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 НОМЕР 3

Студент Горейнов Д.В.

Группа 6301-030301D

Проверил Борисов.Д.С

Оценка _____

САМАРА 2025

Задание 2

В пакете functions были созданы два класса исключения:

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

```java
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException
{
    private String message;

    public FunctionPointIndexOutOfBoundsException(String message)
    {
        this.message=message;
    }

    public String getMessage()
    {
        return message;
    }

}
```

Фото 1

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

```java
package functions;

public class InappropriateFunctionPointException extends Exception
{
    private String message;

    public InappropriateFunctionPointException(String message)
    {
        this.message=message;
    }

    public String getMessage()
    {
        return message;
    }

}
```

Фото 2

Наследование от класса Exception реализуется с помощью extends.

Задание 3

В разработанный ранее класс TabulatedFunction внес 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.

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

```java
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
    if(leftX >= rightX)
    {
        throw new IllegalArgumentException("the left border is larger than the right one");
    }

    if(pointsCount < 2)
    {
        throw new IllegalArgumentException("Number of points is less than 2");
    }

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

}

public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
    if(leftX >= rightX)
    {
        throw new IllegalArgumentException("the left border is larger than the right one");
    }

    if(values.length < 2)
    {
        throw new IllegalArgumentException("Number of points is less than 2");
    }

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

```java
public FunctionPoint getPoint(int index) {
    if(index<0||index>=pointsCount)
    {
        throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
    }
    return points[index];
}

public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
    double leftX=getLeftDomainBorder();
    double rightX=getRightDomainBorder();
    if (index < 0 || index >= pointsCount)
    {
        throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
    }
    if(leftX > point.getX() || rightX < point.getX())
    {
        throw new InappropriateFunctionPointException(message: "x out of boreder");
    }
    points[index] = new FunctionPoint(point);
}

public double getPointX(int index) {

    if (index < 0 || index >= pointsCount)
    {
        throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");

    }
        if (index < 0 || index >= pointsCount)
        {
            return Double.NaN;
        }
    return points[index].getX();
}
```

```java
public void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
    double leftX=getLeftDomainBorder();
    double rightX=getRightDomainBorder();
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
    }



    if (x < leftX || x > rightX) {
        throw new InappropriateFunctionPointException(message: "x out of border");
    }

    double epsilon = 1e-9;
    for (int i = 0; i < pointsCount; i++) {
        if (i != index && Math.abs(points[i].getX() - x) < epsilon) {
            throw new InappropriateFunctionPointException(message: "Точка с таким x уже существует");
        }
    }

    points[index].setX(x);
}

    public double getPointY(int index) {
        if (index < 0 || index > pointsCount)
        {
            throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");

        }
         if (index < 0 || index >= pointsCount)
        {
        return Double.NaN;
        }

        return points[index].getY();
    }

    public void setPointY(int index, double y)
    {
        if (index < 0 || index > pointsCount)
        {
            throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
        }
        points[index].setY(y);
    }
```

```java
public void deletePoint(int index){
  if (index < 0 || index >= pointsCount)
      {
          throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
      }
  if(pointsCount<3)
  {
      throw new IllegalStateException("The number of points is less than 3");
  }

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


      }
```

```
public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
    double epsilon = 1e-9;


    for (int i = 0; i < pointsCount; i++) {
        if (Math.abs(points[i].getX() - point.getX()) < epsilon) {
            throw new InappropriateFunctionPointException(message: "There is already such an x");
        }
    }


    int insertIndex = 0;
    while (insertIndex < pointsCount && point.getX() > points[insertIndex].getX()) {
        insertIndex++;
    }


    FunctionPoint[] newPoints = new FunctionPoint[pointsCount + 1];
    System.arraycopy(points, 0, newPoints, 0, insertIndex);
    newPoints[insertIndex] = new FunctionPoint(point);
    System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, pointsCount - insertIndex);

    points = newPoints;
    pointsCount++;
}

}
```

Фото 3-7

Задание 4

В пакете functions создал класс LinkedListTabulatedFunction, объект которого описывает табулированную функцию. Отличие этого класса заключаться в том, что для хранения набора точек в нем должен использоваться не массив, а динамическая структура – связный список.

Описал класс элементов списка FunctionNode, содержащий информационное поле для хранения данных типа FunctionPoint, а также поля для хранения ссылок на предыдущий и следующий элемент.

Обоснование:

1)Защитное копирование в методах getPoint() и setPoint()

2)Приватный внутренний класс FunctionNode

3)Контролируемый доступ к полям prev и next

4)Безопасные конструкторы с защитным копированием

```
package functions;

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

    public FunctionNode()
    {
        next=null;
        prev=null;
        point = new FunctionPoint();

    }

    public FunctionNode(FunctionNode prev,FunctionNode next)
    {
        point = new FunctionPoint();
        this.prev = prev;
        this.next = next;

    }

    // Безопасные геттеры
    public FunctionPoint getPoint() {
        return new FunctionPoint(point);
    }


    public void setPoint(FunctionPoint point) {
        this.point = new FunctionPoint(point);
    }

    FunctionNode getPrev() {
        return prev;
    }

    FunctionNode getNext() {
        return next;
    }

    void setPrev(FunctionNode prev) {
        this.prev = prev;
    }

    void setNext(FunctionNode next) {
        this.next = next;
    }
}
private FunctionNode head;
```

Фото 8

Методы getPrev и getNext предназначены только для внутреннего использования, благодаря чему другие классы не могут видеть эти методы

Описал класс LinkedListTabulatedFunction объектов списка, содержащий поле ссылки на объект головы, а также иные вспомогательные поля.

```java
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.pointsCount = 0; // сначала 0
    if (leftX >= rightX) throw new IllegalArgumentException("the left border is larger than the right one");
    if (pointsCount < 2) throw new IllegalArgumentException("Number of points is less than 2");

    head = new FunctionNode();
    head.setNext(head);
    head.setPrev(head); // создаем циклический список

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

    public LinkedListTabulatedFunction(double leftX, double rightX, double values[]) {
        if (leftX >= rightX) throw new IllegalArgumentException("the left border is larger than the right one");
        if (values.length < 2) throw new IllegalArgumentException("Number of points is less than 2");
        head = new FunctionNode(head, head);
        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            FunctionPoint point = new FunctionPoint(leftX + i * step, values[i]);
            addNodeToTail().setPoint(point);
        }
    }
```

Фото 9

В классе LinkedListTabulatedFunction реализовал метод FunctionNode getNodeByIndex(int index), возвращающий ссылку на объект элемента списка по его номеру. Метод обеспечивает оптимизацию доступа к элементам списка. Оптимизация реализована следующим образом:

Если указанный элемент находится в первой половине списка, ход ведётся в прямом направлении. Если элемент находится во второй половине, ход ведётся в обратном направлении, так как мы используем связный список и каждый элемент хранит ссылку на следующий и предыдущий элемент

```java
FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException(message: "Invalid index");

    FunctionNode current;
    if(index < pointsCount/2) {
        current = head.getNext();
        for(int i = 0; i < index; i++) {
            current = current.next;
        }
        return current;
    } else {
        current = head.getPrev();
        for(int i = pointsCount - 1; i > index; i--) {
            current = current.prev;
        }
        return current;
    }
}
```

Фото 10

В классе LinkedListTabulatedFunction реализовал метод FunctionNode addNodeToTail(), добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента

```
private FunctionNode addNodeToTail() {
        FunctionNode newNode = new FunctionNode();


        if (pointsCount == 0) {
            newNode.setNext(head);
            newNode.setPrev(head);
            head.setNext(newNode);
            head.setPrev(newNode);
        } else {

            FunctionNode tail = head.getPrev();
            newNode.setPrev(tail);
            newNode.setNext(head);
            tail.setNext(newNode);
            head.setPrev(newNode);
        }

        pointsCount++;
        return newNode;
    }
```

Фото 11

В классе LinkedListTabulatedFunction реализовал метод FunctionNode addNodeByIndex(int index), добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента.

```
FunctionNode addNodeByIndex(int index)
{
    FunctionNode Node = getNodeByIndex(index);
    FunctionNode prevNode = Node.getPrev();
    FunctionNode newNode=new FunctionNode();
    newNode.setNext(Node);
    newNode.setPrev(prevNode);
    Node.setPrev(newNode);
    prevNode.setNext(newNode);
    pointsCount++;
    return newNode;
}
```

Фото 12

В классе LinkedListTabulatedFunction реализовать метод FunctionNode deleteNodeByIndex(int index), удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента.

```
FunctionNode deleteNodeByIndex(int index)
{
    FunctionNode Node = getNodeByIndex(index);
    FunctionNode nextNode = Node.getNext();
    FunctionNode prevNode = Node.getPrev();
    Node.setNext(next: null);
    Node.setPrev(prev: null);
    pointsCount--;
    prevNode.setNext(nextNode);
    nextNode.setPrev(prevNode);
    return Node;

}
```

Фото 13

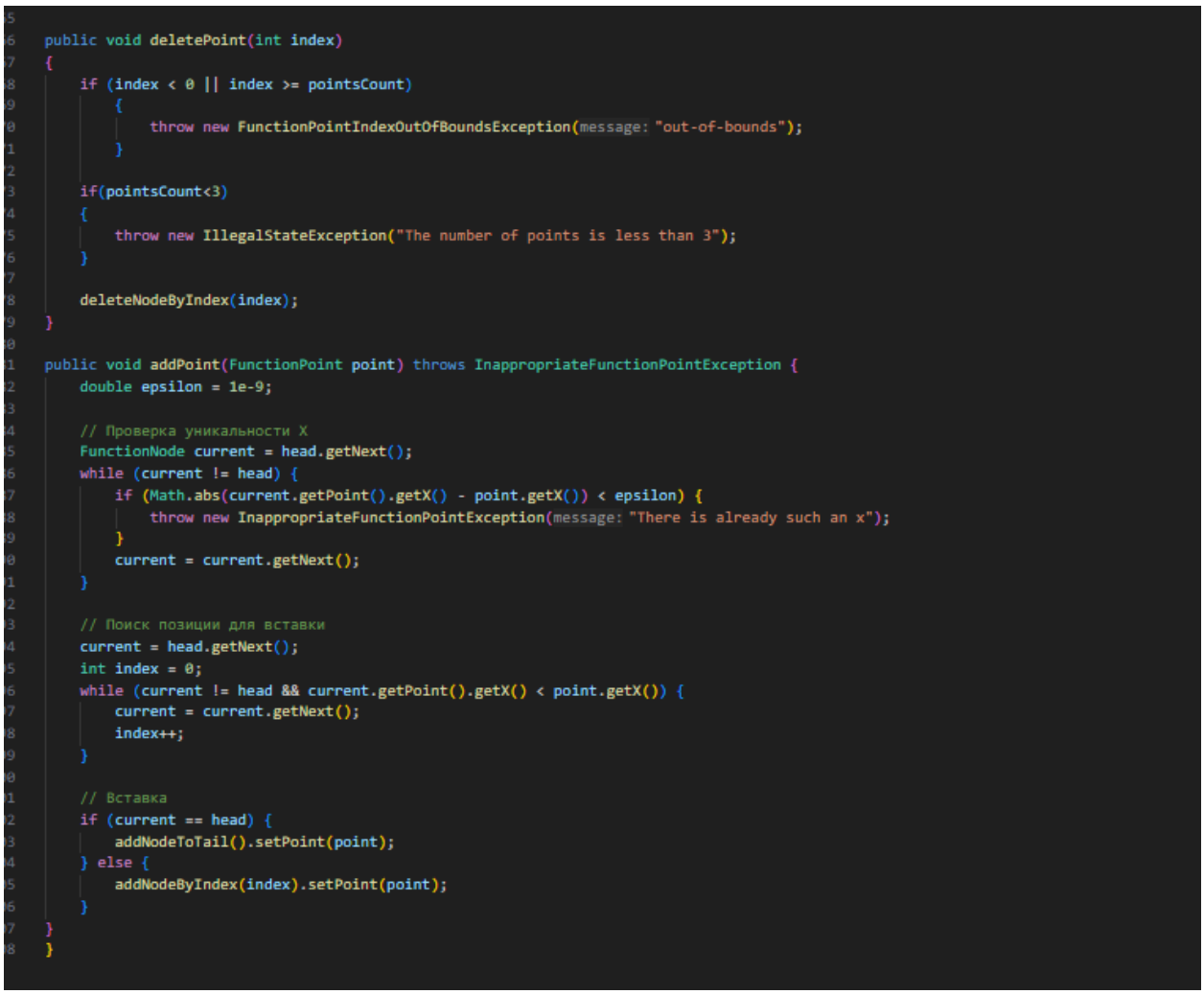
Задание 5

Для обеспечения второй функции
класса LinkedListTabulatedFunction реализовал в классе конструкторы и методы, аналогичные конструкторам и методам класса TabulatedFunction. Конструкторы должны имеют те же параметры, методы имеют те же сигнатуры. Выбрасываются те же виды исключений в аналогичных случаях.

```
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
    this.pointsCount = 0; // сначала 0
    if (leftX >= rightX) throw new IllegalArgumentException("the left border is larger than the right one");
    if (pointsCount < 2) throw new IllegalArgumentException("Number of points is less than 2");

    head = new FunctionNode();
    head.setNext(head);
    head.setPrev(head); // создаем циклический список

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

    public LinkedListTabulatedFunction(double leftX, double rightX, double values[]) {
        if (leftX >= rightX) throw new IllegalArgumentException("the left border is larger than the right one");
        if (values.length < 2) throw new IllegalArgumentException("Number of points is less than 2");
        head = new FunctionNode(head, head);
        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            FunctionPoint point = new FunctionPoint(leftX + i * step, values[i]);
            addNodeToTail().setPoint(point);
        }
    }

    public double getLeftDomainBorder()
    {
        return head.getNext().getPoint().getX();
    }

    public double getRightDomainBorder()
    {
        return head.getPrev().getPoint().getX();
    }
     public int getPointsCount()
    {
        return this.pointsCount;
    }
```

```java
public double getFunctionValue(double x) {
    double epsilon = 1e-9;
    double leftX = getLeftDomainBorder();
    double rightX = getRightDomainBorder();

    if (x < leftX || x > rightX) return Double.NaN;

    FunctionNode current = head.getNext();


    if (Math.abs(x - leftX) < epsilon)
    {
        return current.getPoint().getY();
    }

    if (Math.abs(x - rightX) < epsilon)
    {
        return head.getPrev().getPoint().getY();
    }

    while (current != head && current.getNext() != head) {
        double x1 = current.getPoint().getX();
        double x2 = current.getNext().getPoint().getX();

        if (x >= x1 && x <= x2) {
            double y1 = current.getPoint().getY();
            double y2 = current.getNext().getPoint().getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
        current = current.getNext();
    }

    return Double.NaN;
}

FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException(message: "Invalid index");

    FunctionNode current;
    if(index < pointsCount/2) {
        current = head.getNext();
        for(int i = 0; i < index; i++) {
            current = current.next;
        }
        return current;
    } else {
        current = head.getPrev();
        for(int i = pointsCount - 1; i > index; i--) {
            current = current.prev;
        }
        return current;
    }
}
```

```
public void setPoint(int index,FunctionPoint point) throws InappropriateFunctionPointException
{
    double leftX=getLeftDomainBorder();
    double rightX=getRightDomainBorder();
    if (index < 0 || index >= pointsCount)
        {
            throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
        }
        if(leftX > point.getX() || rightX < point.getX())
        {
            throw new InappropriateFunctionPointException(message: "x out of boreder");
        }
        FunctionNode node = getNodeByIndex(index);
        node.setPoint(point);
    }

public double getPointX(int index)
{
    if (0 > index || index >= pointsCount) throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
    return getPoint(index).getX();
}

public void setPointX(int index, double x) throws InappropriateFunctionPointException
{
    double leftX=getLeftDomainBorder();
    double rightX=getRightDomainBorder();
    if (index < 0 || index >= pointsCount)
        {
            throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
        }
        if(leftX > x || rightX < x)
        {
            throw new InappropriateFunctionPointException(message: "x out of boreder");
        }
        FunctionNode node = getNodeByIndex(index);
        node.getPoint().setX(x);

}
public double getPointY(int index)
{
    if (index < 0 || index >= pointsCount)
    {
        throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
    }
    FunctionNode node = getNodeByIndex(index);
    return node.getPoint().getY();
}

public void setPointY(int index, double y)
{
    if (index < 0 || index >= pointsCount)
        {
            throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
        }

    FunctionNode node = getNodeByIndex(index);
    node.getPoint().setY(y);
}
```



```
public void deletePoint(int index)
{
    if (index < 0 || index >= pointsCount)
        {
            throw new FunctionPointIndexOutOfBoundsException(message: "out-of-bounds");
        }

    if(pointsCount<3)
    {
        throw new IllegalStateException("The number of points is less than 3");
    }

    deleteNodeByIndex(index);
}

public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
    double epsilon = 1e-9;

    // Проверка уникальности X
    FunctionNode current = head.getNext();
    while (current != head) {
        if (Math.abs(current.getPoint().getX() - point.getX()) < epsilon) {
            throw new InappropriateFunctionPointException(message: "There is already such an x");
        }
        current = current.getNext();
    }

    // Поиск позиции для вставки
    current = head.getNext();
    int index = 0;
    while (current != head && current.getPoint().getX() < point.getX()) {
        current = current.getNext();
        index++;
    }

    // Вставка
    if (current == head) {
        addNodeToTail().setPoint(point);
    } else {
        addNodeByIndex(index).setPoint(point);
    }
}
}
```

Фото14-17

Оптимизировал работу getFunctionValue за счет возможности обращаться к элементам списка напрямую.

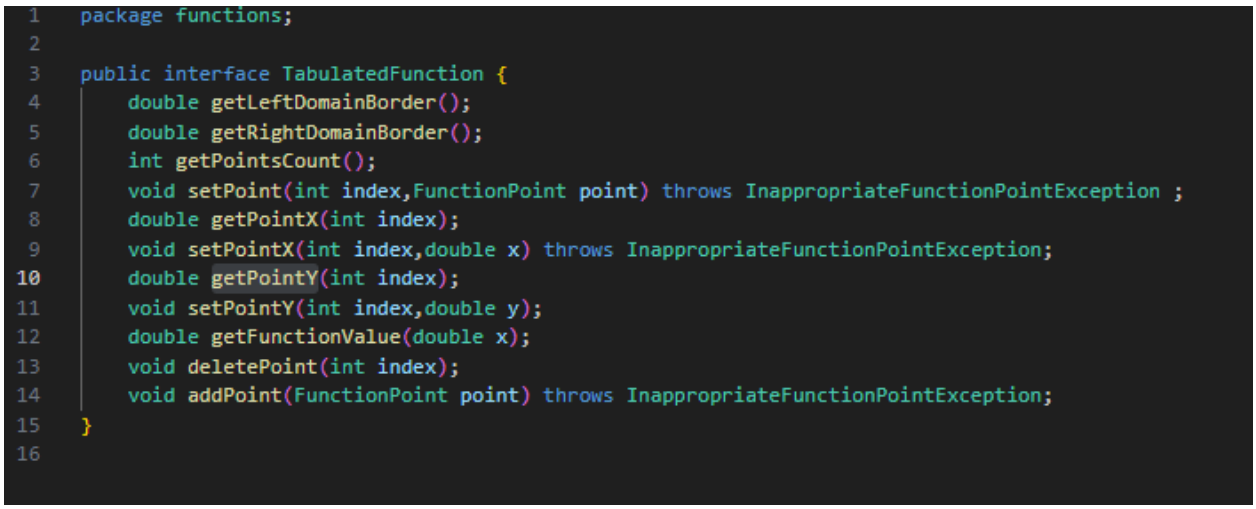
```java
public double getFunctionValue(double x) {
    double epsilon = 1e-9;
    double leftX = getLeftDomainBorder();
    double rightX = getRightDomainBorder();

    if (x < leftX || x > rightX) return Double.NaN;

    FunctionNode current = head.getNext();


    if (Math.abs(x - leftX) < epsilon)
    {
        return current.getPoint().getY();
    }

    if (Math.abs(x - rightX) < epsilon)
    {
        return head.getPrev().getPoint().getY();
    }

    while (current != head && current.getNext() != head) {
        double x1 = current.getPoint().getX();
        double x2 = current.getNext().getPoint().getX();

        if (x >= x1 && x <= x2) {
            double y1 = current.getPoint().getY();
            double y2 = current.getNext().getPoint().getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
        current = current.getNext();
    }

    return Double.NaN;
}
```

Фото 18

Задание 6

Класс TabulatedFunction переименовал в класс ArrayTabulatedFunction.

Создал интерфейс TabulatedFunction, содержащий объявления общих методов классов ArrayTabulatedFunction и LinkedListTabulatedFunction.



```java
package functions;

public interface TabulatedFunction {
    double getLeftDomainBorder();
    double getRightDomainBorder();
    int getPointsCount();
    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);
    double getFunctionValue(double x);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}

```

Фото 19

Сделал так, чтобы оба класса функций реализовывали созданный интерфейс.

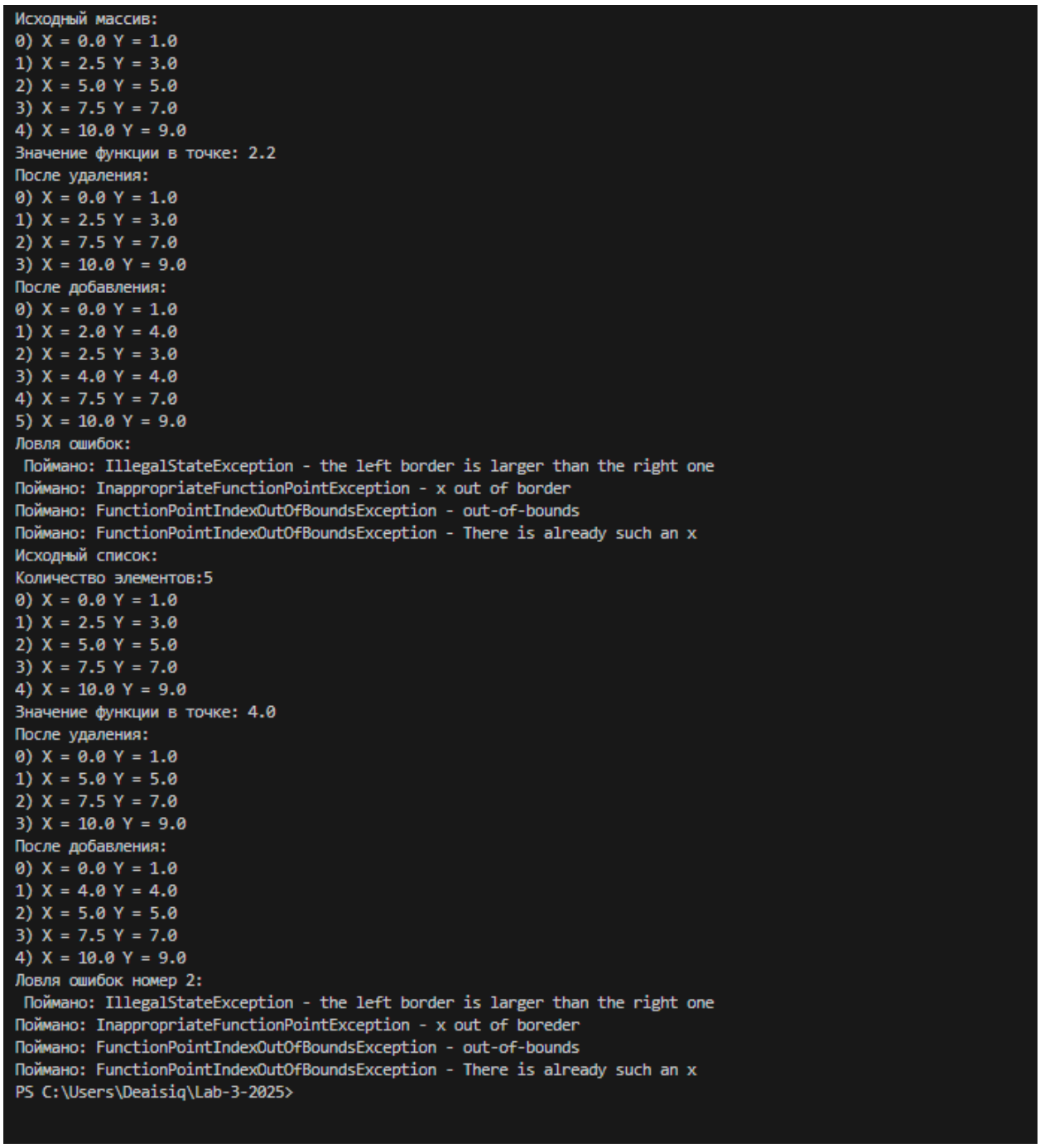

Исходный массив:
0) X = 0.0 Y = 1.0
1) X = 2.5 Y = 3.0
2) X = 5.0 Y = 5.0
3) X = 7.5 Y = 7.0
4) X = 10.0 Y = 9.0
Значение функции в точке: 2.2
После удаления:
0) X = 0.0 Y = 1.0
1) X = 2.5 Y = 3.0
2) X = 7.5 Y = 7.0
3) X = 10.0 Y = 9.0
После добавления:
0) X = 0.0 Y = 1.0
1) X = 2.0 Y = 4.0
2) X = 2.5 Y = 3.0
3) X = 4.0 Y = 4.0
4) X = 7.5 Y = 7.0
5) X = 10.0 Y = 9.0
Ловля ошибок:
Поймано: IllegalStateException - the left border is larger than the right one
Поймано: InappropriateFunctionPointException - x out of border
Поймано: FunctionPointIndexOutOfBoundsException - out-of-bounds
Поймано: FunctionPointIndexOutOfBoundsException - There is already such an x
Исходный список:
Количество элементов:5
0) X = 0.0 Y = 1.0
1) X = 2.5 Y = 3.0
2) X = 5.0 Y = 5.0
3) X = 7.5 Y = 7.0
4) X = 10.0 Y = 9.0
Значение функции в точке: 4.0
После удаления:
0) X = 0.0 Y = 1.0
1) X = 5.0 Y = 5.0
2) X = 7.5 Y = 7.0
3) X = 10.0 Y = 9.0
После добавления:
0) X = 0.0 Y = 1.0
1) X = 4.0 Y = 4.0
2) X = 5.0 Y = 5.0
3) X = 7.5 Y = 7.0
4) X = 10.0 Y = 9.0
Ловля ошибок номер 2:
Поймано: IllegalStateException - the left border is larger than the right one
Поймано: InappropriateFunctionPointException - x out of boreder
Поймано: FunctionPointIndexOutOfBoundsException - out-of-bounds
Поймано: FunctionPointIndexOutOfBoundsException - There is already such an x
PS C:\Users\Deaisiq\Lab-3-2025>

Фото 20