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 №3

Студент Волков Д.Н.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Задание №1

1) java.lang.Exception - является корневым классом для всех некритических ошибок, которые программа может (и должна) попытаться обработать.

2) java.lang.IndexOutOfBoundsException – исключение происходит тогда, когда программа пыталась получить доступ к элементу по индексу, который выходит за допустимые границы

3) java.lang.ArrayIndexOutOfBoundsException – частный случай для второго исключения, когда работа идет с массивом, то есть была попытка получить элемент по индексу, который выходит за границы массива.

4) java.lang.IllegalArgumentException - Указывает на то, что метод получил аргумент, который является недопустимым, даже если он синтаксически правилен. Это значит, что значение аргумента не соответствует требованиям, предъявляемым методом.

5) java.lang.IllegalStateException - Указывает на то, что метод был вызван в неподходящее время, или что объект находится в недопустимом состоянии для выполнения данного метода. Операция не может быть выполнена из-за текущего состояния объекта.

Задание №2

Создал два класса необходимые для обработки исключений.

Исключение необходимое для обработки индексов, выходящих за пределы массива или списка. При использовании try-catch в Main через getMessage() позволит вывести причину выбрасывания исключения.

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException {

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

Исключения для остальных случаев, когда программа может некорректно работать и с помощью try-catch в Main через getMessage() позволит вывести причину выбрасывания исключения.

```java
package functions;

public class InappropriateFunctionPointException extends Exception {

    public InappropriateFunctionPointException() {
        super();
    }

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

Задание №3

В каждом геттер и сеттер методе были добавлены выбрасывания исключения FunctionPointIndexOutOfBoundsException для случаев, когда индекс выходит за границы массива.

Также исключения InappropriateFunctionPointException, IllegalStateException необходимые для правильной работы программы в TabulatedFunction, deletePoint, addPoint (Например случай, когда leftX будет больше или равен rightX в конструкторах ArrayTabulatedFunction).

```java
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
    if (leftX >= rightX) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    if (pointsCount < 3) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    this.pointsCount = pointsCount;
```

```java
*/
public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
    if (leftX >= rightX) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    if (pointsCount < 3) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
```

```java
public FunctionPoint getPoint(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Получить точку с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    return new FunctionPoint(points[index].getX(), points[index].getY());
}
/*
```

```java
public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Задать точку с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    if (index == 0) { // Проверка, если пользователь пытается изменить левую границу
        if (point.getX() > getRightDomainBorder()) {
            throw new InappropriateFunctionPointException("Новая точка X (" + point.getX() + ") выходит за границы (больше правой границы).");
        }
    } else if (index == pointsCount - 1) {
        if (point.getX() < getLeftDomainBorder())  { // Проверка, если пользователь пытается изменить правую границу
            throw new InappropriateFunctionPointException("Новая точка X (" + point.getX() + ") выходит за границы (меньше левой границы).");
        }
    }
    else {
        if (point.getX() < points[index - 1].getX() || point.getX() > points[index + 1].getX()) { // Условия проверяют координату X у point на то, что она не выходит за значения соседних X
            throw new InappropriateFunctionPointException("Новая точка X (" + point.getX() + ") выходит за границы соседних к ней точек.");
        }
    }
```

```java
public double getPointX(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Получить координату X с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    else{
        return points[index].getX();
    }
}
```

```java
public void setPointX(int index, double x) throws InappropriateFunctionPointException {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Задать координату X с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    if (x < points[index - 1].getX() || x > points[index + 1].getX()) {
        throw new InappropriateFunctionPointException("Новая точка X (" + x + ") выходит за границы соседних к ней точек.");
    }
```

```java
public double getPointY(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Получить координату Y с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    else{
        return points[index].getY();
    }
}
```

```java
public void setPointY(int index, double y) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Задать координату Y с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    else{
        points[index].setY(y);
    }
}
```

```java
public void deletePoint(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Удалить точку с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    if (pointsCount < 3) {
        throw new IllegalStateException(s: "В массиве меньше 3 точек");
    }
```

```java
while (point.getX() > points[i].getX()) ++i;
if(Math.abs(points[i].getX() - point.getX()) < EPSILON_DOUBLE) {
    throw new InappropriateFunctionPointException(message: "Координата X добавляемой точки совпадает с уже сужествующим X ");
}
if(pointsCount >= points.length){
```

Задание №4

Для работы со связным списком был создан класс, имеющий в себе конструктор, позволяющий создать новый узел в списке, методы позволяющие получить или изменить точку в узле, получить или изменить ссылку на предыдущий узел и получить или изменить ссылку на следующий узел.

```
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction{

    private class FunctionNode {
        private FunctionPoint point;
        private FunctionNode prev;
        private FunctionNode next;

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

        public FunctionPoint getPoint() {
            return point;
        }

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

        public FunctionNode getPrev() {
            return prev;
        }

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

        public FunctionNode getNext() {
            return next;
        }

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

Метод позволяющий получить точку находящуюся на переданном индексе, для оптимизации поиска, метод находит к какой границе ближе находится точка и находит ее смещением ссылки.

```
private FunctionNode head;
private int pointsCount;
private final double EPSILON_DOUBLE = 1e-9;

private FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс" + index + " лежит вне границ массива" + pointsCount);
    }

    FunctionNode current;

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

    return current;
}
```

Метод позволяющий добавить элемент в конец списка

```
private FunctionNode addNodeToTail(FunctionPoint point) {
    FunctionNode newNode = new FunctionNode(point, head.getPrev(), head);
    FunctionNode tail = head.getPrev();
    tail.setNext(newNode);
    head.setPrev(newNode);
    return newNode;
}
```

Метод позволяющий добавить элемент на переданный индекс.

```
private FunctionNode addNodeByIndex(int index, FunctionPoint point) {
    if (index < 0 || index > pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " лежит вне границ массива " + pointsCount);
    }
    if (index == pointsCount) {
        pointsCount++;
        return addNodeToTail(point);
    }
    FunctionNode nextNode = getNodeByIndex(index);
    FunctionNode prevNode = nextNode.getPrev();
    FunctionNode newNode = new FunctionNode(point, prevNode, nextNode);
    prevNode.setNext(newNode);
    nextNode.setPrev(newNode);
    pointsCount++;
    return newNode;
}
```

И метод позволяющий удалить точку по переданному индексу.

```
private FunctionNode deleteNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " лежит вне границ массива " + pointsCount);
    }
    FunctionNode nodeToDelete = getNodeByIndex(index);
    FunctionNode prevNode = nodeToDelete.getPrev();
    FunctionNode nextNode = nodeToDelete.getNext();
    prevNode.setNext(nextNode);
    nextNode.setPrev(prevNode);
    pointsCount--;
    return nodeToDelete;
}
```

Два конструктора создающих список в двух случаях, когда задано количество точек или массив точек Y.

```
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount){
    if (leftX >= rightX) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    if (pointsCount < 3) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    this.pointsCount = pointsCount;
    head = new FunctionNode(point: null, prev: null, next: null);
    head.setNext(head);
    head.setPrev(head);
    double step = (rightX - leftX)/(pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        addNodeToTail(new FunctionPoint(leftX + step * i, y: 0));
    }
}

public LinkedListTabulatedFunction(double leftX, double rightX, double[] values){
    if (leftX >= rightX) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    if (pointsCount < 3) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    this.pointsCount = values.length;
    head = new FunctionNode(point: null, prev: null, next: null);
    head.setNext(head);
    head.setPrev(head);
    double step = (rightX - leftX)/(pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        addNodeToTail(new FunctionPoint(leftX + step * i,  values[i]));
    }
}
```

Методы позволяющие получить левую и правую границы списка

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

@Override
public double getRightDomainBorder() {
    return head.getPrev().getPoint().getX();
}
```

Метод позволяющий найти координату Y с помощью линейной интерполяции для переданного X.

```
public double getFunctionValue(double x) {
    if (x <  getLeftDomainBorder()|| x > getRightDomainBorder()) {
        return Double.NaN;
    }
    if (Math.abs(x - getLeftDomainBorder()) < EPSILON_DOUBLE) {
        return getLeftDomainBorder();
    }
    if (Math.abs(x  - getRightDomainBorder()) < EPSILON_DOUBLE) {
        return getRightDomainBorder();
    }
    else {
        int i;
        double value = 0;
        for (i = 0; x > getNodeByIndex(i).getPoint().getX(); i++){
            value = getNodeByIndex(i).getPoint().getY()+(getNodeByIndex(i+1).getPoint().getY() - getNodeByIndex(i).getPoint().getY())*(x-getNodeByIndex(i).getPoint().getX())
            /(getNodeByIndex(i+1).getPoint().getX()- getNodeByIndex(i).getPoint().getX());
        }
        return value;
    }
}
```

Методы для получения количества точек в списке и для получения и изменения точки в списке

```
@Override
public int getPointsCount() {
    return pointsCount;
}

@Override
public FunctionPoint getPoint(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Получить точку с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    return new FunctionPoint(getNodeByIndex(index).getPoint().getX(), getNodeByIndex(index).getPoint().getY());
}

@Override
public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Задать точку с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    FunctionNode currentNode = getNodeByIndex(index);
    if (index == 0) {
        if (point.getX() > getRightDomainBorder()) {
            throw new InappropriateFunctionPointException("Новая точка X (" + point.getX() + ") выходит за границы (больше правой границы).");
        }
    } else if (index == pointsCount - 1) {
        if (point.getX() < getLeftDomainBorder()) {
            throw new InappropriateFunctionPointException("Новая точка X (" + point.getX() + ") выходит за границы (меньше левой границы).");
        }
    }
    else {
        if (point.getX() < getNodeByIndex(index-1).getPoint().getX() || point.getX() > getNodeByIndex(index+1).getPoint().getX()) {
            throw new InappropriateFunctionPointException("Новая точка X (" + point.getX() + ") выходит за границы соседних к ней точек.");
        }
    }
    currentNode.setPoint(point);
}
```

Геттер и сеттер методы для точек X и Y.

```java
@Override
public double getPointX(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Получить координату X с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    else{
        return getNodeByIndex(index).getPoint().getX();
    }
}

@Override
public void setPointX(int index, double x) throws InappropriateFunctionPointException {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Задать координату X с индексом " + index + " невозможно, так как размер массива" + this.pointsCount);
    }
    if (x < getNodeByIndex(index-1).getPoint().getX() || x > getNodeByIndex(index+1).getPoint().getX()) {
        throw new InappropriateFunctionPointException("Новая точка X (" + x + ") выходит за границы соседних к ней точек.");
    }
    FunctionNode currentNode = getNodeByIndex(index);
    currentNode.getPoint().setX(x);
}

@Override
public double getPointY(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Получить координату Y с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    else{
        return  getNodeByIndex(index).getPoint().getY();
    }
}

@Override
public void setPointY(int index, double y) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Задать координату Y с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    else{
        FunctionNode currentNode = getNodeByIndex(index);
        currentNode.getPoint().setY(y);
    }
}
```

Методы для удаления и добавления точек в список

```java
@Override
public void deletePoint(int index){
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Удалить точку с индексом " + index + " невозможно, так как размер массива " + this.pointsCount);
    }
    if (pointsCount < 3) {
        throw new IllegalStateException(s: "В массиве меньше 3 точек");
    }
    deleteNodeByIndex(index);
}

@Override
public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
    if (point.getX() > getNodeByIndex(pointsCount - 1).getPoint().getX()){
        addNodeByIndex(pointsCount, point);
    }
    else{
        int i = 0;
        while (point.getX() > getNodeByIndex(i).getPoint().getX()) ++i;
        if (Math.abs(point.getX() - getNodeByIndex(i).getPoint().getX()) < EPSILON_DOUBLE) {
            throw new InappropriateFunctionPointException("Координата X добавляемой точки совпадает с уже сужествующим X = " + getNodeByIndex(i).getPoint().getX());
        }
        else{
            addNodeByIndex(i, point);
        }
    }
}
```

В методы(и конструкторы), как и в ArrayTabulatedFunction добавлены выбрасывания исключений для необходимых случаев.

Задание №6

Переименовал класс TabulatedFunction в ArrayTabulatedFunction, создал интерфейс, благодаря которому программа сама понимает, с каким классом необходимо работать ArrayTabulatedFunction или LinkedListTabuladedFunction.

```
package functions;

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

Задание №7

В main для всех методов, которые требуют проверки исключений try-catch добавил ее, это addPoint и setPoint

```
System.out.println(data.length);
System.out.println(x: "\nДобавление точек ");
try {
    Function.addPoint(PointToAdd1);
    Function.addPoint(PointToAdd2);
    Function.addPoint(PointToAdd3);
} catch (InappropriateFunctionPointException mes) {
    System.out.println("\n Ошибка при добавлении точек: "+ mes.getMessage());
}
```

```
try {
    Function.setPointX(testIndex, x: 16);
}   catch (InappropriateFunctionPointException mes) {
    System.out.println("\nОшибка при измненении точки X :"+ mes.getMessage());
}
```

```
System.out.println("\nТочка X был успешно изменена: X = " + Function.getPointX(testIndex
try {
    Function.setPointX(testIndex, afterSetPointX);
} catch(InappropriateFunctionPointException mes){
    System.out.println("\nОшибка при изменении точки X : "+ mes.getMessage());
}
```

```
try {
    Function.setPoint(testIndex, testPoint);
} catch (InappropriateFunctionPointException mes) {
    System.out.println("\nОшибка при измненени точки :"+ mes.getMessage());
}
```

```
try {
    Function.setPoint(testIndex, afterSetPoint);
} catch (InappropriateFunctionPointException mes) {
    System.out.println("\nОшибка при измненении точки: "+ mes.getMessage());
}
```

Также добавил реализацию для списка

```
TabulatedFunction ListFunction = new LinkedListTabulatedFunction(leftX: 1, rightX: 50, data);

System.out.println(x: "Исходная матрица:");
for (int i = 0; i < ListFunction.getPointsCount(); i++) {
    System.out.println("X[" + i + "] = " + ListFunction.getPointX(i) + " and Y[" + i + "] = " + ListFunction.getPointY(i));
}

System.out.println(x: "\nУдаление точек с индексами 0, 4, 6");
if (ListFunction.getPointsCount() > 0) {
    ListFunction.deletePoint(index: 4);
    ListFunction.deletePoint(index: 2);
    ListFunction.deletePoint(index: 0);
}
else {
    System.out.println(x: "Нет точек для удаления");
}

System.out.println(x: "\nМатрица после удаления точек:");
for (int i = 0; i < ListFunction.getPointsCount(); i++) {
    System.out.println("X[" + i + "] = " + ListFunction.getPointX(i) + " and Y[" + i + "] = " + ListFunction.getPointY(i));
}

System.out.println(x: "\nДобавление точек ");
try {
    ListFunction.addPoint(PointToAdd1);
    ListFunction.addPoint(PointToAdd2);
    ListFunction.addPoint(PointToAdd3);
} catch (InappropriateFunctionPointException mes) {
    System.out.println("\n Ошибка при добавлении точек: "+ mes.getMessage());
}
System.out.println(ListFunction.getPointsCount());
System.out.println(x: "\nМатрица после добавления точек: ");
for (int i = 0; i < ListFunction.getPointsCount(); i++) {
    System.out.println("X[" + i + "] = " + ListFunction.getPointX(i) + " and Y[" + i + "] = " + ListFunction.getPointY(i));
}
System.out.println(x: "\nРасчет значения Y с помощью линейной интерполяции");
double ListValueY = ListFunction.getFunctionValue(ValueX);
System.out.println("Значение функции X = " + ValueX + " Y = " + ListValueY);

//Проверка геттер и сеттер методов
```

Результат работы программы

```
Исходная матрица:
X[0] = 1.0 and Y[0] = 1.0
X[1] = 6.444444444444445 and Y[1] = 3.7
X[2] = 11.88888888888889 and Y[2] = 6.9
X[3] = 17.333333333333336 and Y[3] = 8.0
X[4] = 22.77777777777778 and Y[4] = 4.0
X[5] = 28.22222222222222 and Y[5] = 2.0
X[6] = 33.66666666666667 and Y[6] = 20.0
X[7] = 39.111111111111114 and Y[7] = 15.5
X[8] = 44.55555555555556 and Y[8] = 73.0
X[9] = 50.0 and Y[9] = 100.0

Удаление точек с индексами 0, 4, 6

Матрица после удаления точек:
X[0] = 6.444444444444445 and Y[0] = 3.7
X[1] = 17.333333333333336 and Y[1] = 8.0
X[2] = 28.22222222222222 and Y[2] = 2.0
X[3] = 33.66666666666667 and Y[3] = 20.0
X[4] = 39.111111111111114 and Y[4] = 15.5
X[5] = 44.55555555555556 and Y[5] = 73.0
X[6] = 50.0 and Y[6] = 100.0
10

Добавление точек

Матрица после добавления точек:
X[0] = 3.5 and Y[0] = 5.0
X[1] = 6.444444444444445 and Y[1] = 3.7
X[2] = 15.0 and Y[2] = 6.0
X[3] = 17.333333333333336 and Y[3] = 8.0
X[4] = 28.22222222222222 and Y[4] = 2.0
X[5] = 33.66666666666667 and Y[5] = 20.0
X[6] = 39.111111111111114 and Y[6] = 15.5
X[7] = 44.55555555555556 and Y[7] = 73.0
X[8] = 50.0 and Y[8] = 100.0
X[9] = 101.0 and Y[9] = 20.5

Расчет значения Y с помощью линейной интерполяции
Значение функции X = 7.0 Y = 3.8493506493506495
Использование LinkenListTabulatedFunction:
Исходная матрица:
X[0] = 1.0 and Y[0] = 1.0
X[1] = 6.444444444444445 and Y[1] = 3.7
X[2] = 11.88888888888889 and Y[2] = 6.9
X[3] = 17.333333333333336 and Y[3] = 8.0
X[4] = 22.77777777777778 and Y[4] = 4.0
X[5] = 28.22222222222222 and Y[5] = 2.0
X[6] = 33.66666666666667 and Y[6] = 20.0
X[7] = 39.111111111111114 and Y[7] = 15.5
X[8] = 44.55555555555556 and Y[8] = 73.0
X[9] = 50.0 and Y[9] = 100.0

Удаление точек с индексами 0, 4, 6

Матрица после удаления точек:
X[0] = 6.444444444444445 and Y[0] = 3.7
X[1] = 17.333333333333336 and Y[1] = 8.0
X[2] = 28.22222222222222 and Y[2] = 2.0
X[3] = 33.66666666666667 and Y[3] = 20.0
X[4] = 39.111111111111114 and Y[4] = 15.5
X[5] = 44.55555555555556 and Y[5] = 73.0
X[6] = 50.0 and Y[6] = 100.0
```

```
Добавление точек
10

Матрица после добавления точек:
X[0] = 3.5 and Y[0] = 5.0
X[1] = 6.444444444444445 and Y[1] = 3.7
X[2] = 15.0 and Y[2] = 6.0
X[3] = 17.333333333333336 and Y[3] = 8.0
X[4] = 28.22222222222222 and Y[4] = 2.0
X[5] = 33.66666666666667 and Y[5] = 20.0
X[6] = 39.111111111111114 and Y[6] = 15.5
X[7] = 44.55555555555556 and Y[7] = 73.0
X[8] = 50.0 and Y[8] = 100.0
X[9] = 101.0 and Y[9] = 20.5

Расчет значения Y с помощью линейной интерполяции
Значение функции X = 7.0 Y = 3.8493506493506495

              Проверка get() и set() методов

Точка X был успешно изменена: X = 16.0

Возращено начальное значение X: 17.333333333333336

Точка X был успешно изменена: Y = 6.8

Возращено начальное значение Y: 8.0

Точка была успешно изменена: X = 16.0 Y = 10.0

Возращено начальное значение точки X = 17.333333333333336 Y = 8.0
PS C:\Users\LuckyFrut\Documents\University\Lab-3-2025>
```