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 №7

Студент Кольчугина Е. А.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Содержание

Задание 1

Итератор (Iterator) в Java — это объект, который позволяет последовательно перебирать элементы коллекции, не раскрывая её внутреннюю структуру. Использует методы hasNext() для проверки наличия следующего элемента и next() для его получения.

В интерфейс TabulatedFunction добавлен родительский тип Iterable<FunctionPoint>

```
package functions;
import java.io.Serializable;

public interface TabulatedFunction extends Function, Serializable, Cloneable, Iterable<FunctionPoint> {

    int getPointsCount();

    FunctionPoint getPoint(int index);

    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;

    double getPointX(int index);
```

В ArrayTabulatedFunction и LinkedListTabulatedFunction:

Метод Iterator<FunctionPoint> iterator() возвращает объект итератора:

- В методе hasNext() проверяется наличие последующего элемента
- В методе next() возвращается копия точки для сохранения инкапсуляции.
- При попытке вызова функции удаления выходит ошибка о том, что операция не поддерживается

```
@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private int currentIndex = 0;

        @Override
        public boolean hasNext() {
            return currentIndex < pointsCount;
        }

        @Override
        public FunctionPoint next() {
            if(!hasNext()) {
                throw new NoSuchElementException (s: "В табулированной функции кончились точки");
            }
            return new FunctionPoint(points[currentIndex++]);
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException(message: "Операция удаления не поддерживается");
        }
```

```java
@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private FunctionNode currentNode = head.getNext();

        @Override
        public boolean hasNext() {
            return currentNode != head;
        }

        @Override
        public FunctionPoint next() {
            if(!hasNext()) {
                throw new NoSuchElementException (s: "В табулированной функции кончились точки");
            }
            FunctionPoint point = new FunctionPoint(currentNode.getPoint());
            currentNode = currentNode.getNext();
            return point;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException(message: "Операция удаления не поддерживается");
        }
```

В Main.java добавлена проверка цикла for для каждой точки табулированной функции из массива и листа.

```java
import functions.*;

public class Main {
    Run main | Debug main
    public static void main(String[] args) {
        System.out.println(x: "__testing iterator ArrayTabulatedFunction__");
        FunctionPoint[] points1 = {
            new FunctionPoint(x: 1.0, y: 1.0),
            new FunctionPoint(x: 2.0, y: 4.0),
            new FunctionPoint(x: 3.0, y: 9.0)
        };
        TabulatedFunction f_1 = new ArrayTabulatedFunction(points1);
        for (FunctionPoint p : f_1) {
            System.out.println(p);
        }

        System.out.println(x: "__testing iterator LinkedListTabulatedFunction__");
        TabulatedFunction f_2 = new LinkedListTabulatedFunction(points1);
        for (FunctionPoint p : f_2) {
            System.out.println(p);
        }
    }
}
```

```java
import functions.*;
import functions.basic.*;



public class Main {
    Run main | Debug main
    public static void main(String[] args) {
        Function exp = new Exp();
        double leftX = 0.0;
        double rightX = 1.0;
        double step = 0.1;

        double integral = Functions.integrate(exp, leftX, rightX, step);
        double theoreticalIntegral = Math.exp(a: 1.0) - Math.exp(a: 0.0);

        System.out.println("Integral for step " + step + ":" + integral);
        System.out.println("Teoretical integral: " + theoreticalIntegral);
        System.out.println("Difference: " + Math.abs(integral - theoreticalIntegral));

        double delta = 1e-7;
        double currentStep = step;
        boolean fit = false;
        for (int i = 0; i < 100; i++) {
            integral = Functions.integrate(exp, leftX, rightX, currentStep);
            double difference = Math.abs(integral - theoreticalIntegral);
            System.out.printf(format: "\nStep = %e, Integral = %.8f, Difference = %e", currentStep, integral, difference);

            if(difference < delta) {
                System.out.println("\nDelta = 1e-7 with step " + currentStep);
                fit = true;
                break;
            }

            currentStep /= 2.0;

        }

        if (!fit) {
            System.out.println("\nAccuracy isn't achieved with step " + step);
        }

    }
}
```

Задание 2

Реализован базовый интерфейс TabulatedFunctionFactory, в котором описано три метода TabulatedFunction createTabulatedFunction()

```
package functions;

public interface TabulatedFunctionFactory {
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount);

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

    TabulatedFunction createTabulatedFunction(FunctionPoint[] points);
}
```

В ArrayTabulatedFunction и LinkedListTabulatedFunction добавлены вложенные классы фабрик ArrayTabulatedFunctionFactory и LinkedListTabulatedFunctionFactory, порождающие объекты createTabulatedFunction

```
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
        return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
    }

    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }

    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points){
        return new ArrayTabulatedFunction(points);
    }
}
```

В классе TabulatedFunctions:

1. **Добавлено приватное статическое поле фабрики**:
2. **Добавлен метод для замены фабрики**:
4. **Добавлены три фабричных метода**: TabulatedFunction createTabulatedFunction
5. **Изменены существующие методы**:

- tabulate(), inputTabulatedFunction(), readTabulatedFunction() - используют фабрику для создания объектов

```java
package functions;

import java.io.*;


public final class TabulatedFunctions {
    private TabulatedFunctions() {
        throw new RuntimeException(message: "Объекты этого класса нельзя создать");
    }

    private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();

    public static void setTabulatedFunctionFactory(TabulatedFunctionFactory newFactory) {
        if (newFactory == null) {
            throw new IllegalArgumentException(s: "Фабрика не может быть нулевой");
        }
        factory = newFactory;
    }


    public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
        return factory.createTabulatedFunction(leftX, rightX, pointsCount);
    }

    public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return factory.createTabulatedFunction(leftX, rightX, values);
    }

    public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return factory.createTabulatedFunction(points);
    }

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount){
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException(s: "Границы выходят за область определения");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException(s: "Левая граница области определения должна быть меньше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException(s: "Количество точек должно быть >=2");
        }

        double distance = (rightX - leftX) / (pointsCount - 1);
        double[] values = new double[pointsCount];

        for (int i = 0; i < pointsCount; i++){
            double x = leftX + i * distance;
            values[i] = function.getFunctionValue(x);
        }

        return createTabulatedFunction(leftX, rightX, values);

    }
```

```java
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out){
    DataOutputStream dataOut = new DataOutputStream(out);
    try {
        dataOut.writeInt(function.getPointsCount());
        for (int i = 0; i < function.getPointsCount(); i++) {
            dataOut.writeDouble(function.getPointX(i));
            dataOut.writeDouble(function.getPointY(i));
        }

        dataOut.flush();
    } catch (IOException e) {
        throw new RuntimeException (message: "Ошибка при выводе функции", e);
    }
}

public static TabulatedFunction inputTabulatedFunction(InputStream in){
    DataInputStream dataIn = new DataInputStream(in);
    try {
        int pointsCount = dataIn.readInt();
        FunctionPoint[] points = new FunctionPoint[pointsCount];

        for (int i = 0; i < pointsCount; i++) {
            double x = dataIn.readDouble();
            double y = dataIn.readDouble();
            points[i] = new FunctionPoint(x, y);
        }
        return createTabulatedFunction(points);

    } catch (IOException e) {
        throw new RuntimeException (message: "Ошибка при вводе функции", e);
    }
}
public static void writeTabulatedFunction(TabulatedFunction function, Writer out){
    PrintWriter writer = new PrintWriter(out);
    try {
        writer.print(function.getPointsCount());
        writer.print(c: ' ');

        for (int i = 0; i < function.getPointsCount(); i++) {
            writer.print(function.getPointX(i));
            writer.print(c: ' ');
            writer.print(function.getPointY(i));
            if (i < function.getPointsCount() - 1) {
                writer.print(c: ' ');
            }
        }
        writer.flush();
    } catch (Exception e) {
        throw new RuntimeException(message: "Ошибка при записи функции", e);
    }
}
```

```java
public static TabulatedFunction readTabulatedFunction(Reader in) {
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    try {
        tokenizer.resetSyntax();
        tokenizer.wordChars(low: '0', hi: '9');
        tokenizer.wordChars(low: '.', hi: '.');
        tokenizer.wordChars(low: '-', hi: '-');
        tokenizer.wordChars(low: 'e', hi: 'e');
        tokenizer.wordChars(low: 'E', hi: 'E');
        tokenizer.whitespaceChars(low: ' ', hi: ' ');
        tokenizer.whitespaceChars(low: '\t', hi: '\t');
        tokenizer.whitespaceChars(low: '\n', hi: '\n');
        tokenizer.whitespaceChars(low: '\r', hi: '\r');


        if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
            throw new RuntimeException(message: "Отсутствует количество точек");
        }
        int pointsCount = Integer.parseInt(tokenizer.sval);

        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD){
                throw new RuntimeException (message: "Отсутствует координата x");
            }
            double x = Double.parseDouble(tokenizer.sval);
            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD){
                throw new RuntimeException (message: "Отсутствует координата y");
            }
            double y = Double.parseDouble(tokenizer.sval);
            points[i] = new FunctionPoint(x, y);
        }

        return createTabulatedFunction(points);

    } catch (IOException e) {
        throw new RuntimeException(message: "Ошибка при чтении", e);
    } catch(NumberFormatException e) {
        throw new RuntimeException(message: "Неверный формат числа", e);
    }
}
```

В метод main() добавлена проверка работы фабрик

```java
System.out.println(x: "__testing factory__");
Function f = new Cos();
TabulatedFunction tf;

tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
System.out.println("Array factory: " + tf.getClass());

TabulatedFunctions.setTabulatedFunctionFactory(new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
System.out.println("LinkedList factory: " + tf.getClass());

TabulatedFunctions.setTabulatedFunctionFactory(new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
System.out.println("Array factory: " + tf.getClass());
```

Задание 3

Было добавлено три новых статических метода в класс TabulatedFunctions, которые принимают в качестве параметра класс создаваемого объекта:

- public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, int pointsCount);
- public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, double[] values);
- public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, FunctionPoint[] points);

Каждый метод сначала проверяет, что переданный класс реализует интерфейс TabulatedFunction с помощью метода isAssignableFrom(). Используется рефлексия для поиска соответствующего конструктора через getConstructor()

Рефлексия позволяет:

- Узнать/определить класс объекта;
- Получить информацию о модификаторах класса, полях, методах, константах, конструкторах и суперклассах;
- Выяснить, какие методы принадлежат реализуемому интерфейсу/интерфейсам;
- Создать экземпляр класса, причем имя класса неизвестно до момента выполнения программы;
- Получить и установить значение поля объекта по имени;
- Вызвать метод объекта по имени.

Конструкторы вызываются динамически с помощью newInstance(). Все исключения рефлексии оборачиваются в IllegalArgumentException для сохранения информации об исходной причине ошибки

Реализована перегруженная версия метода tabulate()

- Метод выполняет табуляцию функции в заданном диапазоне
- Полученные значения сохраняются в массив
- Создает объект указанного класса с использованием рефлексивного метода createTabulatedFunction()

```java
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, int pointsCount) {
    if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
        throw new IllegalArgumentException (s: "Класс не реализует интерфейс");
    }

    try {
        // Находим конструктор с параметрами (double, double, int)
        Constructor<?> constructor = functionClass.getConstructor(...parameterTypes: double.class, double.class, int.class);
        // Создаем экземпляр
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount); //ссылка на обьект
    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException(message: "Не найден конструктор с параметрами (double, double, int) в классе", e);
    } catch (InstantiationException | IllegalAccessException | InvocationTargetException e){
        throw new IllegalArgumentException(message: "Ошибка при создании экземпляра", e);
    }

}
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, double[] values) {
    if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
        throw new IllegalArgumentException(s: "Класс не реализует интерфейс TabulatedFunction");
    }

    try {
        // Находим конструктор с параметрами (double, double, double[])
        Constructor<?> constructor = functionClass.getConstructor(...parameterTypes: double.class, double.class, double[].class);
        // Создаем экземпляр
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, values);
    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException(message: "Не найден конструктор с параметрами (double, double, double[]) в классе ", e);
    } catch (InstantiationException | IllegalAccessException | InvocationTargetException e) {
        throw new IllegalArgumentException(message: "Ошибка при создании экземпляра", e);
    }
}

public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, FunctionPoint[] points) {
    if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
        throw new IllegalArgumentException(s: "Класс не реализует интерфейс TabulatedFunction");
    }

    try {
        // Находим конструктор с параметрами (FunctionPoint[])
        Constructor<?> constructor = functionClass.getConstructor(...parameterTypes: FunctionPoint[].class);
        // Создаем экземпляр
        return (TabulatedFunction) constructor.newInstance((Object)points);
    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException(message: "Не найден конструктор с параметрами (FunctionPoint[]) в классе ", e);
    } catch (InstantiationException | IllegalAccessException | InvocationTargetException e) {
        throw new IllegalArgumentException(message: "Ошибка при создании экземпляра класса ", e);
    }
}
```

```java
// Перегруженный метод tabulate с использованием рефлексии
public static TabulatedFunction tabulate(Class<?> functionClass, Function function, double leftX, double rightX, int pointsCount) {
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
        throw new IllegalArgumentException(s: "Границы выходят за область определения");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException(s: "Левая граница области определения должна быть меньше правой");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException(s: "Количество точек должно быть >=2");
    }

    double distance = (rightX - leftX) / (pointsCount - 1);
    double[] values = new double[pointsCount];

    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * distance;
        values[i] = function.getFunctionValue(x);
    }
    return createTabulatedFunction(functionClass, leftX, rightX, values);
}
```

Добавлена проверка в Main.java

```java
System.out.println(x: "\n__testing reflection__");

f = TabulatedFunctions.createTabulatedFunction(functionClass: ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3);
System.out.println("ArrayTabulatedFunction (double, double, int): " + f.getClass());
System.out.println(f);

f = TabulatedFunctions.createTabulatedFunction(
functionClass: ArrayTabulatedFunction.class, leftX: 0, rightX: 10, new double[] {0, 10});
System.out.println("\nArrayTabulatedFunction (double, double, double[]): " + f.getClass());
System.out.println(f);

f = TabulatedFunctions.createTabulatedFunction(
functionClass: LinkedListTabulatedFunction.class,
new FunctionPoint[] {
    new FunctionPoint(x: 0, y: 0),
    new FunctionPoint(x: 10, y: 10)
}
);
System.out.println("\nLinkedListTabulatedFunction (FunctionPoint[]): " + f.getClass());
System.out.println(f);

f = TabulatedFunctions.tabulate(functionClass: LinkedListTabulatedFunction.class, new Sin(), leftX: 0, Math.PI, pointsCount: 11);
System.out.println("\ntabulate with reflection (LinkedListTabulatedFunction, Sin): " + f.getClass());
System.out.println(f);
```

Консоль:

```
lisako@lisa MINGW64 ~/Documents/oop/Lab-7-2025 (main)
$ javac functions/*.java
$ javac Main.java
$ java Main
 __testing iterator ArrayTabulatedFunction__
(1.0;1.0)
(2.0;4.0)
(3.0;9.0)
__testing iterator LinkedListTabulatedFunction__
(1.0;1.0)
(2.0;4.0)
(3.0;9.0)
__testing factory__
Array factory: class functions.ArrayTabulatedFunction
LinkedList factory: class functions.LinkedListTabulatedFunction
Array factory: class functions.ArrayTabulatedFunction

__testing reflection__
ArrayTabulatedFunction (double, double, int): class functions.ArrayTabulatedFunction
{(0.0; 0.0),(5.0; 0.0),(10.0; 0.0)}

ArrayTabulatedFunction (double, double, double[]): class functions.ArrayTabulatedFunction
{(0.0; 0.0),(10.0; 10.0)}

LinkedListTabulatedFunction (FunctionPoint[]): class functions.LinkedListTabulatedFunction
{(0.0; 0.0),(10.0; 10.0)}

tabulate with reflection (LinkedListTabulatedFunction, Sin): class
functions.LinkedListTabulatedFunction

{(0.0; 0.0),(0.3141592653589793; 0.3090169943749474),(0.6283185307179586;
0.5877852522924731),(0.9424777960769379; 0.8090169943749475),(1.2566370614359172;
0.9510565162951535),(1.5707963267948966; 1.0),(1.8849555921538759;
0.9510565162951536),(2.199114857512855; 0.8090169943749475),(2.5132741228718345;
0.5877852522924732),(2.827433388230814; 0.3090169943749475),(3.141592653589793;
1.2246467991473532E-16)} Integrator 171: left=15,6733, right=176,7870, step=0,1836,
result=7028,68489584
```