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 Луковникова Е.Д

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Задание №1

В интерфейсе TabulatedFunction добавила необходимый родительский тип.

```java
package functions;

import java.io.*;

public interface TabulatedFunction extends Function, Cloneable,  Iterable<FunctionPoint> {
    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);
    void deletePoint(int index) throws InappropriateFunctionPointException;
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
    Object clone() throws CloneNotSupportedException;
}
```

В классе ArrayTabulatedFunction добавила метод возвращающий объект итератора.

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

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


        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new java.util.NoSuchElementException(s: "Больше нет доступных точек");
            }

            return new FunctionPoint(points[curIndex++]);
        }


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

Метод удаления выбрасывает исключение UnsupportedOperationException. Метод получения следующего элемента выбрасывает исключение NoSuchElementException, если следующего элемента нет.

В LinkedListTabulatedFunction реализовает аналогично.

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

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new java.util.NoSuchElementException(s: "Нет доступных точек");
            }

            FunctionPoint point = new FunctionPoint(currentNode.getPoint());
            currentNode = currentNode.getNext();
            return point;
        }

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

В методе main() выполнена проверка итераторов: созданы тестовые функции обоих типов и осуществлён их обход.

```java
public static void main(String[] args){
    TabulatedFunction f = TabulatedFunctions.tabulate(new Tan(), -10, rightX: 10, pointsCount: 5);
    for (FunctionPoint p : f) {
        System.out.println(p);
    }
```

```
(-10.0; -0.6483608274590866)
(-5.0; 3.380515006246586)
(0.0; 0.0)
(5.0; -3.380515006246586)
(10.0; 0.6483608274590866)
```

Задание №2

В пакете functions реализован интерфейс TabulatedFunctionFactory. Интерфейс содержит три перегруженных фабричных метода, соответствующих различным способам инициализации функций.

```
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);
}
```

Описала в LinkedListTabulatedFunction и ArrayTabulatedFunction классы фабрик.

```
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);
    }

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

```
public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory {

    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
        return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
    }

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

    @Override
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return new LinkedListTabulatedFunction(points);
    }
}
```

В классе TabulatedFunctions добавлено приватное статическое поле типа TabulatedFunctionFactory, инициализированное объектом фабрики ArrayTabulatedFunctionFactory.

```
public final class TabulatedFunctions {

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

Также реализован метод setTabulatedFunctionFactory(), предоставляющий возможность динамической замены используемой фабрики.

```
public static void setTabulatedFunctionFactory(TabulatedFunctionFactory newFactory) {
    factory = newFactory;
}
```

В классе TabulatedFunctions описала три перегруженных метода TabulatedFunction createTabulatedFunction, возвращающих объекты табулированных функций, созданные с помощью текущей фабрики.

```
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);
}
```

В остальных методах (tabulate, inputTabulatedFunction, readTabulatedFunction) класса заменила явное создание объектов с помощью конструкторов на вызов метода createTabulatedFunction.

```
return createTabulatedFunction(functionClass, leftX, rightX, values);
```

В main

```
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction
```

Задание №3

В классе TabulatedFunctions добавила ещё три перегруженных версии метода createTabulatedFunction. Эти методы получают ссылку типа Class на описание класса, объект которого требуется создать. В эти методы можно передавать только ссылки на классы, реализующие интерфейс TabulatedFunction.

```
public static TabulatedFunction createTabulatedFunction(Class<? extends TabulatedFunction> functionClass, double leftX, double rightX, int pointsCount) {

    if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
        throw new IllegalArgumentException("Класс " + functionClass.getName() + " не реализует интерфейс TabulatedFunction");
    }

    try {
        Constructor<?> constructor = functionClass.getConstructor(...parameterTypes: double.class, double.class, int.class);

        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);

    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException("Не найден конструктор (double, double, int) в классе " + functionClass.getName(), e);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании объекта " + functionClass.getName(), e);
    }
}

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

    try {

        Constructor<?> constructor = functionClass.getConstructor(...parameterTypes: double.class, double.class, double[].class);

        return (TabulatedFunction) constructor.newInstance(leftX, rightX, values);

    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException("Не найден конструктор (double, double, double[]) в классе " + functionClass.getName(), e);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании объекта " + functionClass.getName(), e);
    }
}
```

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

    try {
        Constructor<?> constructor = functionClass.getConstructor(...parameterTypes: FunctionPoint[].class);

        return (TabulatedFunction) constructor.newInstance((Object) points);

    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException("Не найден конструктор (FunctionPoint[]) в классе " + functionClass.getName(), e);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании объекта " + functionClass.getName(), e);
    }
}
```

Добавила рефлексивные методы создания TabulatedFunction, которые принимают класс и параметры конструктора. Реализована проверка типа класса и обработка исключений. Также перегружены методы ввода-вывода для поддержки рефлексивного создания объектов.

Сделала так, чтобы в эти методы можно было передать только ссылки на классы, реализующие интерфейс TabulatedFunction.

```
public static TabulatedFunction tabulate(Class<? extends TabulatedFunction> functionClass, Function function, double leftX, double rightX, int pointsCount) {
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
```

```
public static TabulatedFunction inputTabulatedFunction(Class<? extends TabulatedFunction> functionClass, InputStream in) throws IOException {
    DataInputStream dis = new DataInputStream(in);
```

```
public static TabulatedFunction readTabulatedFunction(Class<? extends TabulatedFunction> functionClass, Reader in) throws IOException {
    java.io.StreamTokenizer st = new java.io.StreamTokenizer(in);
```

Методы вызывают соответствующий метод createTabulatedFunction, передавая на него ссылку типа Class, leftX, rightX и массив values.

```
return createTabulatedFunction(functionClass, leftX, rightX, values);
```

Проверила в методе main работу методов рефлексивного создания объектов, а также методов класса TabulatedFunctions, использующих создание объектов.

Результат:

class functions.ArrayTabulatedFunction

{(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

class functions.ArrayTabulatedFunction

{(0.0; 0.0), (10.0; 10.0)}

class functions.LinkedListTabulatedFunction

{(0.0; 0.0), (10.0; 10.0)}

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)}