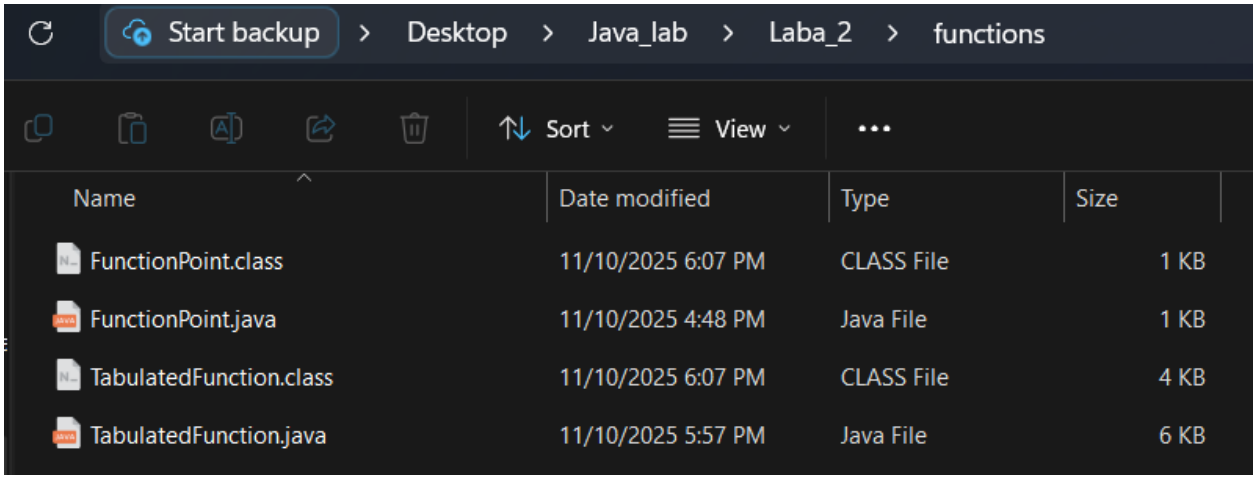
Лабораторная работа 2

Бавтута Дмитрий Валерьевич

6203-010302D

Task 1

Класс functions для хранения классов программы



Task 2

```
package functions;


public class FunctionPoint{

    private double x;
    private double y;

    public FunctionPoint( double x, double y){
        this.x = x;
        this.y = y;
    }

    public FunctionPoint(FunctionPoint point){
        this.x = point.x;
        this.y = point.y;
    }

    public FunctionPoint(){
        this.x = 0;
        this.y = 0;
```

```
    }

    // Геттеры и сеттеры для чтения и изменения данных

    public double getX(){
        return x;
    }

    public double getY(){
        return y;
    }

    public void setX(double x){
        this.x = x;
    }

    public void setY(double y){
        this.y = y;
    }

    @Override
public String toString() {
    // Для обычных чисел используем стандартное форматирование
    // Для специальных значений (Infinity, NaN, очень большие/малые) - прямое преобразование
    if (Double.isInfinite(x) || Double.isInfinite(y) ||
        Double.isNaN(x) || Double.isNaN(y) ||
        Math.abs(x) > 1e10 || Math.abs(x) < 1e-10 ||
        Math.abs(y) > 1e10 || Math.abs(y) < 1e-10) {

        // Для специальных и экстремальных значений
        return "(" + x + ", " + y + ")";
    } else {
        // Для обычных чисел - красивое форматирование
        return String.format("(%.2f, %.2f)", x, y);
```

```
    }
}
}
```

Task 3

Для хранения данных о точках использует массив типа FunctionPoint.

В классе описаны следующие конструкторы:

TabulatedFunction(double leftX, double rightX, int pointsCount) и

TabulatedFunction(double leftX, double rightX, double[] values)

Первый создаёт объект табулированной функции по заданным левой и правой границе области определения, значения функции равны нулю, а также количеству точек для табулирования, а второй аналогично первому но получает вместо количества точек массив со значениями функции для разных точек,

```
package functions;


public class TabulatedFunction{
    private FunctionPoint[] points;
    private int pointsCount;


    public TabulatedFunction(double leftX, double rightX, int pointsCount){


        this.pointsCount = pointsCount;
        this.points = new FunctionPoint[pointsCount];


        double step = (rightX - leftX) / (pointsCount - 1);


        for(int i = 0; i < pointsCount; ++i){
            double x = leftX + i*step;
            points[i] = new FunctionPoint(x, 0.0);
        }
    }
```

```
public TabulatedFunction(double leftX, double rightX, double[] values){

    this.pointsCount = values.length;
    this.points = new FunctionPoint[pointsCount];

    double step = (rightX - leftX) / (pointsCount - 1);

    for (int i = 0; i < pointsCount; i++){
        double x = leftX + i * step;
        points[i] = new FunctionPoint(x, values[i]);
    }
}
```

Task 4

Также описаны методы для работы с функцией такие как:

double getLeftDomainBorder()

double getRightDomainBorder()

Они нужны чтобы возвращать значения левой и правой границы функции.

double getFunctionValue(double x) – Возвращает значение функции в точке x, если эта точка лежит в области определения функции.

```
public double getLeftDomainBorder(){
    return points[0].getX();
}

public double getRightDomainBorder(){
    return points[pointsCount - 1].getX();
}

public double getFunctionValue(double x){
```

```
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
        return Double.NaN;
    }

    for (int i = 0; i < pointsCount; i++){

        double x1 = points[i].getX();
        double x2 = points[i + 1].getX();

        if (x >= x1 && x <= x2){
            double y1 = points[i].getY();
            double y2 = points[i + 1].getY();



            return y1 + (y2 - y1) * (x - x1) / (x2 -x1);
        }
    }
    return 0;
}
```

Task 5

Содержит методы: для вывода количества точек - int getPointsCount(), геттеры и сеттеры которые выводят и изменяют как точку целиком так и отдельно разные координаты x или y - FunctionPoint getPoint(int index), void setPoint(int index, FunctionPoint point), double getPointX(int index), void setPointX(int index, double x), double getPointY(int index), void setPointY(int index, double y).

Для метода setpoint разделил его работу на три отдельных метода – setFirstPoint, setLustPoint, setMiddlePoint, для улучшения читаемости. Также для метода setPointX.

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

public FunctionPoint getPoint(int index){
    return new FunctionPoint(points[index]);
```

```
}

public void setFirstPoint(FunctionPoint point){

   if(point.getX() < points[1].getX()){
      points[0] = new FunctionPoint(point);
   } else {
      throw new IllegalArgumentException("First point must be left of second point");
   }
}

public void setLastPoint(FunctionPoint point){

   if(point.getX() > points[pointsCount - 2].getX()){
      points[pointsCount - 1] = new FunctionPoint(point);
   } else {
      throw new IllegalArgumentException(" Lust point must be left of previous point");
   }
}

public void setMiddlePoint(int index,  FunctionPoint point){

   double x1 = points[index - 1].getX();
   double x2 = points[index + 1].getX();
   double x = point.getX();

      if( x1 < x && x < x2){
         points[index] = new FunctionPoint(point);
      } else {
         throw new IllegalArgumentException(" Middle point must be betwen " + x1 + " and " + x2);
      }
}
```

```
public void setPoint(int index, FunctionPoint point){

    if(index == 0){
        setFirstPoint(point);
    }

    else if(index == pointsCount - 1){
        setLastPoint(point);
    }

    else{
        setMiddlePoint(index, point);
    }
}

public double getPointX(int index){
    if(0 <= index && index <= pointsCount - 1){
        return points[index].getX();
    } else {
        throw new IllegalArgumentException(" Point must be betwen " + 0 + " and lust element");
    }
}

public void setFirstPointX(double x){
    if(x < points[1].getX()){
        points[0].setX(x);
    } else {
        throw new IllegalArgumentException("First point must be left of second point");
    }
}

public void setLustPointX(double x){
    if(x > points[pointsCount - 2].getX()){
        points[pointsCount - 1].setX(x);
```

```
    } else {
        throw new IllegalArgumentException(" Lust point must be left of previous point");
    }
}

public void setMiddlePointX(int index, double x){
    if(x > points[index - 1].getX() && x < points[index + 1].getX()){
        points[index].setX(x);
    }
}

public void setPointX(int index, double x){
    if(index == 0){
        setFirstPointX(x);
    } else if(index == pointsCount - 1){
        setLustPointX(x);
    } else{
        setMiddlePointX(index, x);
    }
}



public double getPointY(int index){
    if(0 <= index && index <= pointsCount - 1){
        return points[index].getY();
    } else {
        throw new IllegalArgumentException(" Point must be betwen " + 0 + " and lust element");
    }
}

public void setPointY(int index, double y){
    if(0 <= index && index <= pointsCount - 1){
        points[index].setY(y);
    } else {
```

```
            throw new IllegalArgumentException(" Point must be betwen " + 0 + " and lust element");
        }
    }
```

Task 6

Методы void deletePoint(int index) и void addPoint(FunctionPoint point) нужны для изменения количества точек функции. Метод void addPoint(FunctionPoint point) для копирования участков массива использует arraycopy()

```
    public void addPoint(FunctionPoint point) {
        if (pointsCount == points.length) {
            FunctionPoint[] newPoints = new FunctionPoint[points.length + points.length / 2 + 1];
            System.arraycopy(points, 0, newPoints, 0, pointsCount);
            points = newPoints;
        }

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

        if (insertIndex < pointsCount && point.getX() == points[insertIndex].getX()) {
            return;
        }

        System.arraycopy(points, insertIndex, points, insertIndex + 1, pointsCount - insertIndex);

        points[insertIndex] = new FunctionPoint(point);
        pointsCount++;
    }

    public void deletePoint(int index){
        System.arraycopy(points, index + 1, points, index, pointsCount - index - 1);
        pointsCount--;
```

```
    }


}
```

Task 7

В пакете по умолчанию (вне пакета functions) создан класс Main, содержащий точку входа программы.

Создается экземпляр класса FunctionPoint, а также TabulatedFunction для него заданы значения квадратичной функции для точек (0, 2, 4, 6, 8, 10).
Проверяются все методы предназначенные для работы с экземплярами этих типов.

```
import functions.*;


public class Main {


    public static void main(String[] args) {
        System.out.println(" ТЕСТИРОВАНИЕ КЛАССА FunctionPoint \n");


        // 1. Тестирование конструкторов
        testConstructors();


        // 2. Тестирование геттеров и сеттеров
        testGettersAndSetters();


        // 3. Тестирование методов в различных сценариях
        testMethods();


        // 4. Тестирование с TabulatedFunction
        testWithTabulatedFunction();


        System.out.println("\n ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ ");
    }


    public static void testConstructors() {
        System.out.println("1. ТЕСТИРОВАНИЕ КОНСТРУКТОРОВ:");
```

```
    // Конструктор с параметрами
    FunctionPoint point1 = new FunctionPoint(3.5, 7.2);
    System.out.println(" FunctionPoint(3.5, 7.2) = " + point1);

    // Конструктор копирования
    FunctionPoint point2 = new FunctionPoint(point1);
    System.out.println(" FunctionPoint(copy) = " + point2);

    // Конструктор по умолчанию
    FunctionPoint point3 = new FunctionPoint();
    System.out.println(" FunctionPoint() = " + point3);

    // Проверка, что копия независима от оригинала
    point1.setX(10.0);
    System.out.println(" После изменения оригинала, копия не изменилась: " + point2);

    System.out.println();
}

public static void testGettersAndSetters() {
    System.out.println("2. ТЕСТИРОВАНИЕ ГЕТТЕРОВ И СЕТТЕРОВ:");

    FunctionPoint point = new FunctionPoint(2.0, 4.0);

    // Тестирование геттеров
    System.out.println(" getX() = " + point.getX());
    System.out.println(" getY() = " + point.getY());

    // Тестирование сеттеров
    point.setX(5.0);
    point.setY(25.0);
    System.out.println(" После setX(5.0) и setY(25.0) = " + point);
```

```
    // Тестирование с отрицательными значениями
    point.setX(-3.0);
    point.setY(-9.0);
    System.out.println(" С отрицательными значениями = " + point);

    // Тестирование с нулевыми значениями
    point.setX(0.0);
    point.setY(0.0);
    System.out.println(" С нулевыми значениями = " + point);

    // Тестирование с дробными значениями
    point.setX(2.75);
    point.setY(3.14159);
    System.out.println(" С дробными значениями = " + point);

    System.out.println();
}

public static void testMethods() {
    System.out.println("3. ТЕСТИРОВАНИЕ МЕТОДОВ В РАЗЛИЧНЫХ СЦЕНАРИЯХ:");

    // Создание нескольких точек для сравнения
    FunctionPoint pointA = new FunctionPoint(1.0, 1.0);
    FunctionPoint pointB = new FunctionPoint(1.0, 1.0); // Такая же как A
    FunctionPoint pointC = new FunctionPoint(2.0, 4.0); // Другая
    FunctionPoint pointD = new FunctionPoint(pointA);   // Копия A

    System.out.println(" pointA = " + pointA);
    System.out.println(" pointB = " + pointB + " (такая же как A)");
    System.out.println(" pointC = " + pointC + " (другая)");
    System.out.println(" pointD = " + pointD + " (копия A)");

    // Тестирование работы с массивом точек
    FunctionPoint[] pointsArray = {
```

```
        new FunctionPoint(0, 0),
        new FunctionPoint(1, 1),
        new FunctionPoint(2, 4),
        new FunctionPoint(3, 9)
    };

    System.out.println("\nМассив точек:");
    for (int i = 0; i < pointsArray.length; i++) {
        System.out.println("  points[" + i + "] = " + pointsArray[i]);
    }

    // Изменение точек в массиве
    pointsArray[1].setX(1.5);
    pointsArray[1].setY(2.25);
    System.out.println(" После изменения points[1] = " + pointsArray[1]);

    System.out.println();
}

public static void testWithTabulatedFunction() {
    System.out.println("4. ТЕСТИРОВАНИЕ TabulatedFunction:");

    // Создание объекта типа TabulatedFunction
    TabulatedFunction func = new TabulatedFunction(0.0, 10.0, 6);

    // Заполнение функции с использованием FunctionPoint
    for (int i = 0; i < func.getPointsCount(); i++) {
        double x = func.getPointX(i);
        FunctionPoint point = new FunctionPoint(x, x * x); // f(x) = x²
        func.setPoint(i, point);
    }

    System.out.println("Массив типа квадратичной функции f(x) = x²:");
    for (int i = 0; i < func.getPointsCount(); i++) {
```

```
    FunctionPoint point = func.getPoint(i);
    System.out.println("  " + point);
}

// Тестирование добавления новых точек
System.out.println("\nДобавление новых точек:");

FunctionPoint newPoint1 = new FunctionPoint(1.5, 2.25);
FunctionPoint newPoint2 = new FunctionPoint(3.5, 12.25);
FunctionPoint newPoint3 = new FunctionPoint(7.5, 56.25);

func.addPoint(newPoint1);
func.addPoint(newPoint2);
func.addPoint(newPoint3);

System.out.println("После добавления 3 точек:");
for (int i = 0; i < func.getPointsCount(); i++) {
    System.out.println("  " + func.getPoint(i));
}

// Тестирование изменения существующей точки
System.out.println("\nИзменение точки с индексом 2:");
FunctionPoint modifiedPoint = new FunctionPoint(2.0, 8.0); // Было (2.0, 4.0)
func.setPoint(2, modifiedPoint);
System.out.println("  Новая точка: " + func.getPoint(2));

// Тестирование граничных значений
System.out.println("\nТестирование граничных значений:");
FunctionPoint edgePoint1 = new FunctionPoint(func.getLeftDomainBorder(), 0);
FunctionPoint edgePoint2 = new FunctionPoint(func.getRightDomainBorder(), 100);

System.out.println("  Левая граница: " + edgePoint1);
System.out.println("  Правая граница: " + edgePoint2);
```

```
        // Тестирование специальных значений
        System.out.println("\nТестирование специальных значений:");
        FunctionPoint specialPoint1 = new FunctionPoint(Double.MAX_VALUE, Double.MAX_VALUE);
        FunctionPoint specialPoint2 = new FunctionPoint(Double.MIN_VALUE, Double.MIN_VALUE);

        System.out.println("  MAX_VALUE: " + specialPoint1);
        System.out.println("  MIN_VALUE: " + specialPoint2);

        System.out.println();
    }
}
```