Лабораторная работа №3. ООП.

Выполнил лабораторную работу:
Гордеев Максим Дмитриевич
Группа: 6204-010302D

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

Ход выполнения лабораторной работы

Задание 1:

Ознакомимся со следующими классами исключений:

java.lang.Exception - базовый класс для всех проверяемых исключений:

- Является суперклассом для всех исключений, которые должны обрабатываться в коде
- Проверяемые исключения должны быть объявлены в сигнатуре метода или обработаны в блоке try-catch

java.lang.IndexOutOfBoundsException - исключение выхода за границы:

- Выбрасывается при попытке доступа к индексу за пределами допустимого диапазона
- Используется для коллекций и структур данных
- Наследуется от RuntimeException (непроверяемое исключение)

java.lang.ArrayIndexOutOfBoundsException - специализированное исключение для массивов:

- Наследуется от IndexOutOfBoundsException
- Выбрасывается при обращении к несуществующему индексу массива

java.lang.IllegalArgumentException - исключение неверного аргумента:

- Выбрасывается, когда метод получает неподходящий или некорректный аргумент
- Часто используется для проверки входных параметров методов

java.lang.IllegalStateException - исключение недопустимого состояния:

- Выбрасывается, когда метод вызывается в неподходящий момент или объект находится в некорректном состоянии
- Используется для контроля жизненного цикла объектов

Задание 2:

Создадим свои классы исключений **FunctionPointIndexOutOfBoundsException** и **InappropriateFunctionPointException.**

```java
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException {

    public FunctionPointIndexOutOfBoundsException(){
        super();
    }

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

    }

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

```

```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);
    }

}

```

Класс исключений **FunctionPointIndexOutOfBoundsException** наследуется от **IndexOutOfBoundsException** и является непроверяемым исключением, мы используем его в качестве исключение выхода за границы набора точек при обращении к ним по номеру.

Класс исключений **InappropriateFunctionPointException** наследуется напрямую от класса **Exception**, поэтому является проверяемым исключением, мы используем его в качестве исключения, выбрасываемого при попытке добавления или изменения точки функции несоответствующим образом.

Задание 3:

Добавим изменения в код, обеспечивающие выбрасывание исключений методами класса при несоблюдении необходимых условий:

```
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException{ // Конструктор

public ArrayTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException{ // Конструктор
```

Конструкторы выбрасывают исключение **IllegalArgumentException** в случае попытки создать **ArrayTabulatedFunction**, в котором левая граница больше или равна правой или количество точек меньше двух.

```
82      public FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException{
83
84          if (index < 0 || index >=size){ // Если не подходит под условие выбрасываем исключение
85              throw new FunctionPointIndexOutOfBoundsException(message: "Выход за пределы массива");
86          }
87
88          FunctionPoint pointCopy = new FunctionPoint(points[index]); // Создаём копию, чтобы не нарушать инкапчуляцию
89          return pointCopy;
90      }
```

Метод getPoint(int index) выбрасывает исключение **FunctionPointIndexOutOfBoundsException**, если пытаемся взять элемент под несуществующим индексом.

```
92      public void setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException,
93              InappropriateFunctionPointException{
94          if (point == null){
95              throw new InappropriateFunctionPointException(message: "Попытка изменить точку на null");
96          }
97          if (index < 0 || index >= size) { // Если не подходит под условие выбрасываем исключение
98              throw new FunctionPointIndexOutOfBoundsException(message: "Выход за пределы массива");
99          }
100         if (index > 0 && point.getCoorX() < points[index-1].getCoorX()){
101             throw new InappropriateFunctionPointException(message: "x Нарушает порядок,выберите другой индекс");
102         }
103         if (index < size - 1 && point.getCoorX() > points[index + 1].getCoorX()){
104             throw new InappropriateFunctionPointException(message: "x Нарушает порядок,выберите другой индекс");
105         }
106         points[index].setCoorX(point.getCoorX());
107         points[index].setCoorY(point.getCoorY());
108     }
```

Метод **setpoint(int index, FunctionPoint point)** выбрасывает несколько исключений:

- **InappropriateFunctionPointException,** когда мы пытаемся изменить точку на **null** и когда у передоваемой **point** координата **x** нарушает порядох элементов.
- **FunctionPointIndexOutOfBoundsException** – выход за пределы массива.

Аналогичным образом исключениями дополнены и другие методы класса.

Задание 4:

Создадим класс двусвязанного циклического списка:

```java
package functions;

public class FunctionNode{

    private FunctionPoint point;
    private FunctionNode prev;
    private FunctionNode next;

    public FunctionNode(FunctionPoint point) {
        this.point = (point != null) ? new FunctionPoint(point) : null;
    }

    public void setPrev(FunctionNode prev){ this.prev = prev;}
    public void setNext(FunctionNode next){this.next = next;}
    public void setPoint(FunctionPoint newPoint){
        point.setCoorX(newPoint.getCoorX());
        point.setCoorY(newPoint.getCoorY());
    }

    public FunctionNode getPrev(){ return prev;} // возвращает ссылку!!!
    public FunctionNode getNext(){return next;}
    public FunctionPoint getPoint() throws IllegalStateException{
        if (point == null)
            throw new IllegalStateException(s: "Текущий узел имеет point равный null");
        return point;} // возвращаем ссылку
}
```

```java
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction{

    private int size; // Размер списка

    private FunctionNode head; // Голова списка
    private FunctionNode tail; // Хвост списка

    private double leftDomainBorder, rightDomainBorder; // Левая и правая границы


    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException{…

    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException{…

    // Гет методы
    public double getLeftDomainBorder() {…

    public double getRightDomainBorder() {…


    // Метод для получения прямой по двум точкам
    private double interpolate(double x, double x1, double y1, double x2, double y2) {…

    // Метод для получения значения функции
    public double getFunctionValue(double x) {…
    public int getPointsCount() { // Возвращает количество точек…

    public FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException{…

    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException, FunctionPointIndexOutOfBoundsException, IllegalStateException {…


    public double getPointX(int index) throws FunctionPointIndexOutOfBoundsException,  IllegalStateException{…

    public double getPointY(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException {…

    public void setPointX(int index, double x)…

    public void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException {…

    public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException {…

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



```java
    // Методы списка
    private FunctionNode getNodeByIndex(int index){//Возвращает ссылку на узел // Может вернуть узел с point = null…

    private FunctionNode addNodeToTail(FunctionNode newNode){ // Добавляет узел в конец списка…

    private FunctionNode addNodeByIndex(int index) { // Добавляем элемент по индексу…

    private FunctionNode deleteNodeByIndex(int index) { // Удаления элемента по индексу…

    private int getSize(){…

    }
```

Методы двусвязанного списка инкапсулированы и не выполняют никаких действий с **point**, они лишь реализуют логику взаимодействия с узлами. Методы, которые реализуют функционал табулированной функции используют методы списка.

Задание 5:

Реализуем конструкторы, аналогичные конструкторам класса **ArrayTabulatedFunction**:

```
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException{
        if (leftX >= rightX) {
            throw new IllegalArgumentException(s: "Левая ганица не может быть больше или равной правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException(s: "Количество точек не может быть меньше двух.");
        }

        this.leftDomainBorder = leftX;
        this.rightDomainBorder = rightX;
        this.size = pointsCount;

        head = new FunctionNode(new FunctionPoint(leftX, y: 0));
        FunctionNode currNode = head;

        double delta = (rightX - leftX) / (pointsCount - 1);
        for (int i = 1; i < pointsCount - 1; i++){
                currNode.setNext(new FunctionNode(new FunctionPoint(leftX+delta*i, y: 0)));
                currNode.getNext().setPrev(currNode);
                currNode = currNode.getNext();}
        tail = new FunctionNode(new FunctionPoint(rightX, y: 0));
        currNode.setNext(tail);
        tail.setPrev(currNode);
    }
```

```
public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException{
    if (leftX >= rightX) {
        throw new IllegalArgumentException(s: "Левая ганица не может быть больше или равной правой");
    }
    if (values.length < 2) {
        throw new IllegalArgumentException(s: "Количество точек не может быть меньше двух.");
    }
    this.leftDomainBorder = leftX;
    this.rightDomainBorder = rightX;
    this.size = values.length;

    head = new FunctionNode(new FunctionPoint(leftX, values[0]));
    FunctionNode currNode = head;

    double delta = (rightX - leftX) / (size - 1);
    for (int i = 1; i < size - 1; i++) {
        currNode.setNext(new FunctionNode(new FunctionPoint(leftX + delta * i, values[i])));
        currNode.getNext().setPrev(currNode);
        currNode = currNode.getNext();
    }
    tail = new FunctionNode(new FunctionPoint(rightX, values[size-1]));
    currNode.setNext(tail);
    tail.setPrev(currNode);
}
```

Задание 6:

Создадим общий интерфейс для **ArrayTabulatedFunction** и **LinkedListTabulatedFunction**:

```java
package functions;
public interface TabulatedFunction {


public double getLeftDomainBorder();
public double getRightDomainBorder();

public double getFunctionValue(double x);

public int getPointsCount();

public FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;

public void setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointExcep

public double getPointX(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;

public double getPointY(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;

public void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException,InappropriateFunctionPointException, Illeg

public void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;

public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;

public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}

```

И укажем, что наши классы реализуют этот интерфейс:

```java
public class ArrayTabulatedFunction implements TabulatedFunction{
```

```java
public class LinkedListTabulatedFunction implements TabulatedFunction{
```

Задание 7:

Протестируем наш код:

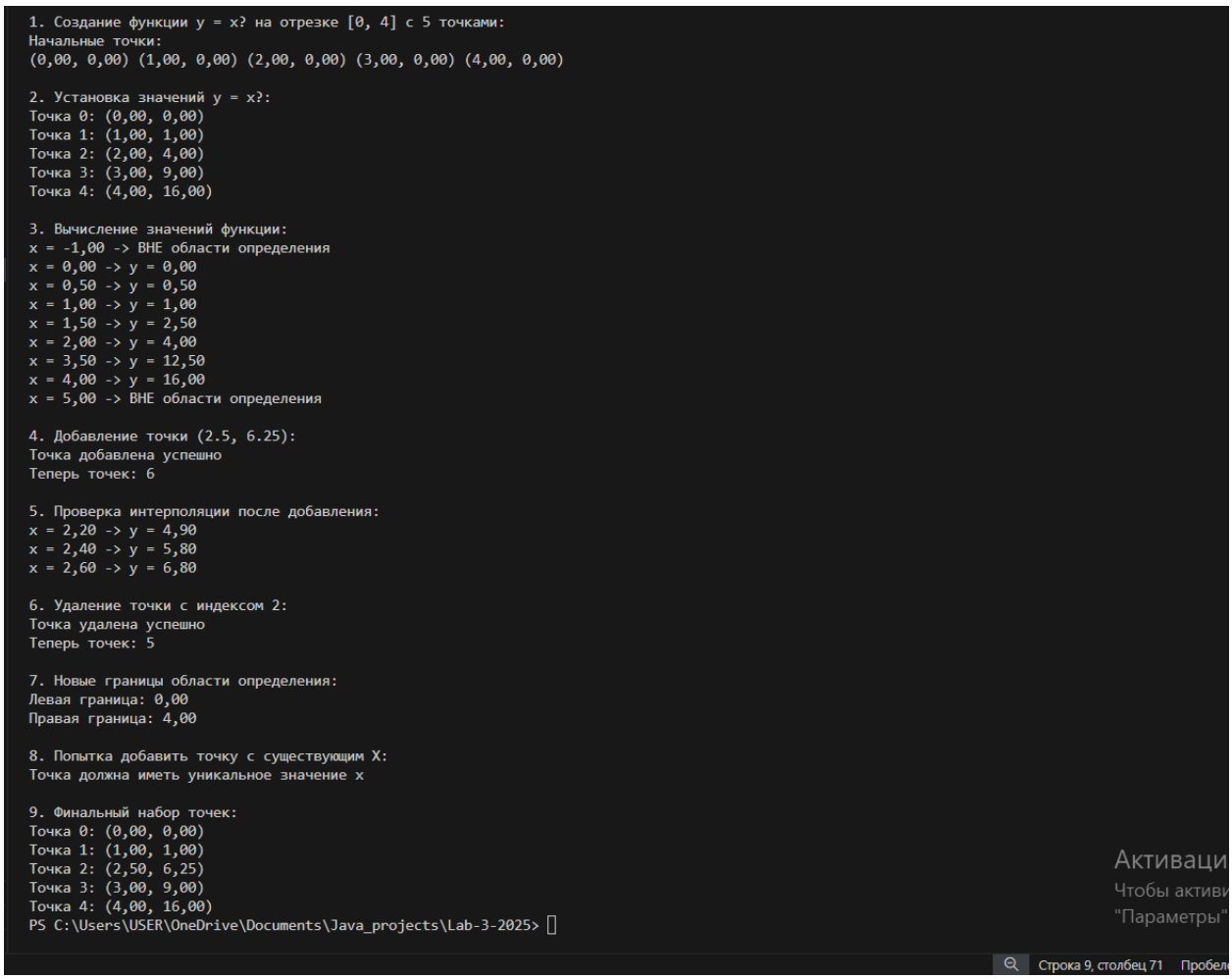
```java
        // Вывод начальных точек
        System.out.println(x: "Начальные точки:");
        for (int i = 0; i < function.getPointsCount(); i++) {
            System.out.printf(format: "(%.2f, %.2f) ", function.getPointX(i), function.getPointY(i
        }
        System.out.println(x: "\n");

        // Установка реальных значений y = x²
        System.out.println(x: "2. Установка значений y = x²:");

        for (int i = 0; i < function.getPointsCount(); i++) {
            double x = function.getPointX(i);
            function.setPointY(i, x * x);
            System.out.printf(format: "Точка %d: (%.2f, %.2f)\n", i, x, x * x);
        }
        System.out.println();

        // Тестирование вычисления значений в разных точках
        System.out.println(x: "3. Вычисление значений функции:");
        double[] testPoints = { -1, 0, 0.5, 1, 1.5, 2, 3.5, 4, 5 };

        for (double x : testPoints) {
            double y = function.getFunctionValue(x);

            if (Double.isNaN(y)) {
                System.out.printf(format: "x = %.2f -> ВНЕ области определения\n", x);
            } else {
                System.out.printf(format: "x = %.2f -> y = %.2f\n", x, y);
            }
        }
        System.out.println();

        // Добавление новой точки
        System.out.println(x: "4. Добавление точки (2.5, 6.25):");
        try {
            function.addPoint(new FunctionPoint(x: 2.5, y: 6.25));
            System.out.println(x: "Точка добавлена успешно");
            System.out.println("Теперь точек: " + function.getPointsCount());
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        } catch (InappropriateFunctionPointException e) {
            System.out.println(e.getMessage());
        }
```

```java
        // 5. Проверка интерполяции после добавления
        System.out.println(x: "5. Проверка интерполяции после добавления:");
        for (double x = 2.2; x <= 2.8; x += 0.2) {
            double y = function.getFunctionValue(x);
            System.out.printf(format: "x = %.2f -> y = %.2f\n", x, y);
        }
        System.out.println();

        // 6. Удаление точки
        System.out.println(x: "6. Удаление точки с индексом 2:");
        try {
            function.deletePoint(index: 2);

            System.out.println(x: "Точка удалена успешно");
            System.out.println("Теперь точек: " + function.getPointsCount());

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

        // 7. Проверка границ после изменений
        System.out.println(x: "7. Новые границы области определения:");
        System.out.printf(format: "Левая граница: %.2f\n", function.getLeftDomainBorder());
        System.out.printf(format: "Правая граница: %.2f\n", function.getRightDomainBorder());
        System.out.println();

        // 8. Попытка добавить точку с существующим X
        System.out.println(x: "8. Попытка добавить точку с существующим X:");

        try {
            function.addPoint(new FunctionPoint(x: 3.0, y: 100)); // Должен быть отказ
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        } catch (InappropriateFunctionPointException e) {
            System.out.println(e.getMessage());
        }

        // 9. Финальный вывод всех точек
        System.out.println(x: "\n9. Финальный набор точек:");

        for (int i = 0; i < function.getPointsCount(); i++) {
            System.out.printf(format: "Точка %d: (%.2f, %.2f)\n", i, function.getPointX(i), function.getPointY(i));

        }

    }
}
```

Сначала посмотрим вывод для списка:

```
TabulatedFunction function = new LinkedListTabulatedFunction(leftX: 0, rightX: 4, pointsCount: 5);
1. Создание функции y = x? на отрезке [0, 4] с 5 точками:
Начальные точки:
(0,00, 0,00) (1,00, 0,00) (2,00, 0,00) (3,00, 0,00) (4,00, 0,00)

2. Установка значений y = x?:
Точка 0: (0,00, 0,00)
Точка 1: (1,00, 1,00)
Точка 2: (2,00, 4,00)
Точка 3: (3,00, 9,00)
Точка 4: (4,00, 16,00)

3. Вычисление значений функции:
x = -1,00 -> ВНЕ области определения
x = 0,00 -> y = 0,00
x = 0,50 -> y = 0,50
x = 1,00 -> y = 1,00
x = 1,50 -> y = 2,50
x = 2,00 -> y = 4,00
x = 3,50 -> y = 12,50
x = 4,00 -> y = 16,00
x = 5,00 -> ВНЕ области определения

4. Добавление точки (2.5, 6.25):
Точка добавлена успешно
Теперь точек: 6

5. Проверка интерполяции после добавления:
x = 2,20 -> y = 4,90
x = 2,40 -> y = 5,80
x = 2,60 -> y = 6,80

6. Удаление точки с индексом 2:
Точка удалена успешно
Теперь точек: 5

7. Новые границы области определения:
Левая граница: 0,00
Правая граница: 4,00

8. Попытка добавить точку с существующим X:
Точка должна иметь уникальное значение x

9. Финальный набор точек:
Точка 0: (0,00, 0,00)
Точка 1: (1,00, 1,00)
Точка 2: (2,50, 6,25)
Точка 3: (3,00, 9,00)
Точка 4: (4,00, 16,00)
PS C:\Users\USER\OneDrive\Documents\Java_projects\Lab-3-2025>
```

Теперь для массива:



Отдельно проверим работу исключений:

```
=== ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ ===

1. ТЕСТИРОВАНИЕ ArrayTabulatedFunction:

Тест 1: Некорректные границы
Поймано ожидаемое исключение: Левая ганица не может быть больше или равной правой

Тест 2: Слишком мало точек
Поймано ожидаемое исключение: Количество точек не может быть меньше двух.

Тест 3: Выход за границы массива
Поймано ожидаемое исключение: Выход за пределы массива

Тест 4: Нарушение порядка X при установке точки
Поймано ожидаемое исключение: x Нарушает порядок,выберите другой индекс

Тест 5: Удаление точки при малом количестве
Поймано ожидаемое исключение: Нельзя удалить точку из массива

Тест 6: Добавление точки с существующим X
Поймано ожидаемое исключение: Точка должна иметь уникальное значение x

Тест 7: Установка null точки
Поймано ожидаемое исключение: Попытка изменить точку на null
```

```
3. ОБЩИЕ ТЕСТЫ ДЛЯ ОБОИХ РЕАЛИЗАЦИЙ:

Тестирование ArrayTabulatedFunction:
getPointX(-1): Выход за пределы массива
getPointY(10): Выход за пределы массива
setPointX с нарушением порядка: x Нарушает порядок,выберите другой индекс
setPointY(100): Выход за пределы массива

Тестирование LinkedListTabulatedFunction:
getPointX(-1): Выход за пределы массива
getPointY(10): Выход за пределы массива
setPointX с нарушением порядка: x Нарушает порядок,выберите другой индекс
setPointY(100): Выход за пределы массива
```

```
4. ТЕСТИРОВАНИЕ FunctionNode:
FunctionNode с null point: Текущий узел имеет point равный null
```