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»

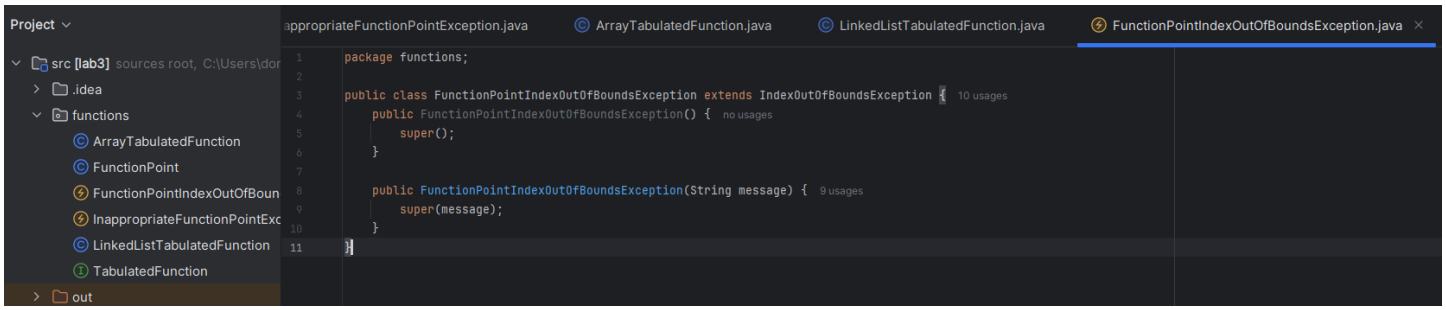
Выполнил: Дорофеева Александра Алексеевна

Группа: 6201-120303D

Проверил: преподаватель Борисов Д.С.

Самара, 2025

Задание 2



Созданы два специализированных класса исключений в пакете functions:

• FunctionPointIndexOutOfBoundsException — исключение выхода за границы набора точек при обращении к ним по номеру, наследуется от класса IndexOutOfBoundsException;

• InappropriateFunctionPointException — исключение, возникающее при попытке добавить или изменить точку функции ненадлежащим образом.

Задание 3

```java
// Конструкторы с проверками IllegalArgumentException
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {  7 usages
    // Проверка условий из задания 3
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не меньше двух");
    }

    this.pointsCount = pointsCount;
    this.points = new FunctionPoint[Math.max(pointsCount * 2, INITIAL_CAPACITY)];

    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        points[i] = new FunctionPoint(x, y: 0.0);
    }
}

public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {  no usages
    // Проверка условий из задания 3
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }
    if (values.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не меньше двух");
    }

    this.pointsCount = values.length;
    this.points = new FunctionPoint[Math.max(values.length * 2, INITIAL_CAPACITY)];

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

Реализация конструкторов класса ArrayTabulatedFunction с проверками корректности аргументов

```java
// Вспомогательный метод для проверки индекса
private void checkIndex(int index) {  7 usages
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона точек [0, " + (pointsCount - 1) + "]");
    }
}
```

Для обработки некорректных значений индекса в методах работы с точками создан вспомогательный метод checkIndex(), который выбрасывает исключение FunctionPointIndexOutOfBoundsException при выходе индекса за границы массива. Данная проверка добавлена в начале методов getPoint, setPoint, getPointX, setPointX, getPointY, setPointY и deletePoint.

```java
public FunctionPoint getPoint(int index) {  no usages
    checkIndex(index);
    return new FunctionPoint(points[index]);
}

public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {  2 usages
    checkIndex(index);

    // Проверка порядка точек из задания 3
    if (index > 0 && point.getX() <= points[index - 1].getX() + EPSILON) {
        throw new InappropriateFunctionPointException("Координата X должна быть больше предыдущей точки");
    }
    if (index < pointsCount - 1 && point.getX() >= points[index + 1].getX() - EPSILON) {
        throw new InappropriateFunctionPointException("Координата X должна быть меньше следующей точки");
    }

    points[index] = new FunctionPoint(point);
}

public double getPointX(int index) {  3 usages
    checkIndex(index);
    return points[index].getX();
}

public void setPointX(int index, double x) throws InappropriateFunctionPointException {  no usages
    checkIndex(index);

    // Проверка порядка точек из задания 3
    if (index > 0 && x <= points[index - 1].getX() + EPSILON) {
        throw new InappropriateFunctionPointException("Координата X должна быть больше предыдущей точки");
    }
    if (index < pointsCount - 1 && x >= points[index + 1].getX() - EPSILON) {
        throw new InappropriateFunctionPointException("Координата X должна быть меньше следующей точки");
    }

    points[index].setX(x);
}
```

```java
    public void setPointX(int index, double x) throws InappropriateFunctionPointException {  no usages
        checkIndex(index);

        // Проверка порядка точек из задания 3
        if (index > 0 && x <= points[index - 1].getX() + EPSILON) {
            throw new InappropriateFunctionPointException("Координата X должна быть больше предыдущей точки");
        }
        if (index < pointsCount - 1 && x >= points[index + 1].getX() - EPSILON) {
            throw new InappropriateFunctionPointException("Координата X должна быть меньше следующей точки");
        }

        points[index].setX(x);
    }

    public double getPointY(int index) {  2 usages
        checkIndex(index);
        return points[index].getY();
    }

    public void setPointY(int index, double y) {  1 usage
        checkIndex(index);
        points[index].setY(y);
    }

    // Методы изменения количества точек
    public void deletePoint(int index) {  1 usage
        checkIndex(index);

        // Проверка из задания 3
        if (pointsCount <= 2) {
            throw new IllegalStateException("Нельзя удалить точку - останется меньше двух точек");
        }

        System.arraycopy(points, srcPos: index + 1, points, index, length: pointsCount - index - 1);
        pointsCount--;
        points[pointsCount] = null;
    }
```

В методы setPoint() и setPointX() дополнительно внедрена проверка корректности значения координаты x, обеспечивающая её нахождение между соседними точками для сохранения монотонности области определения.

В метод addPoint() добавлена проверка на отсутствие точки с таким же значением x во избежание дублирования координат в области определения

Задание 4

Создаем в пакете functions класс LinkedListTabulatedFunction

```java
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction {  1 usage
    private static class FunctionNode {  29 usages
        private FunctionPoint point;  17 usages
        private FunctionNode prev;  11 usages
        private FunctionNode next;  26 usages

        public FunctionNode(FunctionPoint point) {  6 usages
            this.point = point;
        }
    }
```

Обьявляем класс и создадим структуру для головы списка. Мы также опишем внутренний класс FunctionNode, который будет хранить объект точки (FunctionPoint), а также ссылки на предыдущий и следующий элементы списка. Этот класс должен быть вложенным (private static class), чтобы обеспечить инкапсуляцию, то есть внешний код не должен иметь доступа к внутренней структуре списка.

```java
    private FunctionNode getNodeByIndex(int index) {  10 usages
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы количества точек");
        }

        // Оптимизация: начинаем с последнего доступного узла
        FunctionNode node;
        int startIndex;

        if (lastAccessedNode != null && Math.abs(index - lastAccessedIndex) < Math.min(index, pointsCount - index)) {
            node = lastAccessedNode;
            startIndex = lastAccessedIndex;
        } else if (index < pointsCount - index) {
            node = head.next;
            startIndex = 0;
        } else {
            node = head.prev;
            startIndex = pointsCount - 1;
        }

        // Двигаемся к нужному узлу
        while (startIndex < index) {
            node = node.next;
            startIndex++;
        }
        while (startIndex > index) {
            node = node.prev;
            startIndex--;
        }

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

Добавляется метод безопасного удаления узла по индексу с проверкой границ и минимального количества точек, обновлением связей списка и механизма кэширования lastAccessedNode для оптимизации последующих операций.

```
private FunctionNode addNodeToTail() {  no usages
    FunctionNode newNode = new FunctionNode(new FunctionPoint( x: 0,  y: 0));
    insertNodeAfter(head.prev, newNode);
    pointsCount++;
    lastAccessedNode = newNode;
    lastAccessedIndex = pointsCount - 1;
    return newNode;
}
```

Добавляется метод добавления узла в конец списка с созданием новой точки, обновлением связей, увеличением счётчика точек и обновлением кэша lastAccessedNode.

```
private FunctionNode addNodeByIndex(int index) {  1 usage
    if (index < 0 || index > pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы");
    }

    FunctionNode newNode = new FunctionNode(new FunctionPoint( x: 0,  y: 0));
    FunctionNode targetNode = (index == 0) ? head : getNodeByIndex(index - 1);
    insertNodeAfter(targetNode, newNode);
    pointsCount++;
    lastAccessedNode = newNode;
    lastAccessedIndex = index;
    return newNode;
}
```

Добавляется метод вставки узла по указанному индексу с проверкой корректности позиции, обновлением связей списка и механизма кэширования.

```
private FunctionNode deleteNodeByIndex(int index) {  1 usage
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " выходит за границы количества точек");
    }

    if (pointsCount <= 2) {
        throw new IllegalStateException("Нельзя удалить точку - останется меньше 2 точек");
    }

    FunctionNode nodeToDelete = getNodeByIndex(index);
    nodeToDelete.prev.next = nodeToDelete.next;
    nodeToDelete.next.prev = nodeToDelete.prev;
    pointsCount--;

    // Обновляем lastAccessedNode
    if (lastAccessedNode == nodeToDelete) {
        lastAccessedNode = (index < pointsCount) ? nodeToDelete.next : head.next;
        lastAccessedIndex = (index < pointsCount) ? index : 0;
    } else if (lastAccessedIndex > index) {
        lastAccessedIndex--;
    }

    return nodeToDelete;
}
```

Добавляется метод удаления узла по индексу с проверкой границ и минимального количества точек, перелинковкой соседних узлов, уменьшением счётчика и корректировкой кэша lastAccessedNode.

Задание 5

```
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {  1 usage
    if (leftX >= rightX || pointsCount < 2) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой и точек должно быть не менее 2");
    }

    this.pointsCount = pointsCount;
    initList(leftX, rightX, pointsCount);
}

public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {  no usages
    if (leftX >= rightX || values.length < 2) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой и точек должно быть не менее 2");
    }

    this.pointsCount = values.length;
    initListWithValues(leftX, rightX, values);
}
```

Класс LinkedListTabulatedFunction должен реализовать два конструктора, аналогичных ArrayTabulatedFunction:

1.Конструктор с границами и количеством точек - создаёт равномерную сетку точек в заданном интервале

2.Конструктор с границами и массивом значений - создаёт точки на основе переданных значений y с равномерным распределением x

```
    public FunctionPoint getPoint(int index) {
        FunctionNode node = getNodeByIndex(index);
        return new FunctionPoint(node.point);
    }
```

Реализуется метод получения точки по индексу через поиск соответствующего узла списка с возвращением копии объекта FunctionPoint для обеспечения инкапсуляции.

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

    public void setPointX(int index, double x) throws InappropriateFunctionPointException {
        FunctionPoint point = new FunctionPoint(x, getPointY(index));
        setPoint(index, point);
    }

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

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

    public void deletePoint(int index) {
        deleteNodeByIndex(index);
    }
```

Реализуются базовые методы работы с точками: получение и установка координат X/Y по индексу через обращение к соответствующим узлам списка, а также удаление точки с использованием внутреннего метода deleteNodeByIndex.

```
public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {  2 usages
    FunctionNode node = getNodeByIndex(index);

    // Проверка порядка x
    if ((index > 0 && point.getX() <= getNodeByIndex(index - 1).point.getX() + EPSILON) ||
            (index < pointsCount - 1 && point.getX() >= getNodeByIndex(index + 1).point.getX() - EPSILON)) {
        throw new InappropriateFunctionPointException("Нарушен порядок x координат");
    }

    node.point = new FunctionPoint(point);
}
```

Реализуется метод изменения точки с проверкой сохранения порядка координат X: новое значение должно строго находиться между соседними точками, иначе выбрасывается InappropriateFunctionPointException.

```
    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {  2 usages
        // Проверка на существование точки с таким X
        FunctionNode current = head.next;
        while (current != head) {
            if (Math.abs(current.point.getX() - point.getX()) < EPSILON) {
                throw new InappropriateFunctionPointException("Точка с x=" + point.getX() + " уже существует");
            }
            current = current.next;
        }

        // Находим позицию для вставки
        int insertIndex = 0;
        current = head.next;
        while (current != head && current.point.getX() < point.getX()) {
            current = current.next;
            insertIndex++;
        }

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

Реализуется метод добавления новой точки с проверкой уникальности координаты X и вставкой в позици, сохраняющую возрастающий порядок X в списке.

```java
public double getFunctionValue(double x) {  1 usage
    if (x < getLeftDomainBorder() - EPSILON || x > getRightDomainBorder() + EPSILON) {
        return Double.NaN;
    }

    // Находим отрезок, содержащий x
    FunctionNode current = head.next;
    while (current.next != head && current.next.point.getX() < x + EPSILON) {
        current = current.next;
    }

    if (current.next == head) {
        return current.point.getY();
    }

    // Линейная интерполяция
    FunctionPoint left = current.point;
    FunctionPoint right = current.next.point;

    return left.getY() + (right.getY() - left.getY()) *
            (x - left.getX()) / (right.getX() - left.getX());
}
```

Реализуется метод вычисления значения функции через линейную интерполяцию: проверяется принадлежность x области определения, находится окружающий отрезок и вычисляется значение по формуле линейной интерполяции между соседними точками.

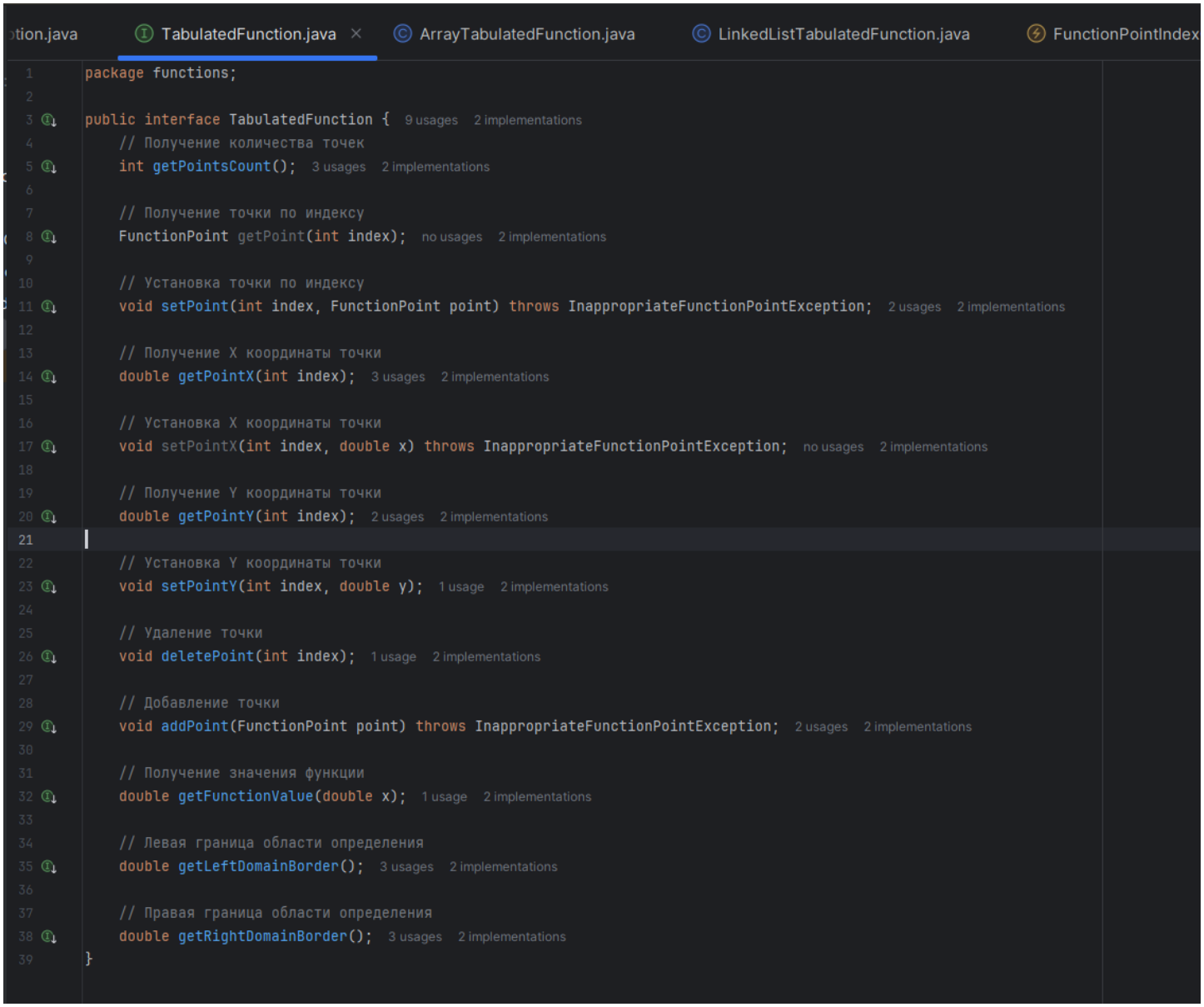
```java
public double getLeftDomainBorder() {  3 usages
    return head.next.point.getX();
}

public double getRightDomainBorder() {  3 usages
    return head.prev.point.getX();
}
```

Реализуются методы получения границ области определения: левая граница - X первой точки списка, правая граница - X последней точки списка.

```java
public int getPointsCount() {  3 usages
    return pointsCount;
}
```

Реализуется метод получения количества точек, возвращающий текущее значение счетчика pointsCount.

Задание 6



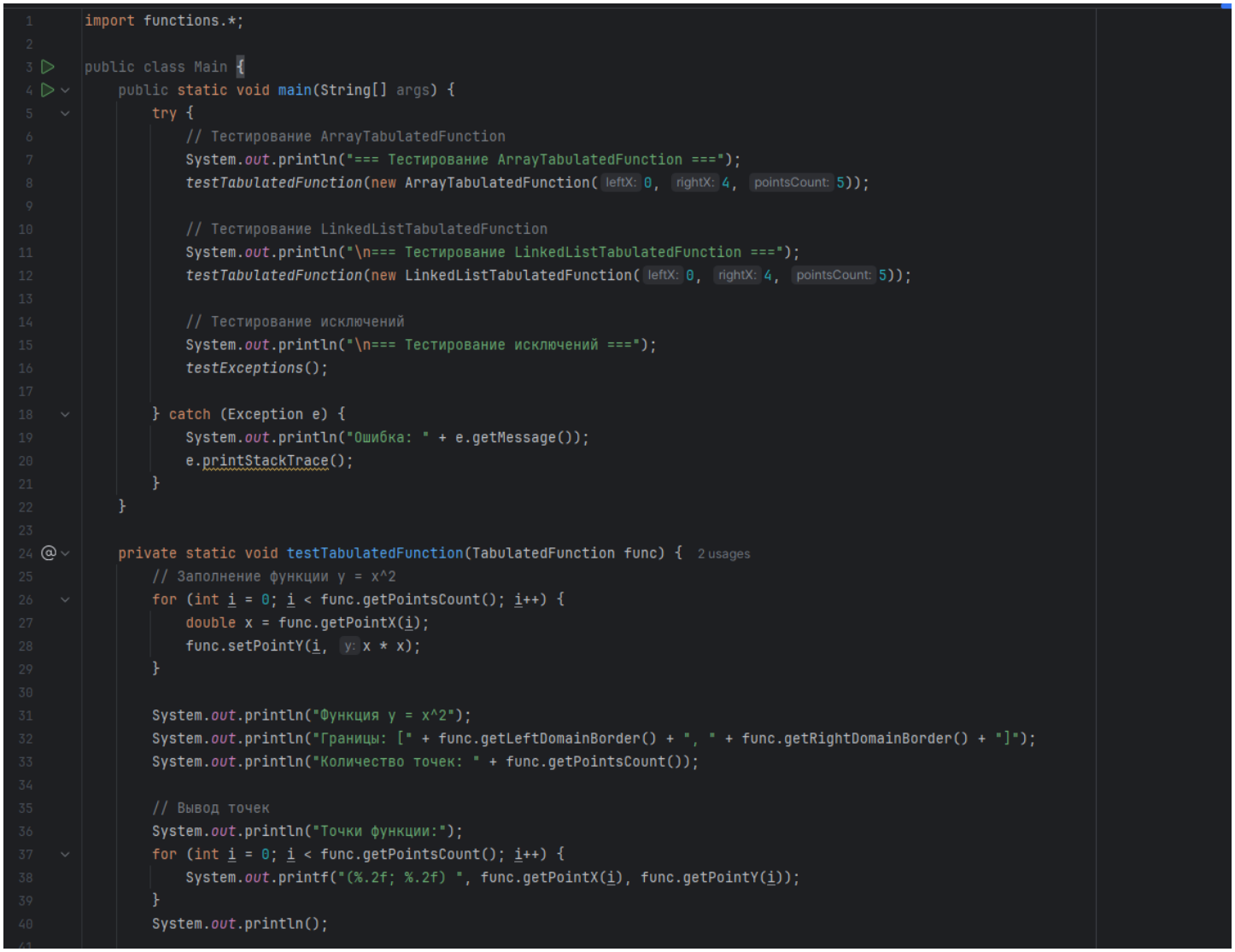
Создан интерфейс TabulatedFunction, определяющий контракт для работы с табулированными функциями. Интерфейс объявляет следующие методы:

- getPoint(), setPoint() - получение и модификация точек
- getPointX(), setPointX(), getPointY(), setPointY() - работа с координатами
- addPoint(), deletePoint() - добавление и удаление точек

- getLeftDomainBorder(), getRightDomainBorder() - границы области определения
- getPointsCount() - количество точек
- getFunctionValue() - вычисление значения функции»

Задание 7

Проверяем



```
import functions.*;

public class Main {
    public static void main(String[] args) {
        try {
            // Тестирование ArrayTabulatedFunction
            System.out.println("=== Тестирование ArrayTabulatedFunction ===");
            testTabulatedFunction(new ArrayTabulatedFunction( leftX: 0,  rightX: 4,  pointsCount: 5));

            // Тестирование LinkedListTabulatedFunction
            System.out.println("\n=== Тестирование LinkedListTabulatedFunction ===");
            testTabulatedFunction(new LinkedListTabulatedFunction( leftX: 0,  rightX: 4,  pointsCount: 5));

            // 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 func) {  2 usages
        // Заполнение функции y = x^2
        for (int i = 0; i < func.getPointsCount(); i++) {
            double x = func.getPointX(i);
            func.setPointY(i,  y: x * x);
        }

        System.out.println("Функция y = x^2");
        System.out.println("Границы: [" + func.getLeftDomainBorder() + ", " + func.getRightDomainBorder() + "]");
        System.out.println("Количество точек: " + func.getPointsCount());

        // Вывод точек
        System.out.println("Точки функции:");
        for (int i = 0; i < func.getPointsCount(); i++) {
            System.out.printf("(%.2f; %.2f) ", func.getPointX(i), func.getPointY(i));
        }
        System.out.println();
```

```
        // Тестирование значений функции
        System.out.println("Значения функции:");
        for (double x = -1; x <= 5; x += 1.0) {
            System.out.printf("f(%.1f) = %.2f\n", x, func.getFunctionValue(x));
        }

        // Тестирование добавления точки
        try {
            func.addPoint(new FunctionPoint( x: 2.5,  y: 6.25));
            System.out.println("Добавлена точка (2.5; 6.25)");
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Не удалось добавить точку: " + e.getMessage());
        }
    }

    private static void testExceptions() {  1 usage
        try {
            // Некорректные параметры конструктора
            TabulatedFunction func = new ArrayTabulatedFunction( leftX: 5,  rightX: 0,  pointsCount: 3);
        } catch (IllegalArgumentException e) {
            System.out.println("Поймано IllegalArgumentException: " + e.getMessage());
        }

        try {
            TabulatedFunction func = new ArrayTabulatedFunction( leftX: 0,  rightX: 5,  pointsCount: 1);
        } catch (IllegalArgumentException e) {
            System.out.println("Поймано IllegalArgumentException: " + e.getMessage());
        }

        try {
            // Выход за границы массива
            TabulatedFunction func = new ArrayTabulatedFunction( leftX: 0,  rightX: 4,  pointsCount: 3);
            func.getPointX( index: 10);
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Поймано FunctionPointIndexOutOfBoundsException: " + e.getMessage());
        }

        try {
            // Нарушение порядка точек
            TabulatedFunction func = new ArrayTabulatedFunction( leftX: 0,  rightX: 4,  pointsCount: 3);
            func.setPoint( index: 1, new FunctionPoint( x: 3,  y: 9)); // Должно вызвать исключение
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Поймано InappropriateFunctionPointException: " + e.getMessage());
        }

        try {
            // Удаление при недостаточном количестве точек
            TabulatedFunction func = new ArrayTabulatedFunction( leftX: 0,  rightX: 2,  pointsCount: 2);
            func.deletePoint( index: 0);
        } catch (IllegalStateException e) {
            System.out.println("Поймано IllegalStateException: " + e.getMessage());
        }

        try {
            // Добавление точки с существующим X
            TabulatedFunction func = new ArrayTabulatedFunction( leftX: 0,  rightX: 4,  pointsCount: 3);
            func.addPoint(new FunctionPoint( x: 1,  y: 1));
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Поймано InappropriateFunctionPointException: " + e.getMessage());
        }
    }
}
```

```
=== Тестирование ArrayTabulatedFunction ===
Функция y = x^2
Границы: [0.0, 4.0]
Количество точек: 5
Точки функции:
(0,00; 0,00) (1,00; 1,00) (2,00; 4,00) (3,00; 9,00) (4,00; 16,00)
Значения функции:
f(-1,0) = NaN
f(0,0) = 0,00
f(1,0) = 1,00
f(2,0) = 4,00
f(3,0) = 9,00
f(4,0) = 16,00
f(5,0) = NaN
Добавлена точка (2.5; 6.25)

=== Тестирование LinkedListTabulatedFunction ===
Функция y = x^2
Границы: [0.0, 4.0]
Количество точек: 5
Точки функции:
(0,00; 0,00) (1,00; 1,00) (2,00; 4,00) (3,00; 9,00) (4,00; 16,00)
Значения функции:
f(-1,0) = NaN
f(0,0) = 0,00
f(1,0) = 1,00
f(2,0) = 4,00
f(3,0) = 9,00
f(4,0) = 16,00
f(5,0) = NaN
Добавлена точка (2.5; 6.25)

=== Тестирование исключений ===
Поймано IllegalArgumentException: Левая граница должна быть меньше правой
Поймано IllegalArgumentException: Количество точек должно быть не меньше двух
Поймано FunctionPointIndexOutOfBoundsException: Индекс 10 вне диапазона точек [0, 2]
Поймано IllegalStateException: Нельзя удалить точку - останется меньше двух точек

Process finished with exit code 0
```