ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА №3

Студент Синцов М.Ю
Группа 6201-120303D
Руководитель Борисов Д. С.

САМАРА 2025

Цель работы

Дополнение пакета для работы с табулированными функциями путем добавления обработки исключений, реализации на основе связного списка и создания общего интерфейса для различных реализаций.

Задание 1: Изучение классов исключений
Изучены следующие классы исключений:

- Exception - базовый класс для всех проверяемых исключений
- IndexOutOfBoundsException - выход за границы коллекции
- ArrayIndexOutOfBoundsException - специфичный для массивов
- IllegalArgumentException - неверный аргумент метода
- IllegalStateException - недопустимое состояние объекта

Особенности:

- Exception и его потомки - проверяемые исключения (checked)
- RuntimeException и его потомки - непроверяемые (unchecked)
- Для сравнения вещественных чисел используется машинный эпсилон (ε = 1e-10)

Задание 2: Создание собственных исключений
Созданы два класса исключений:

FunctionPointIndexOutOfBoundsException
Наследуется от IndexOutOfBoundsException

Используется при выходе за границы набора точек

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends
IndexOutOfBoundsException {
    public FunctionPointIndexOutOfBoundsException(String a){
        super(a);
    }
    public FunctionPointIndexOutOfBoundsException(){
        super();
```

```
    }
}
```

InappropriateFunctionPointException
Наследуется от Exception

Используется при некорректных операциях с точками

```
package functions;

public class InappropriateFunctionPointException extends Exception{
    public InappropriateFunctionPointException(String a){
        super(a);
    }
    public InappropriateFunctionPointException(){
        super();
    }
}
```

Задание 3: Обработка исключений в TabulatedFunction
Внесены изменения в класс (переименованный в ArrayTabulatedFunction):

Конструкторы:

Бросают IllegalArgumentException при:

- leftX >= rightX
- pointsCount < 2
- values.length < 2

Методы работы с точками:

getPoint(), setPoint(), deletePoint() и др.:

Бросают FunctionPointIndexOutOfBoundsException при неверном индексе

- setPoint() и setPointX():

Бросают InappropriateFunctionPointException при нарушении порядка точек

- addPoint():

Бросает InappropriateFunctionPointException при совпадающих абсциссах

- deletePoint():

Бросает IllegalStateException при попытке удаления при pointsCount <= 2

Задание 4: Реализация LinkedListTabulatedFunction
Создан класс с двусвязным циклическим списком:

Структура:

- Внутренний класс FunctionNode для элементов списка
- Голова списка не содержит данных
- Циклические связи (последний → голова → первый)
- Методы работы со списком:
- getNodeByIndex(int index) - получение узла с оптимизацией доступа
- addNodeToTail() - добавление в конец
- addNodeByIndex(int index) - вставка по индексу
- deleteNodeByIndex(int index) - удаление по индексу

Оптимизация доступа:

- Хранится ссылка на последний доступный узел
- Поиск начинается с ближайшего конца

Задание 5: Методы LinkedListTabulatedFunction
Реализованы методы интерфейса TabulatedFunction:

Конструкторы:

- Аналогичны конструкторам ArrayTabulatedFunction
- Используют методы работы со списком

Методы:

- getFunctionValue() - линейная интерполяция

- addPoint(), setPoint() - с проверкой порядка
- deletePoint() - с проверкой минимального количества

Особенности:

- Использование машинного эпсилона для сравнения вещественных чисел
- Все методы бросают те же исключения, что и ArrayTabulatedFunction

Задание 6: Создание интерфейса TabulatedFunction
Создан интерфейс с общими методами:

```
package functions;

public interface TabulatedFunction {
    public double getLeftDomainBorder();
    public double getRightDomainBorder();
    public int getPointsCount();
    public double getFunctionValue(double x);
    public void showPoints();
    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;
}
```

Изменения в классах:

- ArrayTabulatedFunction переименован и реализует интерфейс
- LinkedListTabulatedFunction реализует интерфейс
- Обе реализации имеют одинаковую сигнатуру методов

Задание 7: Тестирование программы
Создан тестовый класс Main:

Тестирование функциональности:

- Создание функций через интерфейс
- Проверка вычисления значений
- Тестирование добавления/удаления точек

Тестирование исключений:

- Некорректные параметры конструктора
- Выход за границы массива
- Некорректное изменение точки
- Добавление существующей точки
- Удаление слишком многих точек

Результаты работы
Код класса ArrayTabulatedFunction:

```
package functions;
public class ArrayTabulatedFunction implements TabulatedFunction {
    private FunctionPoint[] points;
    final double EPSILON = 2.220446049250326E-16;
    private int pointsCount;
    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount){
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть
не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть
меньше правой");
        }
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i<pointsCount;i++){
            double x = leftX + i*step;
            points[i] = new FunctionPoint(x, 0);
        }
        this.pointsCount = pointsCount;
```

```
    }
    public ArrayTabulatedFunction(double leftX, double rightX, double[] values){
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
        pointsCount = values.length;
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i< pointsCount;i++){
            double x = leftX + i*step;
            points[i] = new FunctionPoint(x, values[i]);
        }

    }
    public double getLeftDomainBorder(){
        return points[0].getX();
    }
    public double getRightDomainBorder(){
        return points[pointsCount-1].getX();
    }
    public int getPointsCount() {
        return pointsCount;
    }

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


        for (int i = 0; i < pointsCount; i++) {
            if (Math.abs(points[i].getX() - x) <= EPSILON) {
                return points[i].getY();
            }
        }


        for (int i = 0; i < pointsCount - 1; i++) {
            if (x >= points[i].getX() && x <= points[i + 1].getX()) {
                double leftX = points[i].getX();
```

```
                double rightX = points[i + 1].getX();
                double leftY = points[i].getY();
                double rightY = points[i + 1].getY();


                return leftY + (rightY - leftY) * (x - leftX) / (rightX - leftX);
            }
        }


        return Double.NaN;
    }
    public void showPoints(){
        for (int i = 0; i<pointsCount;i++){
            System.out.print("Значение точки "+ (i+1) + ": ");
            points[i].showPoint();
            System.out.println();
        }
    }
    public FunctionPoint getPoint(int index){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
за границы массива точек");
        }
        return new FunctionPoint(points[index]);
    }
    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException{
        {
            if (index < 0 || index >= pointsCount) {
                throw new FunctionPointIndexOutOfBoundsException("Индекс
выходит за границы массива точек");
            }
            if ((point.getX() <= points[index - 1].getX() && index != 0) || (point.getX()
>= points[index + 1].getX() && index != pointsCount - 1)) {
                throw new InappropriateFunctionPointException("Координата x
задаваемой точки лежит вне интервала, определяемого значениями соседних
точек табулированной функции");
            }
        }
        points[index] = point;
    }
    public double getPointX(int index){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
```

```
за границы массива точек");
        }
        return points[index].getX();
    }
    public void setPointX(int index, double x) throws
InappropriateFunctionPointException{
        {

            if (index < 0 || index >= pointsCount) {
                throw new FunctionPointIndexOutOfBoundsException("Индекс
выходит за границы массива точек");
            }
            if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
                throw new InappropriateFunctionPointException("Координата x
задаваемой точки лежит вне интервала");
            }
            if (x >= points[index + 1].getX() || x <= points[index - 1].getX()) {
                throw new IllegalArgumentException("Новое значение x нарушает
упорядоченность");
            }
        }
        points[index].setX(x);
    }
    public double getPointY(int index){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
за границы массива точек");
        }
        return points[index].getY();
    }
    public void setPointY(int index, double y){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
за границы массива точек");
        }
        points[index].setY(y);
    }
    public void deletePoint(int index){

            if (index < 0 || index >= pointsCount) {
                throw new FunctionPointIndexOutOfBoundsException("Индекс
выходит за границы массива точек");
            }
            if (pointsCount <= 2) {
                throw new IllegalStateException("Нельзя удалить точку: функция
```

```
должна содержать минимум 2 точки");
            }

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


    }
    public void addPoint(FunctionPoint point)throws
InappropriateFunctionPointException {
        {
            int insertIndex = 0;
            while (insertIndex < pointsCount && point.getX() >
points[insertIndex].getX()) {
                insertIndex++;
            }
            if (insertIndex < pointsCount && (Math.abs(point.getX() -
points[insertIndex].getX()) <= EPSILON)) {
                throw new InappropriateFunctionPointException("Точка с x=" +
point.getX() + " уже существует");
            }
        }
        int insertIndex = 0;
        while (insertIndex < pointsCount && point.getX() >
points[insertIndex].getX()) {
            insertIndex++;
        }
        if (pointsCount == points.length) {
            int newCapacity = points.length + points.length / 2 + 1;
            FunctionPoint[] newPoints = new FunctionPoint[newCapacity];
            System.arraycopy(points, 0, newPoints, 0, pointsCount);
            points = newPoints;
        }

        if (insertIndex < pointsCount) {
            System.arraycopy(points, insertIndex, points, insertIndex + 1, pointsCount
- insertIndex);
        }
        points[insertIndex] = new FunctionPoint(point);
        pointsCount++;
    }
}
```

Код класса LinkedListTabulatedFunction

```
package functions;

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

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

    private FunctionNode head;
    private int pointsCount;
    private FunctionNode lastAccessedNode;
    private int lastAccessedIndex;
    private static final double EPSILON = 2.220446049250326E-16;

    public LinkedListTabulatedFunction(double leftX, double rightX, int
pointsCount) {
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть
не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть
меньше правой");
        }
        initializeList();
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            addNodeToTail().point = new FunctionPoint(x, 0.0);
        }

    }

    public LinkedListTabulatedFunction(double leftX, double rightX, double[]
values) {
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек должно быть
не менее 2");
        }
        if (leftX >= rightX) {
```

```
            throw new IllegalArgumentException("Левая граница должна быть
меньше правой");
        }
        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]);
        }
    }

    private void initializeList() {
        head = new FunctionNode(null);
        head.prev = head;
        head.next = head;
        pointsCount = 0;
        lastAccessedNode = head;
        lastAccessedIndex = -1;
    }

    private FunctionNode getNodeByIndex(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс " + index
+ " " +
                    "выходит за границы [0, " + (pointsCount - 1) + "]");
        }
        FunctionNode node;
        if (lastAccessedIndex != -1 && Math.abs(index - lastAccessedIndex) < index
&&
                Math.abs(index - lastAccessedIndex) < pointsCount - index) {
            node = lastAccessedNode;
            if (index > lastAccessedIndex) {
                for (int i = lastAccessedIndex; i < index; i++) {
                    node = node.next;
                }
            } else {
                for (int i = lastAccessedIndex; i > index; i--) {
                    node = node.prev;
                }
            }
        } else {
            // Начинаем с головы
            node = head.next;
            for (int i = 0; i < index; i++) {
                node = node.next;
```

```
            }
        }

        lastAccessedNode = node;
        lastAccessedIndex = index;
        return node;
    }

    private FunctionNode addNodeToTail() {
        return addNodeByIndex(pointsCount);
    }

    private FunctionNode addNodeByIndex(int index) {
        if (index < 0 || index > pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс " + index
+ " " +
                    "выходит за границы [0, " + (pointsCount - 1) + "]");
        }
        FunctionNode newNode = new FunctionNode(null);
        FunctionNode targetNode;
        if (index == pointsCount) {
            targetNode = head;
        } else {
            targetNode = getNodeByIndex(index);
        }



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

        pointsCount++;
        lastAccessedIndex = -1; // Сбрасываем кэш

        return newNode;
    }

    private FunctionNode deleteNodeByIndex(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс " + index
+ " " +
                    "выходит за границы [0, " + (pointsCount - 1) + "]");
        }
        FunctionNode node = getNodeByIndex(index);
```

```
        node.next.prev = node.prev;
        node.prev.next = node.next;

        pointsCount--;
        return node;
    }

    public double getLeftDomainBorder() {
        return getNodeByIndex(0).point.getX();
    }

    public double getRightDomainBorder() {
        return getNodeByIndex(pointsCount - 1).point.getX();
    }

    public int getPointsCount(){
        return pointsCount;
    }

    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        for (int i = 0; i < pointsCount; i++){
            if (Math.abs(getNodeByIndex(i).point.getX() - x) <= EPSILON) {
                return getNodeByIndex(i).point.getY();
            }
        }
        for (int i = 0; i < pointsCount - 1; i++) {
            if (x >= getNodeByIndex(i).point.getX() && x <=
getNodeByIndex(i+1).point.getX()) {
                double leftX = getNodeByIndex(i).point.getX();
                double rightX = getNodeByIndex(i+1).point.getX();
                double leftY = getNodeByIndex(i).point.getY();
                double rightY = getNodeByIndex(i+1).point.getY();


                return leftY + (rightY - leftY) * (x - leftX) / (rightX - leftX);
            }
        }


        return Double.NaN;
    }
```

```
    public void showPoints(){
        for (int i = 0; i<pointsCount;i++){
            System.out.print("Значение точки "+ (i+1) + ": ");
            getNodeByIndex(i).point.showPoint();
            System.out.println();
        }
    }
    public FunctionPoint getPoint(int index){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
за границы массива точек");
        }
        return new FunctionPoint(getNodeByIndex(index).point);
    }
    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException{
        {
            if (index < 0 || index >= pointsCount) {
                throw new FunctionPointIndexOutOfBoundsException("Индекс
выходит за границы массива точек");
            }
            if ((point.getX() <= getNodeByIndex(index-1).point.getX() && index != 0)
|| (point.getX() >= getNodeByIndex(index+1).point.getX() && index !=
pointsCount - 1)) {
                throw new InappropriateFunctionPointException("Координата x
задаваемой точки лежит вне интервала, определяемого значениями соседних
точек табулированной функции");
            }
        }
        getNodeByIndex(index).point = point;
    }
    public double getPointX(int index){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
за границы массива точек");
        }
        return getNodeByIndex(index).point.getX();
    }
    public void setPointX(int index, double x) throws
InappropriateFunctionPointException{
        {

            if (index < 0 || index >= pointsCount) {
                throw new FunctionPointIndexOutOfBoundsException("Индекс
выходит за границы массива точек");
```

```
            }
            if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
                throw new InappropriateFunctionPointException("Координата x
задаваемой точки лежит вне интервала");
            }
            if (x >= getNodeByIndex(index+1).point.getX() || x <=
getNodeByIndex(index+1).point.getX()) {
                throw new IllegalArgumentException("Новое значение x нарушает
упорядоченность");
            }
        }
        getNodeByIndex(index).point.setX(x);
    }
    public double getPointY(int index){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
за границы массива точек");
        }
        return getNodeByIndex(index).point.getY();
    }
    public void setPointY(int index, double y){
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит
за границы массива точек");
        }
        getNodeByIndex(index).point.setY(y);
    }
    public void deletePoint(int index){

            if (index < 0 || index >= pointsCount) {
                throw new FunctionPointIndexOutOfBoundsException("Индекс
выходит за границы массива точек");
            }
            if (pointsCount <= 2) {
                throw new IllegalStateException("Нельзя удалить точку: функция
должна содержать минимум 2 точки");
            }

        deleteNodeByIndex(index).point = null;



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

```
            int insertIndex = 0;
            while (insertIndex < pointsCount && point.getX() >
getNodeByIndex(insertIndex).point.getX()) {
                insertIndex++;
            }
            if (insertIndex < pointsCount && (Math.abs(point.getX() -
getNodeByIndex(insertIndex).point.getX()) <= EPSILON)) {
                throw new InappropriateFunctionPointException("Точка с x=" +
point.getX() + " уже существует");
            }
        }
        int insertIndex = 0;
        while (insertIndex < pointsCount && point.getX() >
getNodeByIndex(insertIndex).point.getX()) {
            insertIndex++;
        }
        addNodeByIndex(insertIndex).point = new FunctionPoint(point);
    }
}
```

Код класса Main

```
import functions.*;
public class Main {
    public static void main(String[] args) {
        try {
            // Тестирование ArrayTabulatedFunction
            System.out.println("=== Тестирование ArrayTabulatedFunction ===");
            testTabulatedFunction(new ArrayTabulatedFunction(0, 5, 3));

            // Тестирование LinkedListTabulatedFunction
            System.out.println("\n=== Тестирование LinkedListTabulatedFunction
===");
            testTabulatedFunction(new LinkedListTabulatedFunction(0, 5, 3));

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

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

    private static void testTabulatedFunction(TabulatedFunction function) {
```

```
        System.out.println("Функция: ");
        function.showPoints();
        System.out.println("Левая граница: " + function.getLeftDomainBorder());
        System.out.println("Правая граница: " + function.getRightDomainBorder());
        System.out.println("Количество точек: " + function.getPointsCount());
        System.out.println("f(2.5) = " + function.getFunctionValue(2.5));

        try {
            // Добавление точки
            function.addPoint(new FunctionPoint(2.5, 6.25));
            System.out.println("После добавления точки (2.5; 6.25): " + function);

            // Изменение точки
            function.setPointY(2, 10.0);
            System.out.println("После изменения Y точки 2: " + function);

        } catch (InappropriateFunctionPointException e) {
            System.out.println("Исключение при работе с точкой: " +
e.getMessage());
        }
    }

    private static void testExceptions() {
        try {
            // Некорректный конструктор
            TabulatedFunction func1 = new ArrayTabulatedFunction(5, 0, 3);
        } catch (IllegalArgumentException e) {
            System.out.println("Поймано IllegalArgumentException: " +
e.getMessage());
        }

        try {
            TabulatedFunction func = new ArrayTabulatedFunction(0, 5, 3);
            // Выход за границы
            func.getPoint(10);
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Поймано FunctionPointIndexOutOfBoundsException:
" + e.getMessage());
        }

        try {
            TabulatedFunction func = new ArrayTabulatedFunction(0, 5, 3);
            // Нарушение упорядоченности
            func.setPoint(1, new FunctionPoint(0, 0));
        } catch (InappropriateFunctionPointException e) {
```

```
            System.out.println("Поймано InappropriateFunctionPointException: " +
e.getMessage());
        }

        try {
            TabulatedFunction func = new ArrayTabulatedFunction(0, 5, 3);
            // Удаление при недостаточном количестве точек
            func.deletePoint(0);
            func.deletePoint(0);
        } catch (IllegalStateException e) {
            System.out.println("Поймано IllegalStateException: " + e.getMessage());
        }
    }
}
```

Результат выполнения:
=== Тестирование ArrayTabulatedFunction ===
Функция:
Значение точки 1: [0.0,0.0]

Значение точки 2: [2.5,0.0]

Значение точки 3: [5.0,0.0]

Левая граница: 0.0
Правая граница: 5.0
Количество точек: 3
f(2.5) = 0.0
Исключение при работе с точкой: Точка с x=2.5 уже существует

=== Тестирование LinkedListTabulatedFunction ===
Функция:
Значение точки 1: [0.0,0.0]

Значение точки 2: [2.5,0.0]

Значение точки 3: [5.0,0.0]

Левая граница: 0.0
Правая граница: 5.0
Количество точек: 3
f(2.5) = 0.0
Исключение при работе с точкой: Точка с x=2.5 уже существует

=== Тестирование исключений ===

Поймано IllegalArgumentException: Левая граница должна быть меньше правой
Поймано FunctionPointIndexOutOfBoundsException: Индекс выходит за границы массива точек
Поймано InappropriateFunctionPointException: Координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции
Поймано IllegalStateException: Нельзя удалить точку: функция должна содержать минимум 2 точки

Выводы
Достигнутые результаты:

Обработка исключений:

- Созданы специализированные классы исключений
- Реализована проверка входных данных
- Обеспечена корректность работы при ошибках

Реализация на связном списке:

- Создан двусвязный циклический список
- Реализована оптимизация доступа к элементам
- Обеспечена инкапсуляция внутренней структуры

Интерфейс и полиморфизм:

- Создан общий интерфейс TabulatedFunction
- Реализовано две различные реализации
- Обеспечена возможность легкой замены реализации

Принципы ООП:

- Инкапсуляция: скрытие внутренней реализации
- Наследование: иерархия исключений

- Полиморфизм: работа через интерфейс
- Абстракция: общий интерфейс для различных реализаций

Качество кода:

- Использование машинного эпсилона для сравнения вещественных чисел
- Проверка граничных случаев
- Осмысленные сообщения об ошибках

Преимущества подхода:

- Единый интерфейс для работы с функциями
- Возможность выбора реализации в зависимости от задачи
- Защита от некорректного использования
- Легкость расширения системы

Программа демонстрирует правильное использование исключений, реализацию сложных структур данных и применение принципов объектно-ориентированного программирования в Java.