Лабораторная работа №2

Коновалов Сергей Сергеевич 6204-010302D

**Задание 1**

Создал пакет functions, в котором будут создаваться классы программы.

**Задание 2**

В пакете functions создал класс FunctionPoint, объект которого описывает одну точку табулированной функции

Класс реализует принципы инкапсуляции - поля x и y объявлены как private, доступ к ним осуществляется через геттеры и сеттеры

Конструкторы класса:

- FunctionPoint(double x, double y) - создаёт объект точки с заданными координатами:

public FunctionPoint(double x, double y) {

this.x = x;

this.y = y;

}

- FunctionPoint(FunctionPoint point) - создаёт объект точки с теми же координатами, что у указанной точки (конструктор копирования):

public FunctionPoint(FunctionPoint point) {

this.x = point.x;

this.y = point.y;

}

- FunctionPoint() - создаёт точку с координатами (0; 0):  
public FunctionPoint() {

this(0.0, 0.0);

}

**Задание 3**

В том же пакете создал класс TabulatedFunction, объект которого описывает табулированную функцию

Для хранения данных использую массив FunctionPoint[] points, точки всегда упорядочены по x

Конструкторы класса:

- TabulatedFunction(double leftX, double rightX, int pointsCount) - создаёт табулированную функцию с заданными границами и количеством точек при y = 0 :

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);

}

}

- TabulatedFunction(double leftX, double rightX, double[] values) - создаёт функцию с заданными границами и значениями y из массива :

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]);

}

}

Точки создаются через равные интервалы по x

**Задание 4**

В TabulatedFunction реализовал методы для работы с функцией:

- double getLeftDomainBorder() - возвращает левую границу :

public double getLeftDomainBorder() {

return points[0].getX();

}

- double getRightDomainBorder() - возвращает правую границу :

public double getRightDomainBorder() {

return points[pointsCount - 1].getX();

}

- double getFunctionValue(double x) - возвращает значение функции в точке x с использованием линейной интерполяции, или Double.NaN, если x вне области определения :

public double getFunctionValue(double x) {

if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {

return Double.NaN;

}

for (int i = 0; i < pointsCount - 1; 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;

}

**Задание 5**

Реализовал методы для работы с точками табулированной функции, согласно заданию:

- int getPointsCount() - возвращает количество точек :

public int getPointsCount() {

return pointsCount;

}

- FunctionPoint getPoint(int index) - возвращает копию точки по индексу :

public FunctionPoint getPoint(int index) {

return new FunctionPoint(points[index]);

}

- void setPoint(int index, FunctionPoint point) - заменяет точку на копию переданной точки с проверкой корректности x :

public void setPoint(int index, FunctionPoint point) {

double newX = point.getX();

if (index == 0) {

if (newX >= points[1].getX()) {

return;

}

}

else if (index == pointsCount - 1) {

if (newX <= points[pointsCount - 2].getX()) {

return;

}

}

else {

if (newX <= points[index - 1].getX() || newX >= points[index + 1].getX()) {

return;

}

}

points[index] = new FunctionPoint(point);

}

- double getPointX(int index), double getPointY(int index) - возвращают координаты точки :

public double getPointX(int index) {

return points[index].getX();

}

public double getPointY(int index) {

return points[index].getY();

}

- void setPointX(int index, double x), void setPointY(int index, double y) - изменяют координаты точки с проверкой корректности :

public void setPointX(int index, double x) {

if (index == 0) {

if (x >= points[1].getX()) {

return;

}

}

else if (index == pointsCount - 1) {

if (x <= points[pointsCount - 2].getX()) {

return;

}

}

else {

if (x <= points[index - 1].getX() || x >= points[index + 1].getX()) {

return;

}

}

points[index].setX(x);

}

public void setPointY(int index, double y) {

if (index == 0) {

if (y >= points[1].getY()) {

return;

}

}

else if (index == pointsCount - 1) {

if (y <= points[pointsCount - 2].getY()) {

return;

}

}

else {

if (y <= points[index - 1].getY() || y >= points[index + 1].getY()) {

return;

}

}

points[index].setY(y);

}

**Задание 6**

По требованию заданию, реализовал методы для изменения количества точек:

- void deletePoint(int index) - удаляет точку по индексу:

public void deletePoint(int index) {

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

pointsCount--;

}

- void addPoint(FunctionPoint point) - добавляет новую точку с сохранением упорядоченности по x:

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++;

}

**Задание 7**

Создал класс Main вне functions для тестирования написанных классов:

public class Main {

public static void main(String[] args) {

double[] val = {0.0, 1.0, 4.0, 9.0, 16.0};

TabulatedFunction function = new TabulatedFunction(0.0, 4.0, val);

System.out.println("- Изначальная функция -");

printFunctionInfo(function);

System.out.println("\n- Вычисление значений функции -");

double[] Points = {-1.0, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0};

for (double x : Points) {

double y = function.getFunctionValue(x);

System.out.printf("f(%.1f) = %s%n", x, Double.isNaN(y) ? "не определено" : String.format("%.2f", y));

}

System.out.println("\n- Изменение точек -");

function.setPointY(2, 5.0);

System.out.println("После изменения: ");

printFunctionInfo(function);

System.out.println("\n- Добавление точки -");

FunctionPoint newPoint = new FunctionPoint(1.5, 2.25);

function.addPoint(newPoint);

System.out.println("После добавления:");

printFunctionInfo(function);

System.out.println("\n- Удаление точки -");

function.deletePoint(2); // Удаляем точку с индексом 2

System.out.println("После удаления:");

printFunctionInfo(function);

System.out.println("\n- Значения после всех изменений -");

for (double x : Points) {

double y = function.getFunctionValue(x);

System.out.printf("f(%.1f) = %s%n", x, Double.isNaN(y) ? "не определено" : String.format("%.2f", y));

}

System.out.println("\n- Границы -");

System.out.println("Левая граница: " + function.getLeftDomainBorder());

System.out.println("Правая граница: " + function.getRightDomainBorder());

System.out.println("Количество точек: " + function.getPointsCount());

}

private static void printFunctionInfo(TabulatedFunction function) {

System.out.println("Точки функции:");

for (int i = 0; i < function.getPointsCount(); i++) {

double x = function.getPointX(i);

double y = function.getPointY(i);

System.out.printf(" %d %.1f %.1f %n", i, x, y);

}

}

}

Класс Main проверяет:

* Создание табулированной функции квадрата на интервале [0, 4]
* Выведение значений функции в различных точках, включая точки вне области определения
* Работу линейной интерполяции
* Изменение, добавление и удаление точек
* Корректную работу граничных условий и проверок