + point.getY() + ")");
            }

            // Протестируем вычисленные значения
            System.out.println("\nВычисление значений функции (y = x^2):");
            for (double x = 0.0; x <= 4.0; x += 0.5) {
                double y = arrayFunc.getFunctionValue(x);
                System.out.println("f(" + x + ") = " + y);
            }

            // Протестируем за пределами области определения
            System.out.println("f(-1.0) = " + arrayFunc.getFunctionValue(-1.0));
            System.out.println("f(5.0) = " + arrayFunc.getFunctionValue(5.0));

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

    private static void testLinkedListTabulatedFunction() {
        try {
            // Создадим функцию параболы y = x^2 через количество точек
            TabulatedFunction linkedListFunc = new LinkedListTabulatedFunction(0.0, 2.0, 5);

            System.out.println("LinkedListTabulatedFunction (парабола y = x^2):");
            System.out.println("Область определения: [" + linkedListFunc.getLeftDomainBorder() + ", "
                    + linkedListFunc.getRightDomainBorder() + "]");
            System.out.println("Количество точек: " + linkedListFunc.getPointsCount());

            // Установим значения Y для параболы y = x^2
            for (int i = 0; i < linkedListFunc.getPointsCount(); i++) {
                double x = linkedListFunc.getPoint(i).getX();
                linkedListFunc.setPointY(i, x * x); // y = x^2
            }

            // Выведем все точки
            for (int i = 0; i < linkedListFunc.getPointsCount(); i++) {
```

```
            // Выведем все точки
            for (int i = 0; i < linkedListFunc.getPointsCount(); i++) {
                FunctionPoint point = linkedListFunc.getPoint(i);
                System.out.println("Точка " + i + ": (" + point.getX() + ", " + point.getY() + ")");
            }


            System.out.println("\nВычисление значений функции (y = x^2):");
            for (double x = 0.0; x <= 2.0; x += 0.25) {
                double y = linkedListFunc.getFunctionValue(x);
                System.out.println("f(" + x + ") = " + y);
            }

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

    private static void testExceptions() {
        System.out.println("Тестирование исключительных ситуаций:");

        // Протестируем некорректные параметры конструктора
        try {
            TabulatedFunction func1 = new ArrayTabulatedFunction(5.0, 1.0, 3);
            System.out.println("ОШИБКА: Должно было быть выброшено исключение для leftX >= rightX");
        } catch (IllegalArgumentException e) {
            System.out.println("Поймали исключение: " + e.getMessage());
        }

        try {
            TabulatedFunction func2 = new LinkedListTabulatedFunction(0.0, 1.0, 1);
            System.out.println("ОШИБКА: Должно было быть выброшено исключение для pointsCount < 2");
        } catch (IllegalArgumentException e) {
            System.out.println("Поймали исключение: " + e.getMessage());
        }

        // Тестирование работы с точками
        try {
            double[] values = { 0.0, 1.0, 4.0 }; // y = x^2 для x = 0, 1, 2
            TabulatedFunction func = new ArrayTabulatedFunction(0.0, 2.0, values);

            // Попытка получить точку с неверным индексом
            try {
                func.getPoint(-1);
                System.out.println("ОШИБКА: Должно было быть выброшено исключение для отрицательного индекса");
            } catch (FunctionPointIndexOutOfBoundsException e) {
                System.out.println("Поймали исключение: " + e.getMessage());
            }

            try {
                func.getPoint(10);
                System.out.println("ОШИБКА: Должно было быть выброшено исключение для слишком большого индекса");
            } catch (FunctionPointIndexOutOfBoundsException e) {
                System.out.println("Поймали исключение: " + e.getMessage());
            }

            // Установим некорректную координату X
            try {
                func.setPointX(1, 0.0); // Должно быть между предыдущей и следующей точкой
                System.out.println("ОШИБКА: Должно было быть выброшено исключение для некорректной координаты X");
            } catch (InappropriateFunctionPointException e) {
                System.out.println("Поймали исключение: " + e.getMessage());
            }

            // Попытаемся добавить точку с существующей координатой X
```

```
            } catch (InappropriateFunctionPointException e) {
                System.out.println("Поймали исключение: " + e.getMessage());
            }

            // Попытаемся добавить точку с существующей координатой X
            try {
                func.addPoint(new FunctionPoint(1.0, 5.0));
                System.out.println("ОШИБКА: Должно было быть выброшено исключение для дублирующейся координаты X");
            } catch (InappropriateFunctionPointException e) {
                System.out.println("Поймали исключение: " + e.getMessage());
            }

            // Удалим точку если останется меньше 2
            try {
                func.deletePoint(0);
                func.deletePoint(0);
                func.deletePoint(0); // Должно вызвать исключение
                System.out.println("ОШИБКА: Должно было быть выброшено исключение при удалении последней точки");
            } catch (IllegalStateException e) {
                System.out.println("Поймали исключение: " + e.getMessage());
            }

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

        // Тестируем добавление и удаление точек
        try {
            TabulatedFunction func = new LinkedListTabulatedFunction(0.0, 2.0, 3);

            // Установка значений для параболы y = x^2
            for (int i = 0; i < func.getPointsCount(); i++) {
                double x = func.getPoint(i).getX();
                func.setPointY(i, x * x);
            }
            System.out.println("\nИсходные точки:");
            for (int i = 0; i < func.getPointsCount(); i++) {
                FunctionPoint point = func.getPoint(i);
                System.out.println("Точка " + i + ": (" + point.getX() + ", " + point.getY() + ")");
            }
            System.out.println("\nТестирование добавления точек:");
            func.addPoint(new FunctionPoint(0.5, 0.25));
            func.addPoint(new FunctionPoint(1.5, 2.25));

            System.out.println("Количество точек после добавления: " + func.getPointsCount());
            for (int i = 0; i < func.getPointsCount(); i++) {
                FunctionPoint point = func.getPoint(i);
                System.out.println("Точка " + i + ": (" + point.getX() + ", " + point.getY() + ")");
            }

            System.out.println("\nТестирование удаления точек:");
            func.deletePoint(2); // Удаление средней точки
            System.out.println("Количество точек после удаления: " + func.getPointsCount());

        } catch (Exception e) {
            System.out.println("Ошибка при тестировании добавления/удаления: " + e.getMessage());
        }
    }
}
```

Вывод консоли:

```
=== Тестирование ArrayTabulatedFunction ===
ArrayTabulatedFunction (парабола y = x^2):
Область определения: [0.0, 4.0]
Количество точек: 5
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)

Вычисление значений функции (y = x^2):
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(-1.0) = NaN
f(5.0) = NaN

=== Тестирование LinkedListTabulatedFunction ===
LinkedListTabulatedFunction (парабола y = x^2):
Область определения: [0.0, 2.0]
Количество точек: 5
Точка 0: (0.0, 0.0)
Точка 1: (0.5, 0.25)
Точка 2: (1.0, 1.0)
Точка 3: (1.5, 2.25)
```

```
Точка 3: (1.5, 2.25)
Точка 4: (2.0, 4.0)

Вычисление значений функции (y = x^2):
f(0.0) = 0.0
f(0.25) = 0.125
f(0.5) = 0.25
f(0.75) = 0.625
f(1.0) = 1.0
f(1.25) = 1.625
f(1.5) = 2.25
f(1.75) = 3.125
f(2.0) = 4.0

=== Тестирование исключений ===
Тестирование исключительных ситуаций:
Поймали исключение: Левая граница должна быть меньше правой
Поймали исключение: Кол-во точек меньше 2
Поймали исключение: Некорректный индекс  : -1
Поймали исключение: Некорректный индекс  : 10
Поймали исключение: Координата точки X должна быть больше, чем координата предыдущей точки X
Поймали исключение: Такая точка с координатой X уже существует
Поймали исключение: Невозможно удалить точку, требуется минимум 2

Исходные точки:
Точка 0: (0.0, 0.0)
Точка 1: (1.0, 1.0)
Точка 2: (2.0, 4.0)

Тестирование добавления точек:
Количество точек после добавления: 5
```

```
Исходные точки:
Точка 0: (0.0, 0.0)
Точка 1: (1.0, 1.0)
Точка 2: (2.0, 4.0)

Тестирование добавления точек:
Количество точек после добавления: 5
Точка 0: (0.0, 0.0)
Точка 1: (0.5, 0.25)
Точка 2: (1.0, 1.0)
Точка 3: (1.5, 2.25)
Точка 4: (2.0, 4.0)

Тестирование удаления точек:
Количество точек после удаления: 4
```

Вывод: Я дополнил пакет для работы с функциями одной переменной, заданными в табличной форме, добавив классы исключений, новый класс функций и базовый интерфейс.