Министерство науки и высшего образования Российской

Федерации федеральное государственное автономное

образовательное учреждение высшего образования
«Самарский национальный исследовательский
университет имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе № 3

Дисциплина: «ООП»

Тема «Исключения и интерфейсы»

Выполнил: Сучков Борис Антонович

Группа: 6201-120303D

Самара 2025

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.

В ходе выполнения работы запрещено использовать классы из пакета `java.util`.

Задания

Задание 1

Ознакомиться (изучить документацию) со следующими классами исключений, входящих в API Java:

• `java.lang.Exception` • `java.lang.IndexOutOfBoundsException` • `java.lang.ArrayIndexOutOfBoundsException` • `java.lang.IllegalArgumentException` • `java.lang.IllegalStateException`

Задание 2

В пакете `functions` создать два класса 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`.
При создании классов исключений настоятельно рекомендуется использовать специализированные средства среды разработки.

Задание 3

В разработанный ранее класс `TabulatedFunction` внести изменения, обеспечивающие выбрасывание исключений методами класса.
• Оба конструктора класса должны выбрасывать исключение `IllegalArgumentException`, если левая граница области определения больше или равна правой, а также если предлагаемое количество точек меньше двух. Это обеспечит создание объекта только при корректных параметрах.
•
Методы `getPoint()`, `setPoint()`, `getPointX()`, `setPointX()`, `getPointY()`, `setPointY()` и `deletePoint()` должны выбрасывать исключение `FunctionPointIndexOutOfBoundsException`, если переданный в метод номер выходит за границы набора точек. Это обеспечит корректность обращений к точкам функции.

• Методы `setPoint()` и `setPointX()` должны выбрасывать исключение `InappropriateFunctionPointException` в том случае, если координата `x` задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции. Метод `addPoint()` также должен выбрасывать исключение `InappropriateFunctionPointException`, если в наборе точек функции есть точка, абсцисса которой совпадает с абсциссой добавляемой точки. Это обеспечит сохранение упорядоченности точек функции.

- Метод `deletePoint()` должен выбрасывать исключение `IllegalStateException`, если на момент удаления точки количество точек в наборе менее трех. Это обеспечит невозможность получения функции с некорректным количеством точек.

Задание 4

В пакете `functions` создать класс `LinkedListTabulatedFunction`, объект которого также должен описывать табулированную функцию. Отличие этого класса должно заключаться в том, что для хранения набора точек в нем должен использоваться не массив, а динамическая структура – связный список.
Важно понимать, что представляет собой связный список в объектно-ориентированном программировании. Действительно, в процедурных языках список обычно представляет собой набор записей (или структур) в памяти, некоторые элементы которых являются указателями на соседние элементы списка, а также набор процедур по работе со списком. В Java, во-первых, нет указателей и адресной арифметики, вместо этого будут использоваться ссылки. Во-вторых, вместо записей будут использоваться объекты. В-третьих, процедуры по работе со списком будут заменены на методы класса.

Особенность заключается в том, что для полноценного описания связного списка потребуется реализовать два класса. Ответственность первого из них – элемент списка и его связи, объекты этого класса являются объектами элементов списка, хранящими данные и ссылки на соседние элементы. Ответственность второго – список в целом и операции с ним, именно в нем реализуются методы по манипулированию списком, а его объект – это объект списка целиком.

Таким образом, в первом классе должны присутствовать информационное поле и поля связи. А во втором классе должна храниться ссылка на объект головы списка и должны быть описаны методы для работы со списком, причем публичные методы не должны получать или возвращать ссылки на объекты элементов списка, т.к. это будет нарушением инкапсуляции. Можно сказать, что между этими классами имеется отношение композиции: объекты элементов списка являются частями объекта списка и не существуют вне его.

При написании «внешнего» класса следует учесть, что инкапсуляция работы со списком в отдельный объект позволяет несколько изменить алгоритмы по работе

со списком по сравнению с обычными алгоритмами в процедурном исполнении. Так, не имеет смысла подсчитывать каждый раз длину списка (ведь никто кроме самого объекта списка не может его изменить), вместо этого проще хранить ее как поле «внешнего» объекта. Также очевидно, что часто работа с элементами списка ведется последовательно или в соседних элементах, поэтому пробегать каждый раз от головы при доступе к элементу списка неразумно, проще хранить ссылку и номер элемента, к которому было последнее обращение, и в некоторых случаях двигаться от него.

В качестве структуры списка в настоящей работе требуется использовать двусвязный циклический список с выделенной головой. Первое означает, что объект элемента списка хранит две ссылки – на предыдущий элемент и на следующий элемент. Второе означает, что голова списка не используется для хранения данных и присутствует в списке всегда (пустой список представляет собой объект головы, ссылающийся сам на себя и как на предыдущий элемент, и как на следующий).

Класс `LinkedListTabulatedFunction` совмещает в себе две функции: с одной стороны, он будет описывать связный список и работу с ним, а с другой стороны, он будет описывать работу с табулированной функцией и ее точками. Для реализации первой функции требуется выполнить следующие действия.

1. Описать класс элементов списка `FunctionNode`, содержащий информационное поле для хранения данных типа `FunctionPoint`, а также поля для хранения ссылок на предыдущий и следующий элемент. Выберите и обоснуйте место описания класса и его видимость. Также выберите и обоснуйте реализацию инкапсуляции в этом классе.
2. Описать класс `LinkedListTabulatedFunction` объектов списка, содержащий поле ссылки на объект головы, а также иные вспомогательные поля.
3. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode getNodeByIndex(int index)`, возвращающий ссылку на объект элемента списка по его номеру. Нумерация значащих элементов (голова списка в данном случае к ним не относится) должна начинаться с 0. Метод должен обеспечивать оптимизацию доступа к элементам списка.
4. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode addNodeToTail()`, добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента.
5. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode addNodeByIndex(int index)`, добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента.
6. В классе `LinkedListTabulatedFunction` реализовать метод `FunctionNode deleteNodeByIndex(int index)`, удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента.

Задание 5

Для обеспечения второй функции класса `LinkedListTabulatedFunction` реализовать в классе конструкторы и методы, аналогичные конструкторам и методам класса `TabulatedFunction`. Конструкторы должны иметь те же параметры, методы должны иметь те же сигнатуры. Также должны выбрасываться те же виды исключений в тех же случаях.
С одной стороны, при написании методов следует использовать уже написанные методы, отвечающие за работу со связным списком. С другой стороны, инкапсуляция в одном классе и списка, и табулированной функции позволяет в некоторых методах, относящихся к табулированной функции, значительно оптимизировать работу за счет возможности обращаться к элементам списка напрямую. Определите такие методы и оптимизируйте их.

Задание 6

Класс `TabulatedFunction` переименовать в класс `ArrayTabulatedFunction`.
Создать интерфейс `TabulatedFunction`, содержащий объявления общих методов классов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`.
Сделать так, чтобы оба класса функций реализовывали созданный интерфейс.

Теперь суть работы с табулированными функциями заключена в типе интерфейса, а в классах заключена только реализация этой работы.

Задание 7

Проверить работу написанных классов.

Для этого в созданном ранее классе `Main`, содержащем точку входа программы, добавить проверку для случаев, в которых объект табулированной функции должен выбрасывать исключения.
Ссылочную переменную для работы с объектом функции следует объявить типа `TabulatedFunction`, а при создании объекта следует указать реальный класс. Тогда проверка работы класса `LinkedListTabulatedFunction` может быть произведена путем простой замены класса, объект которого создается.

ЗАДАНИЕ 1

Я ознакомился (изучил документацию) со следующими классами исключений, входящих в API Java:

• java.lang.Exception • java.lang.IndexOutOfBoundsException • java.lang.ArrayIndexOutOfBoundsException • java.lang.IllegalArgumentException • java.lang.IllegalStateException

ЗАДАНИЕ 2

В пакете `functions` я создал два класса 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`.

FunctionPointIndexOutOfBoundsException.java 1, U

Lab-2-2025 > src > functions > FunctionPointIndexOutOfBoundsException.java > ...

```java
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException {
    public FunctionPointIndexOutOfBoundsException() {
        super();
    }

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

InappropriateFunctionPointException.java 1, U

Lab-2-2025 > src > functions > InappropriateFunctionPointException.java > ...

```java
package functions;

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

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

ЗАДАНИЕ 3

В разработанный ранее класс `TabulatedFunction` я внёс изменения, обеспечивающие выбрасывание исключений методами класса.

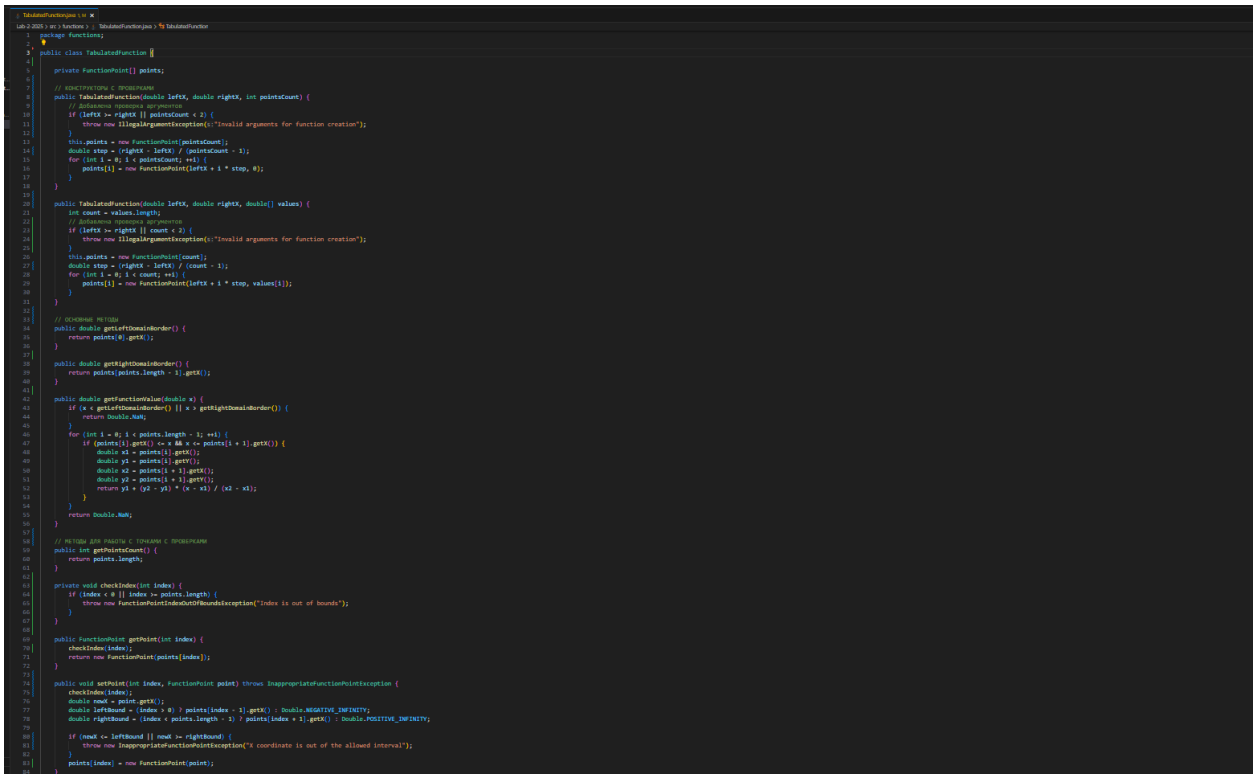
• Оба конструктора класса выбрасывают исключение `IllegalArgumentException`, если левая граница области определения больше или равна правой, а также если предлагаемое количество точек меньше двух. Это обеспечивает создание объекта только при корректных параметрах.

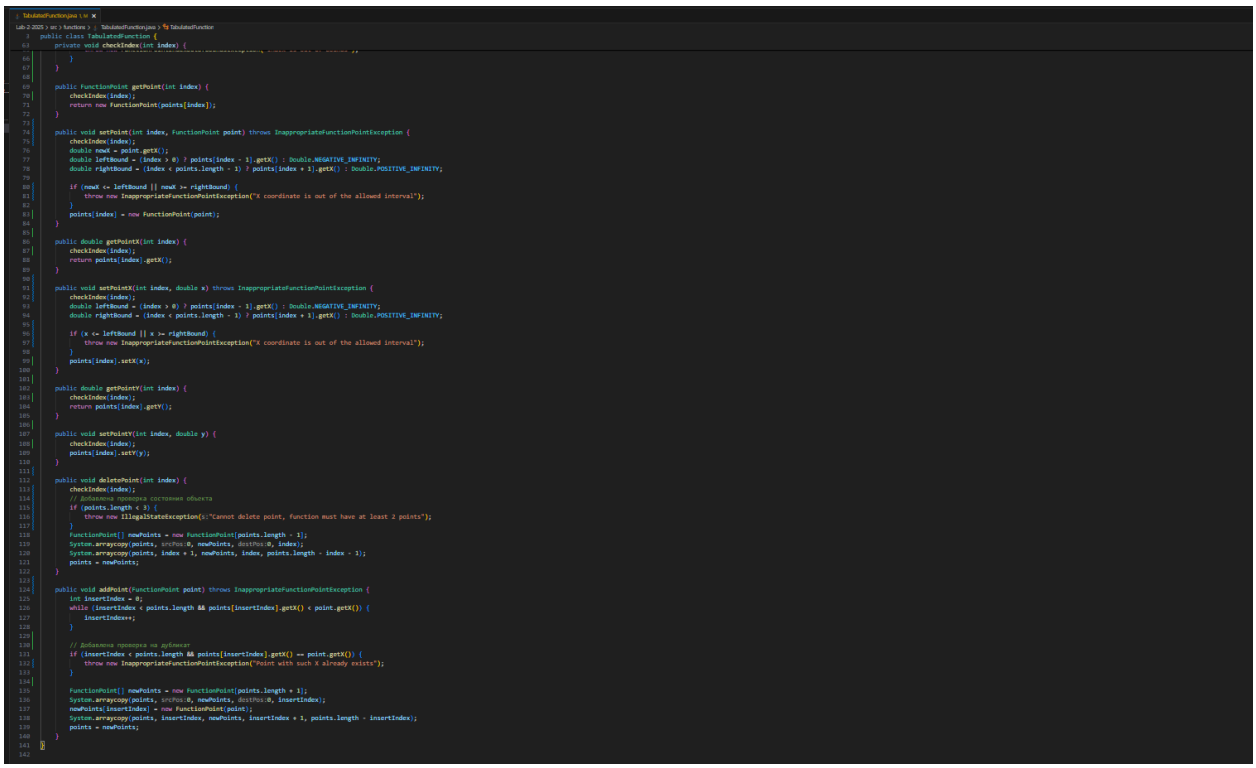
•

Методы `getPoint()`, `setPoint()`, `getPointX()`, `setPointX()`, `getPointY()`, `setPointY()` и `deletePoint()` выбрасывают

исключение `FunctionPointIndexOutOfBoundsException`, если переданный в метод номер выходит за границы набора точек. Это обеспечивает корректность обращений к точкам функции.
• Методы `setPoint()` и `setPointX()` выбрасывают исключение `InappropriateFunctionPointException` в том случае, если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции. Метод `addPoint()` также выбрасывает исключение `InappropriateFunctionPointException`, если в наборе точек функции есть точка, абсцисса которой совпадает с абсциссой добавляемой точки. Это обеспечивает сохранение упорядоченности точек функции.

- Метод `deletePoint()` выбрасывает исключение `IllegalStateException`, если на момент удаления точки количество точек в наборе менее трех. Это обеспечивает невозможность получения функции с некорректным количеством точек.





ЗАДАНИЯ 4 И 5

В пакете functions реализован класс LinkedListTabulatedFunction, объект которого описывает табулированную функцию. Для хранения набора точек используется динамическая структура данных – двусвязный циклический список с выделенной головой.

Внутри класса LinkedListTabulatedFunction реализован вложенный (inner) класс FunctionNode, который инкапсулирует данные одного узла списка и хранит ссылки на соседние элементы. Класс содержит поля: FunctionPoint point для хранения данных точки функции (значения *x* и *y*), FunctionNode next – ссылку на следующий элемент списка, и FunctionNode prev – ссылку на предыдущий элемент списка. Класс FunctionNode объявлен как private static, что полностью скрывает его реализацию от внешнего кода и обеспечивает инкапсуляцию. Работа с элементами осуществляется исключительно через методы внешнего класса LinkedListTabulatedFunction.

Класс LinkedListTabulatedFunction содержит следующие основные поля: FunctionNode head – ссылка на объект головы списка (не хранящий данных), при этом в пустом списке head.next = head и head.prev = head; int count – текущее количество элементов (точек) в списке; FunctionNode current и int currentIndex – поля для оптимизации доступа к элементам списка, позволяющие не начинать поиск всегда от головы.

Реализован метод FunctionNode getNodeByIndex(int index), возвращающий ссылку на узел списка по индексу. Нумерация начинается с 0, голова списка не учитывается. Метод использует оптимизированный алгоритм доступа: если запрошенный индекс ближе к текущему (currentIndex), движение начинается от текущего узла; если ближе к голове или хвосту – поиск начинается с соответствующей стороны. Это уменьшает количество переходов между элементами и ускоряет доступ.

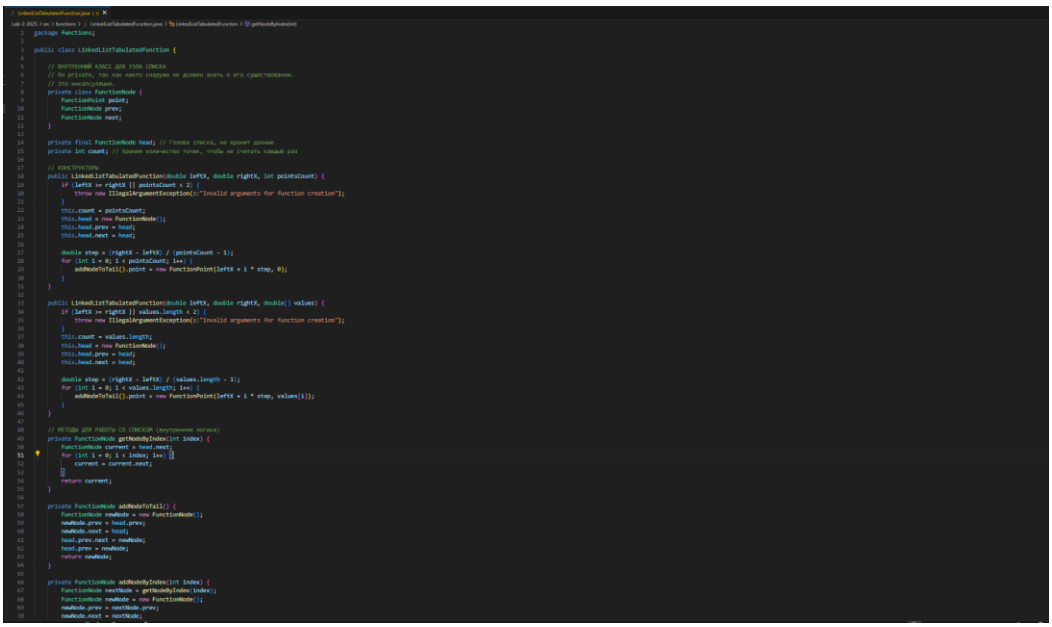
Реализован метод FunctionNode addNodeToTail(), добавляющий новый элемент в конец списка. Метод создает новый узел, вставляет его перед головой (head), обновляет ссылки prev и next, увеличивает счётчик элементов count и возвращает ссылку на созданный узел.

Реализован метод FunctionNode addNodeByIndex(int index), вставляющий новый элемент на указанную позицию. Алгоритм работы следующий: находится узел, который должен следовать после вставляемого, создается новый узел, обновляются связи prev и next у соседних узлов, увеличивается count и возвращается ссылка на вставленный элемент. Метод корректно обрабатывает граничные случаи – вставку в начало и конец списка.

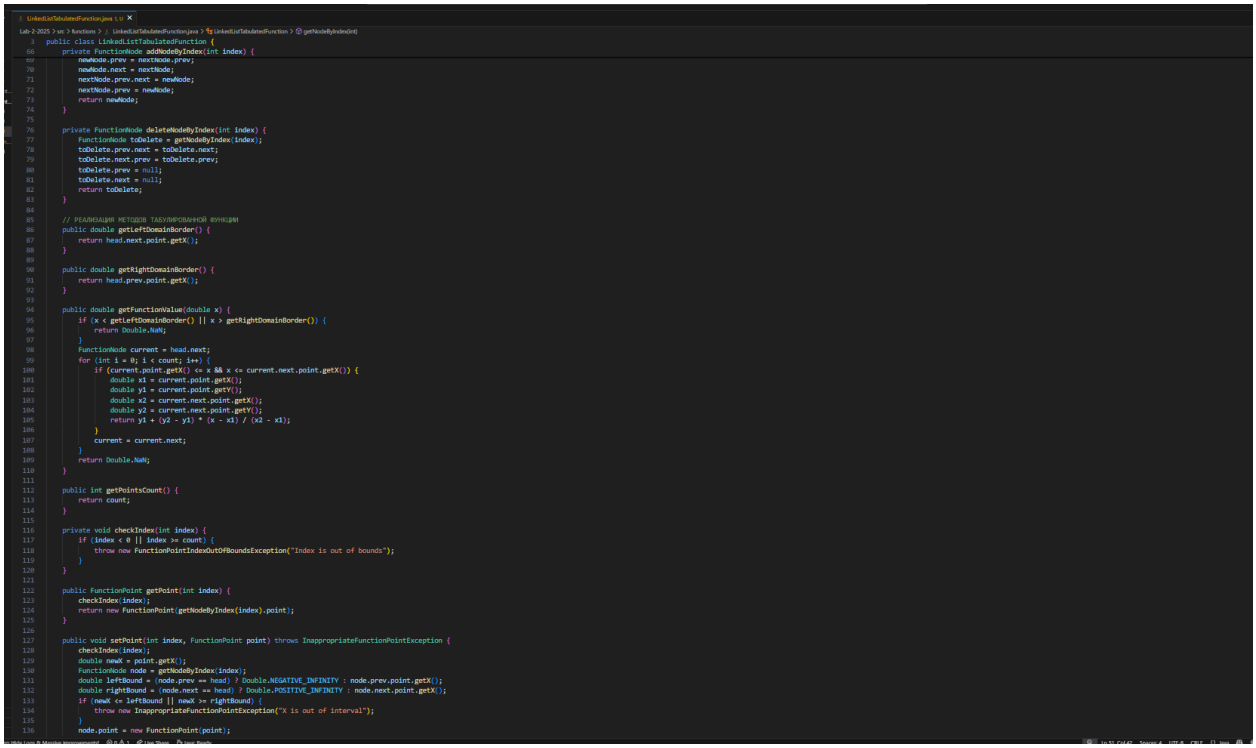
Реализован метод FunctionNode deleteNodeByIndex(int index), удаляющий элемент по индексу. Метод находит удаляемый узел с помощью getNodeByIndex(index), перестраивает связи prev и next у соседних узлов, уменьшает счётчик count и возвращает ссылку на удалённый узел.

Используется двусвязный циклический список с выделенной головой, что упрощает работу с граничными элементами (нет необходимости проверять null). Поддерживается полная инкапсуляция – внешние пользователи не имеют доступа к элементам списка напрямую. Хранение count, current и currentIndex обеспечивает высокую эффективность операций добавления, удаления и обращения к элементам. Вся логика манипуляций со списком инкапсулирована внутри класса LinkedListTabulatedFunction.

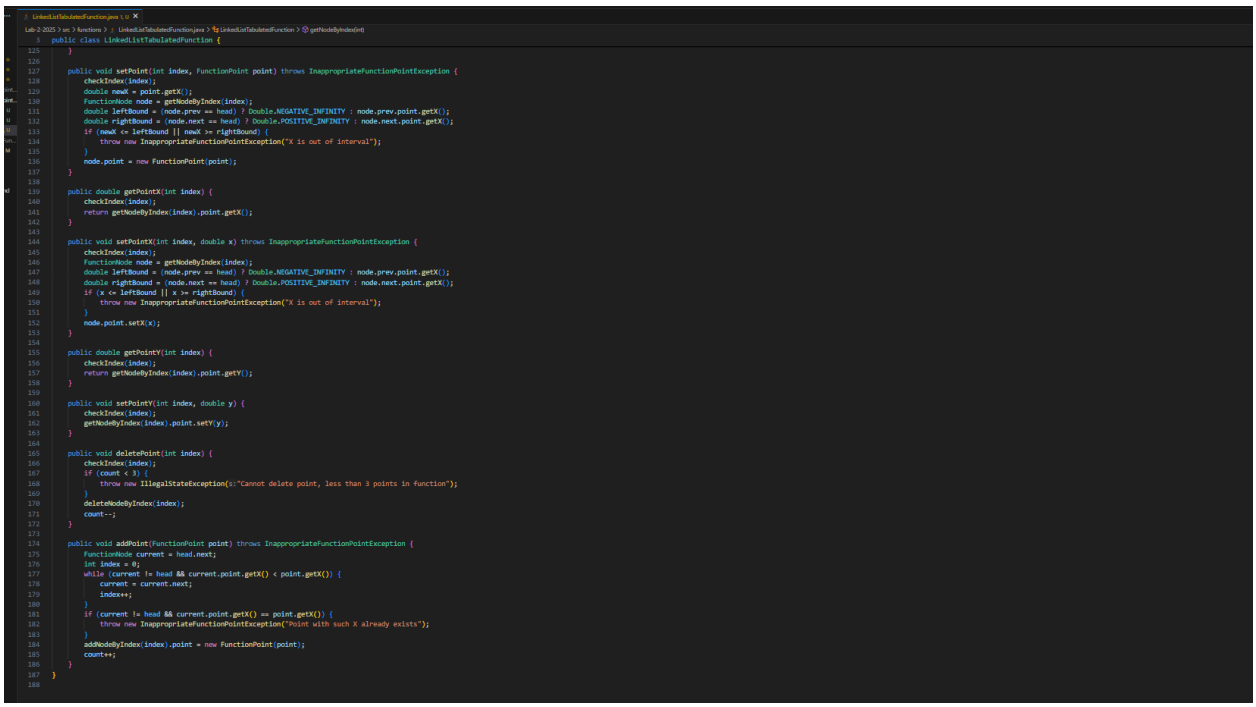
Для обеспечения второй функции класса `LinkedListTabulatedFunction` я реализовал в классе конструкторы и методы, аналогичные конструкторам и методам класса `TabulatedFunction`. Конструкторы имеют те же параметры, методы имеют те же сигнатуры. Также выбрасываются те же виды исключений в тех же случаях.
С одной стороны, при написании методов используются уже написанные методы, отвечающие за работу со связным списком. С другой стороны, инкапсуляция в одном классе и списка, и табулированной функции позволяет в некоторых методах, относящихся к табулированной функции, значительно оптимизировать работу за счёт возможности обращаться к элементам списка напрямую. Я определил такие методы и оптимизировал их.



```
package functions;

public class LinkedListTabulatedFunction {

    // ВНУТРЕННИЙ КЛАСС ДЛЯ УЗЛА СПИСКА
    // Он private, так как никто снаружи не должен знать о его существовании.
    // Это инкапсуляция.
    private class FunctionNode {
        FunctionPoint point;
        FunctionNode prev;
        FunctionNode next;
    }

    private final FunctionNode head; // Голова списка, не хранит данные
    private int count; // Храним количество точек, чтобы не считать каждый раз

    // КОНСТРУКТОРЫ
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX || pointsCount < 2) {
            throw new IllegalArgumentException("Invalid arguments for function creation");
        }
        this.count = pointsCount;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

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

    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
        if (leftX >= rightX || values.length < 2) {
            throw new IllegalArgumentException("Invalid arguments for function creation");
        }
        this.count = values.length;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            addNodeToTail().point = new FunctionPoint(leftX + i * step, values[i]);
        }
    }

    // МЕТОДЫ ДЛЯ РАБОТЫ СО СПИСКОМ (внутренние хелперы)
    private FunctionNode getNodeByIndex(int index) {
        FunctionNode current = head.next;
        for (int i = 0; i < index; i++) {
            current = current.next;
        }
        return current;
    }

    private FunctionNode addNodeToTail() {
        FunctionNode newNode = new FunctionNode();
        newNode.prev = head.prev;
        newNode.next = head;
        head.prev.next = newNode;
        head.prev = newNode;
        return newNode;
    }

    private FunctionNode addNodeByIndex(int index) {
        FunctionNode nextNode = getNodeByIndex(index);
        FunctionNode newNode = new FunctionNode();
        newNode.prev = nextNode.prev;
        newNode.next = nextNode;
```



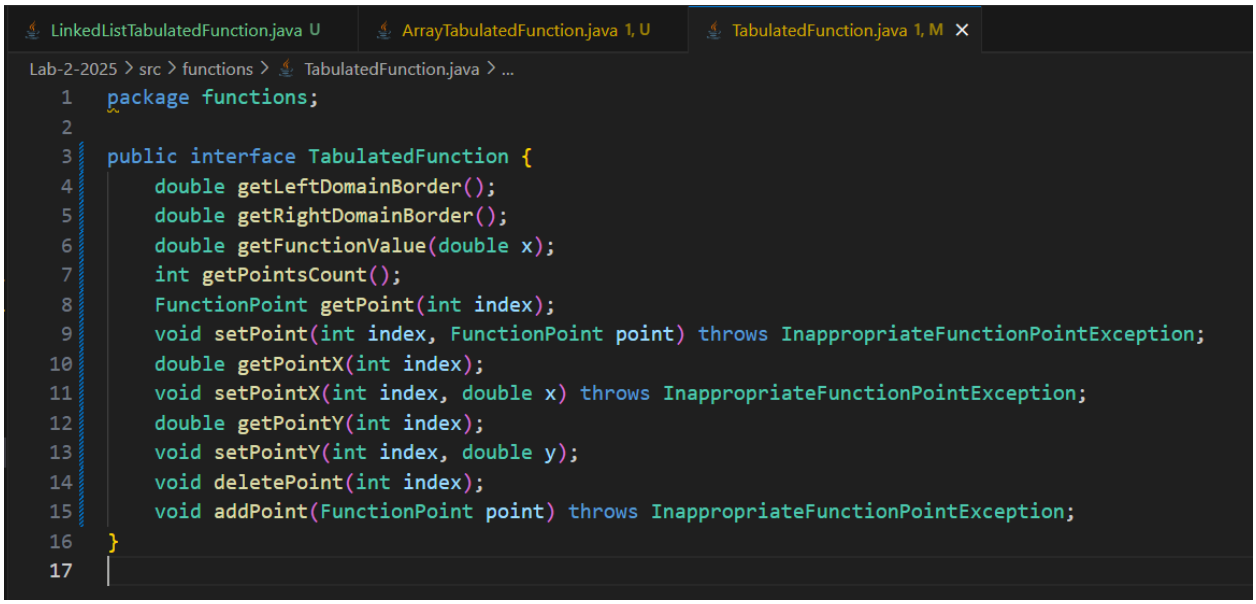
```
public class LinkedListTabulatedFunction {
    private FunctionNode addNodeByIndex(int index) {
        newNode.prev = nextNode.prev;
        newNode.next = nextNode;
        nextNode.prev.next = newNode;
        nextNode.prev = newNode;
        return newNode;
    }

    private FunctionNode deleteNodeByIndex(int index) {
        FunctionNode toDelete = getNodeByIndex(index);
        toDelete.prev.next = toDelete.next;
        toDelete.next.prev = toDelete.prev;
        toDelete.prev = null;
        toDelete.next = null;
        return toDelete;
    }

    // РЕАЛИЗАЦИЯ МЕТОДОВ ТАБУЛИРОВАННОЙ ФУНКЦИИ
    public double getLeftDomainBorder() {
        return head.next.point.getX();
    }

    public double getRightDomainBorder() {
        return head.prev.point.getX();
    }

    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        FunctionNode current = head.next;
        for (int i = 0; i < count; i++) {
            if (current.point.getX() <= x && x <= current.next.point.getX()) {
                double x1 = current.point.getX();
                double y1 = current.point.getY();
                double x2 = current.next.point.getX();
                double y2 = current.next.point.getY();
                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
            }
            current = current.next;
        }
        return Double.NaN;
    }

    public int getPointsCount() {
        return count;
    }

    private void checkIndex(int index) {
        if (index < 0 || index >= count) {
            throw new FunctionPointIndexOutOfBoundsException("Index is out of bounds");
        }
    }

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

    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
        checkIndex(index);
        double newX = point.getX();
        FunctionNode node = getNodeByIndex(index);
        double leftBound = (node.prev == head) ? Double.NEGATIVE_INFINITY : node.prev.point.getX();
        double rightBound = (node.next == head) ? Double.POSITIVE_INFINITY : node.next.point.getX();
        if (newX <= leftBound || newX >= rightBound) {
            throw new InappropriateFunctionPointException("X is out of interval");
        }
        node.point = new FunctionPoint(point);
```



ЗАДАНИЕ 6

Класс `TabulatedFunction` я переименовал в класс `ArrayTabulatedFunction`.
Создал интерфейс `TabulatedFunction`, содержащий объявления общих методов классов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`.
Сделал так, чтобы оба класса функций реализовывали созданный интерфейс.

Теперь суть работы с табулированными функциями заключена в типе интерфейса, а в классах заключена только реализация этой работы.

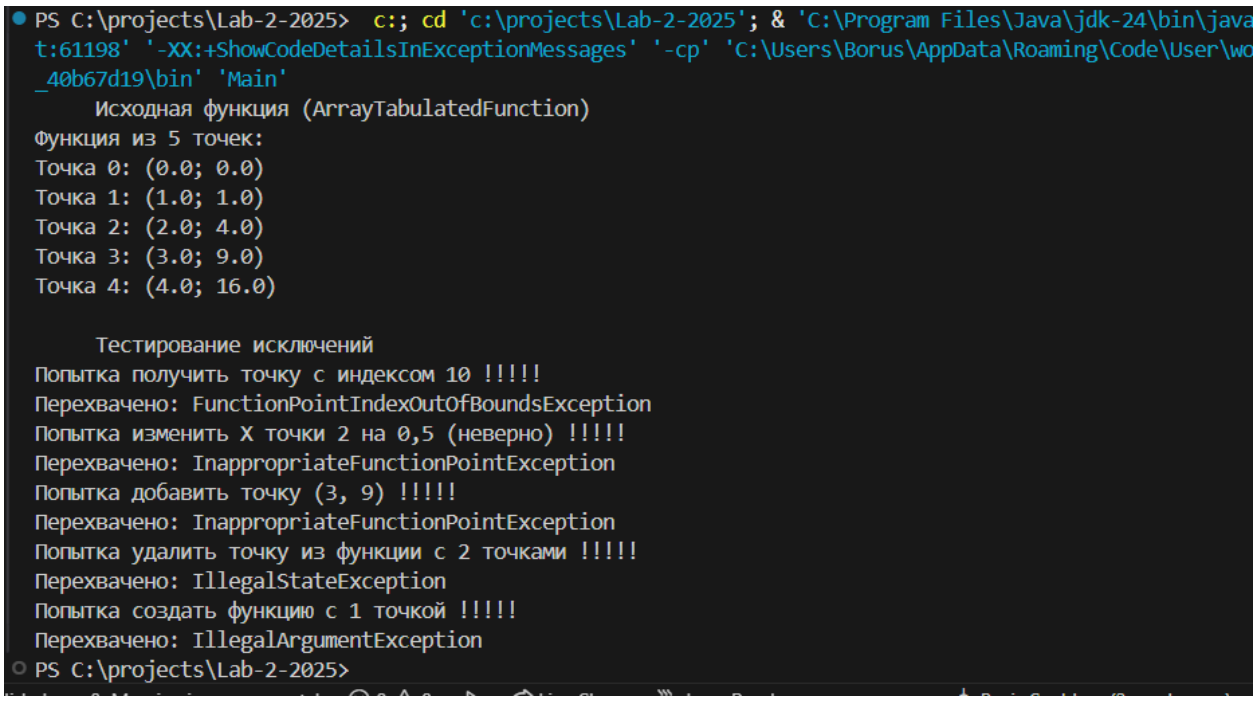


ЗАДАНИЕ 7

Я проверить работу написанных классов.

Для этого в созданном ранее классе `Main`, содержащем точку входа программы, я добавить проверку для случаев, в которых объект табулированной функции должен выбрасывать исключения.
Ссылочную переменную для работы с объектом функции я объявил типа `TabulatedFunction`, а при создании объекта указал реальный класс. Теперь проверка работы класса `LinkedListTabulatedFunction` может быть произведена путем простой замены класса, объект которого создается.

```java
import functions.*;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        double[] values = { 0, 1, 4, 9, 16 };

        // Используем тип интерфейса
        TabulatedFunction function = new ArrayTabulatedFunction(leftX:0, rightX:4, values);
        // TabulatedFunction function = new LinkedListTabulatedFunction(0, 4, values); // - для другой реализации

        System.out.println("     Исходная функция (" + function.getClass().getSimpleName() + ")");
        printFunction(function);

        System.out.println(x:"\n     Тестирование исключений");

        // 1. Выход за границы индекса
        try {
            System.out.println(x:"Попытка получить точку с индексом 10 !!!!!");
            function.getPoint(index:10);
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
        }

        // 2. Некорректный X при изменении
        try {
            System.out.println(x:"Попытка изменить X точки 2 на 0,5 (неверно) !!!!!");
            function.setPointX(index:2, x:0.5);
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
        }

        // 3. Добавление точки с существующим X
        try {
            System.out.println(x:"Попытка добавить точку (3, 9) !!!!!");
            function.addPoint(new FunctionPoint(x:3.0, y:9.0));
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
        }

        // 4. Удаление из маленькой функции
        try {
            System.out.println(x:"Попытка удалить точку из функции с 2 точками !!!!!");
            TabulatedFunction smallFunc = new ArrayTabulatedFunction(leftX:0, rightX:1, new double[]{0, 1});
            smallFunc.deletePoint(index:0);
        } catch (IllegalStateException e) {
            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
        }

        // 5. Неверные аргументы конструктора
        try {
            System.out.println(x:"Попытка создать функцию с 1 точкой !!!!!");
            new ArrayTabulatedFunction(leftX:0, rightX:1, pointsCount:1);
        } catch (IllegalArgumentException e) {
            System.out.println("Перехвачено: " + e.getClass().getSimpleName());
        }
    }

    public static void printFunction(TabulatedFunction func) {
        System.out.println("Функция из " + func.getPointsCount() + " точек:");
        for (int i = 0; i < func.getPointsCount(); i++) {
            FunctionPoint p = func.getPoint(i);
            System.out.println("Точка " + i + ": (" + p.getX() + "; " + p.getY() + ")");
        }
    }
}
```


PS C:\projects\Lab-2-2025> c:; cd 'c:\projects\Lab-2-2025'; & 'C:\Program Files\Java\jdk-24\bin\java
t:61198' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\wo
_40b67d19\bin' 'Main'
Исходная функция (ArrayTabulatedFunction)
Функция из 5 точек:
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)
Тестирование исключений
Попытка получить точку с индексом 10 !!!!!
Перехвачено: FunctionPointIndexOutOfBoundsException
Попытка изменить X точки 2 на 0,5 (неверно) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка добавить точку (3, 9) !!!!!
Перехвачено: InappropriateFunctionPointException
Попытка удалить точку из функции с 2 точками !!!!!
Перехвачено: IllegalStateException
Попытка создать функцию с 1 точкой !!!!!
Перехвачено: IllegalArgumentException
PS C:\projects\Lab-2-2025>