Лабораторная работа №3

Выполнил работу

Гапанюк Антон Андреевич

Группа: 6204-010302D

Самара, 2025 г.

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

Ход работы

Задание 1

Изучим классы исключений:

1. **java.lang.Exception** – базовый класс всех исключений. Родитель всех «проверяемых» исключений.
2. **java.lang.IndexOutOfBoundsException** – выход за границы. Возникает при обращении к несуществующему индексу в массиве, списке и т. д.
3. **java.lang.ArrayIndexOutOfBoundsException** - частный случай. Возникает при обращении к массиву с некорректным индексом
4. **java.lang.IllegalArgumentException** - неверный аргумент. Возникает, когда методу передают некорректный аргумент.
5. **java.lang.IllegalStateException** - неверное состояние. Возникает, когда объект находится в состоянии, не позволяющем выполнить операцию.

Задание 2

Создадим классы исключений

FunctionPointIndexOutOfBoundsException – исключение выхода за границы набора точек при обращении к ним по номеру, наследует от класса **IndexOutOfBoundsException**;

InappropriateFunctionPointException – исключение, выбрасываемое при попытке добавления или изменения точки функции несоответствующим образом, наследует от класса **Exception**.

```java
public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException {
    // Конструктор по умолчанию
    public FunctionPointIndexOutOfBoundsException(){
        super(); // super() - вызов конструктора родителя
    }
    // Конструктор с сообщением
    public FunctionPointIndexOutOfBoundsException(String message){
        super(message);
    }
    // Конструктор с индексом
    public FunctionPointIndexOutOfBoundsException(int index){
        super("Индекс: [" + index + "] выходит за границы");
    }
    // Конструктор с индексом и размером массива
    public FunctionPointIndexOutOfBoundsException(int index, int size){
        super("Индекс: [" + index + "] выходит за границы массива размером " + size);
    }
}
```

Рис. 1 – класс **FunctionPointIndexOutOfBoundsException**

```java
public class InappropriateFunctionPointException extends Exception{
    // Конструктор по умолчанию
    public InappropriateFunctionPointException(){
        super();
    }
    // Конструктор с сообщением
    public InappropriateFunctionPointException(String message){
        super(message);
    }
    // Конструктор с сообщением и причиной
    public InappropriateFunctionPointException(String message, Throwable cause){
        super(message, cause); // Throwable - суперкласс всех ошибок и исключениий в Java
    }
}
```

Рис. 2 – класс **InappropriateFunctionPointException**

Задание 3

Добавим исключения **IllegalArgumentException** в конструкторы **TabulatedFunction** для проверки области определения и количества точек.

```java
// Проверка на область определения
if (leftX >= rightX){
    throw new IllegalArgumentException("Левая граница (" + leftX + ") должна быть меньше правой (" + rightX + ")");
}
// Проверка на кол-во точек
if (pointsCount < 2){
    throw new IllegalArgumentException(s:"Кол-во точек должно быть не меньше двух");
}
```

Рис. 3 – Проверки в конструкторе **TabulatedFunction**

Во втором конструкторе во второй проверке заменим **pointsCount** на **values.length**.

В методы **getPoint()**, **setPoint()**, **getPointX()**, **setPointX()**, **getPointY()**, **setPointY()** и **deletePoint()** добавим исключение **FunctionPointIndexOutOfBoundsException**. Оно будет проверять выходит ли номер, переданный в метод, за границы набора точек. Это обеспечит корректность обращений к точкам функции.

```
// Проверка на номер, выходящий за границы набора точек
if (index < 0 || index >= points.length){
    throw new FunctionPointIndexOutOfBoundsException(index, points.length);
}
```

Рис. 4 – Проверка на номер

Добавим исключения **InappropriateFunctionPointException** в методы **setPoint()** и **setPointX()**. Они будут вызываться в случае, если координата **x** задаваемой точки лежит вне интервала, определяемого значениями соседних точек функции.

```
// Проверяем, лежит ли координата x вне интервала
        if (index > 0 && x <= points[index - 1].getX()){
            throw  new InappropriateFunctionPointException("X координата (" + x + ")
должна быть больше предыдущей точки (" + points[index - 1].getX() + ")");
        }
        if (index < points.length - 1 && x >= points[index + 1].getX()){
             throw  new InappropriateFunctionPointException("X координата (" + x + ")
должна быть меньше следующей точки (" + points[index - 1].getX() + ")");
          }
```

Добавим исключение **InappropriateFunctionPointException** в метод **addPoint()**. Оно будет вызываться в случае, если абсцисса точки из набора точек функции совпадает с абсциссой добавляемой точки.

```
// Проверяем на дубликат
        if (insert_index < points.length && point.getX() ==
points[insert_index].getX())
            throw  new InappropriateFunctionPointException("Точка X с координатой ("
+ point.getX() + ") уже существует");
```

В метод **deletePoin**t добавим исключение, проверяющее, что после удаления останется минимум 2 точки.

```
if (points.length < 3){
            throw new IllegalStateException("Невозможно удалить точку: количество
точек не может быть меньше двух");
        }
```

Задание 4

Реализуем двусвязный циклический список. Он будет состоять из двух классов: **LinkedListTabulatedFunction** и **FunctionNode**. **FunctionNode** будет отвечать за узлы списка, а **LinkedListTabulatedFunction** за весь список. Класс **LinkedListTabulatedFunction** будет совмещать в себе две функциии: будет описывать связный список и работу с ним и будет описывать работы с табулированной функцией и ее точками.

Реализуем первую функцию. Опишем класс **FunctionNode**, класс **LinkedListTabulatedFunction**.

Реализуем методы:

- **FunctionNode getNodeByIndex(int index)** - возвращающий ссылку на объект элемента списка по его номеру
- **FunctionNode addNodeToTail()**, добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента.
- **FunctionNode addNodeByIndex(int index)**, добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента.
- **FunctionNode deleteNodeByIndex(int index)**, удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента.

Задание 5

Реализуем вторую функцию класса **LinkedListTabulatedFunction**. Создадим конструкторы и методы, аналогичные конструкторам и методам класса **TabulatedFunction**. Учтем исключения и не забудем про оптимизацию. Оптимизировать будем благодаря полям **lastIndex** и **lastAccessed.**

Задание 6

Переименуем класс **TabulatedFunction** на **ArrayTabulatedFunction**. Создадим интерфейс **TabulatedFunction**, содержащий объявления общих методов классов **ArrayTabulatedFunction** и **LinkedListTabulatedFunction**.

Оба класса будут реализовать созданный интерфейс. Теперь работы с функциями заключена в типе интерфейса, а в классах заключена только реализация этой работы.

```
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);

    // Методы модификации точек
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Рис. 5 – Интерфейс **TabulatedFunction**

Задание 7

Проведем тесты, сначала убедимся, что базовые функции работают, затем проверим исключения.

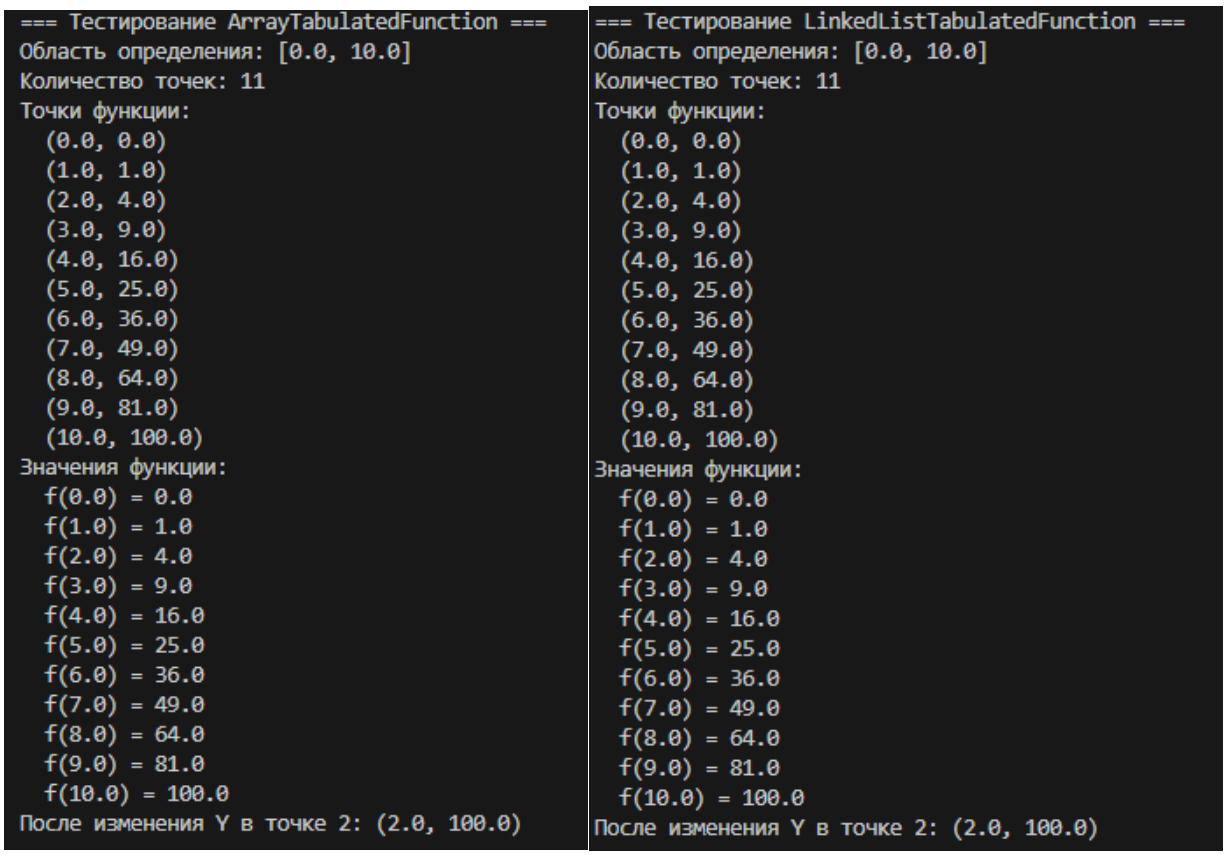
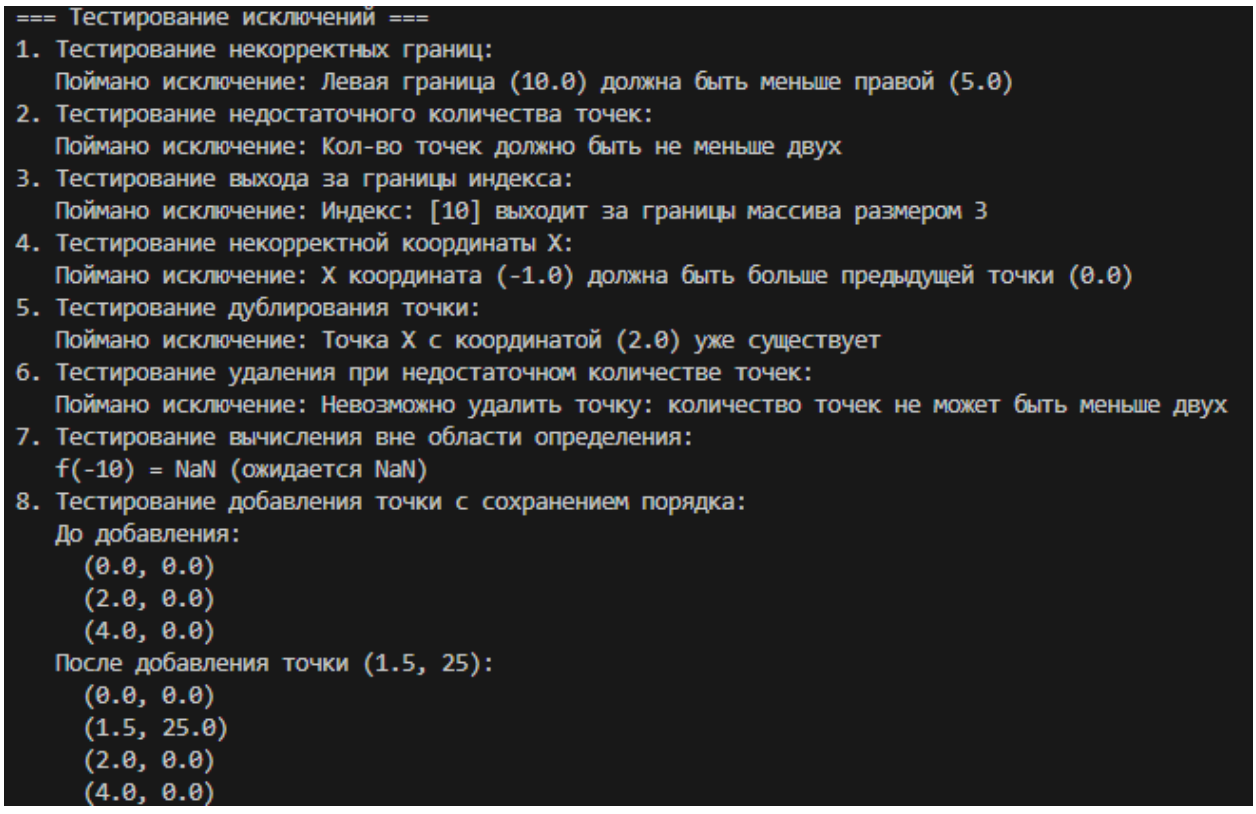


Рис 6-7 Базовые тесты



Из тестов видно, что код рабочий, все тесты выполняются корректно.