МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 7

«Реализация итераторов, фабрик и рефлексивного создания объектов для табулированных функций»

по курсу
Объектно-ориентированное программирование

Выполнила: Яньшина Анастасия Юрьевна
Группа 6203-010302D

Содержание

Задание 1

Итак, добавим итераторы во все объекты типа TabulatedFunction. Для начала, меняем интерфейс TabulatedFunction (Рис. 1). Затем добавляем итератор в ArrayTabulatedFunction (Рис. 2) и в LinkedListTabulatedFunction (Рис. 3).

Сразу же тестируем работу итераторов классов табулированных функций, создавая в main объект функции и обращаясь к ней. Как видим, всё корректно работает (Рис. 4).

```
package functions;
import java.io.Serializable;
13 usages  2 implementations
public interface TabulatedFunction extends Function, Serializable, Cloneable, Iterable<FunctionPoint> {
    11 usages  2 implementations
    int getPointsCount();
    4 usages  2 implementations
    FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException;
    no usages  2 implementations
```

Рис. 1

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

        @Override
        public boolean hasNext() {return currentIndex < points.length;}

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {throw new java.util.NoSuchElementException("В табулированной функции больше нет точек!");}
            //возвращаем копию точки для защиты инкапсуляции
            return new FunctionPoint(points[currentIndex++]);
        }

        @Override
        public void remove() {throw new UnsupportedOperationException("Невозможно выполнить операцию удаления!");}
    };
}
```

Рис. 2

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

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {throw new java.util.NoSuchElementException("В табулированной функции больше нет точек!");}
            FunctionPoint point = new FunctionPoint(currentNode.getPoint());
            currentNode = currentNode.getNext();
            return point;
        }

        @Override
        public void remove() {throw new UnsupportedOperationException("Невозможно выполнить операцию удаления!");}
    };
}
```

Рис. 3

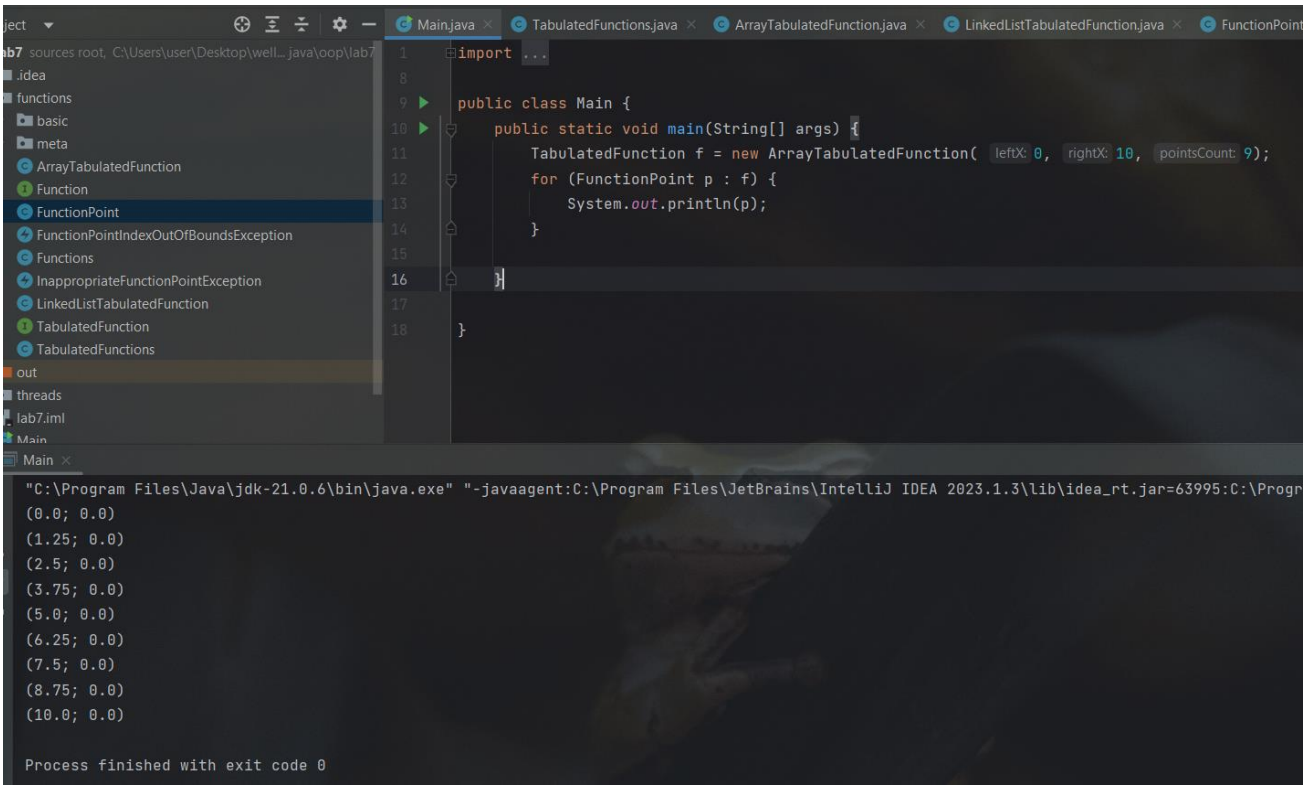


Рис. 4

Задание 2

Следующее задание – реализация фабричного метода. Опишем в пакете `functions` базовый интерфейс фабрик табулированных функций `TabulatedFunctionFactory` (Рис. 5). Теперь добавим в классы `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` вложенные и публичные классы
фабрик `ArrayTabulatedFunctionFactory` и `LinkedListTabulatedFunctionFactory` (Рис. 6 и рис. 7).

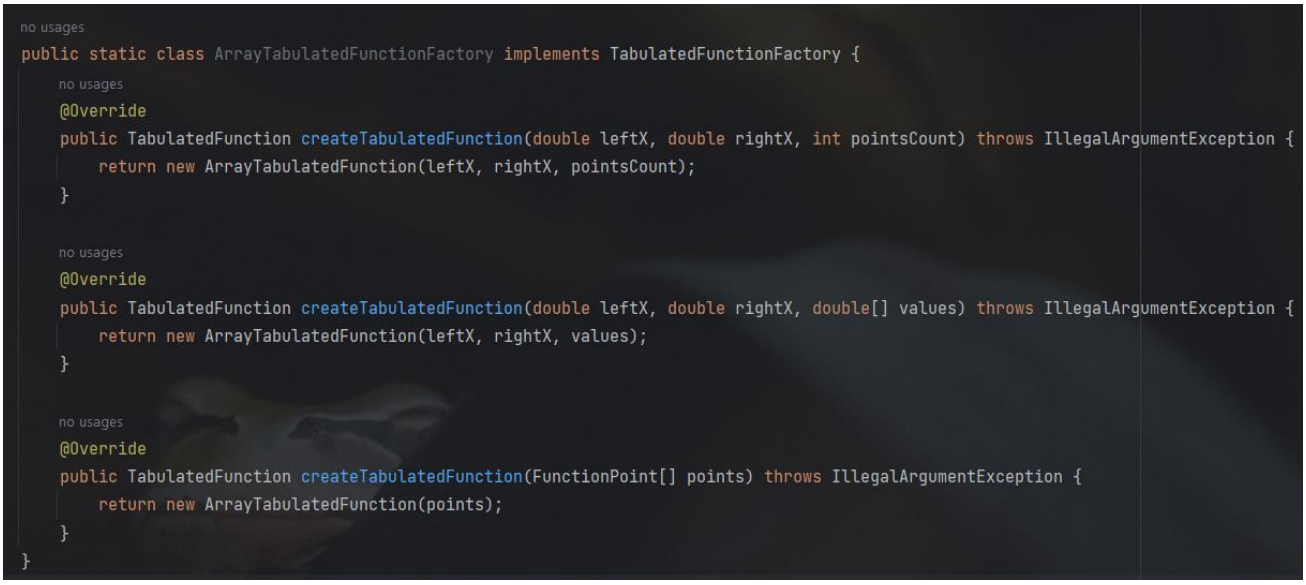
Затем меняем класс `TabulatedFunctions()`. Добавляем поле `TabulatedFunctionFactory factory` и новые методы создания через фабрику (Рис. 8). Помимо этого, в уже существующих методах вносим изменения, чтобы возвращаемый объект был создан через фабрику. Изменения произошли в `inputTabulatedFunction()`, `readTabulatedFunction()` и `tabulate()` (Рис. 9, 10 и 11).

Меняем main и проверяем работу написанного. Вроде всё работает так, как и должно (Рис. 12).

```
package functions;

no usages
public interface TabulatedFunctionFactory {
    no usages
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException;
    no usages
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException;
    no usages
    TabulatedFunction createTabulatedFunction(FunctionPoint[] points) throws IllegalArgumentException;
}
```

Рис. 5

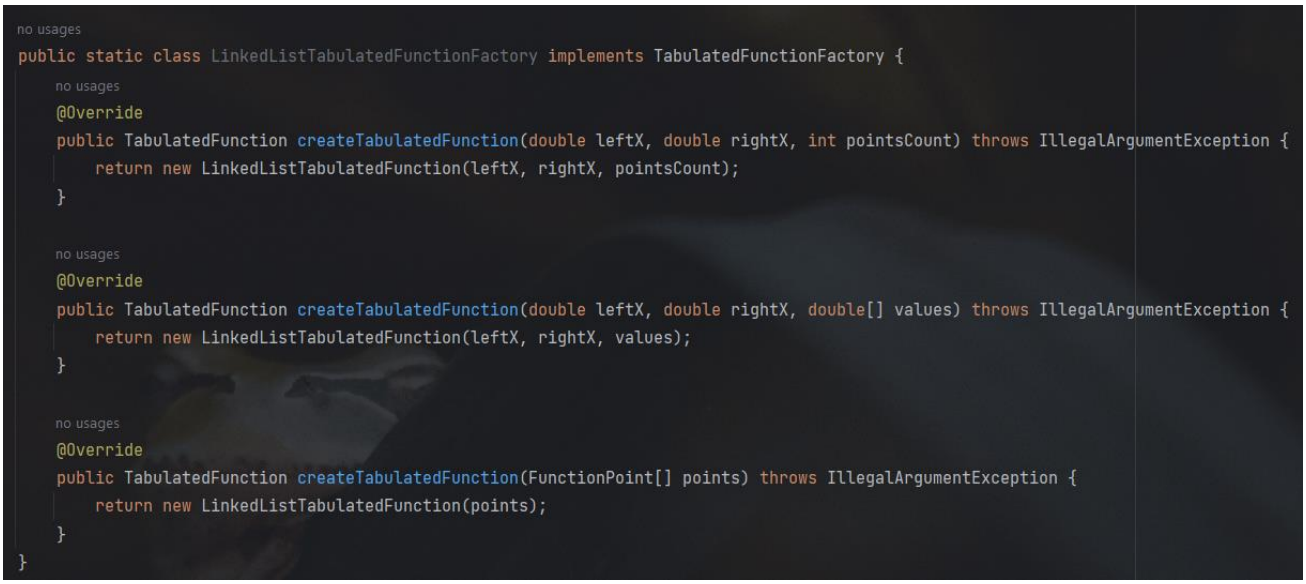


```
no usages
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {
    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {
        return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException {
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) throws IllegalArgumentException {
        return new ArrayTabulatedFunction(points);
    }
}
```

Рис. 6



```
no usages
public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory {
    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {
        return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException {
        return new LinkedListTabulatedFunction(leftX, rightX, values);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) throws IllegalArgumentException {
        return new LinkedListTabulatedFunction(points);
    }
}
```

Рис. 7

```java
public class TabulatedFunctions {
    no usages
    private TabulatedFunctions() {throw new AssertionError( detailMessage: "Нельзя создавать экземпляры утилитного класса");}
    4 usages
    private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();
    no usages
    public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) {TabulatedFunctions.factory = factory;}

    //новые методы создания через фабрику
    no usages
    public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
        return factory.createTabulatedFunction(leftX, rightX, pointsCount);
    }

    no usages
    public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return factory.createTabulatedFunction(leftX, rightX, values);
    }

    no usages
    public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return factory.createTabulatedFunction(points);
    }
```

Рис. 8

```java
3 usages
public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int poin
    //проверяем возможные ошибки в параметрах и корректность границ табулирования
    if (leftX>=rightX) throw new IllegalArgumentException("Левая граница должна быть меньше право
    if (pointsCount < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {thro

    //создаем табулированную функцию, используя TabulatedFunction
    TabulatedFunction result = createTabulatedFunction(leftX, rightX, pointsCount);

    //заполняем значениями исходной функции
    for (int i = 0; i < pointsCount; i++) {
```

Рис. 9

```java
public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
    DataInputStream dis = new DataInputStream(in);
        //читаем количество точек
        int pointsCount = dis.readInt();

        //читаем координаты точек
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(dis.readDouble(), dis.readDouble());
        }

        //создаем табулированную функцию (реализация через ArrayTabulatedFunction как в методе табулирования)
        return factory.createTabulatedFunction(points);
}
```

Рис. 10

```
        st.whitespaceChars( low: ')', hi: ')');
        st.whitespaceChars( low: ',', hi: ',');
        st.whitespaceChars( low: '\t', hi: '\t');
        st.whitespaceChars( low: '\n', hi: '\n');
        st.whitespaceChars( low: '\r', hi: '\r');

        //читаем количество точек
        if (st.nextToken() != StreamTokenizer.TT_WORD) {throw new RuntimeExcep
        int pointsCount = Integer.parseInt(st.sval);

        //читаем координаты точек
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            //читаем X
            if (st.nextToken() != StreamTokenizer.TT_WORD) {throw new RuntimeE
            double x = Double.parseDouble(st.sval);

            //читаем Y
            if (st.nextToken() != StreamTokenizer.TT_WORD) {throw new RuntimeE
            double y = Double.parseDouble(st.sval);

            points[i] = new FunctionPoint(x, y);
        }

        return factory.createTabulatedFunction(points);
}
```

Рис. 11

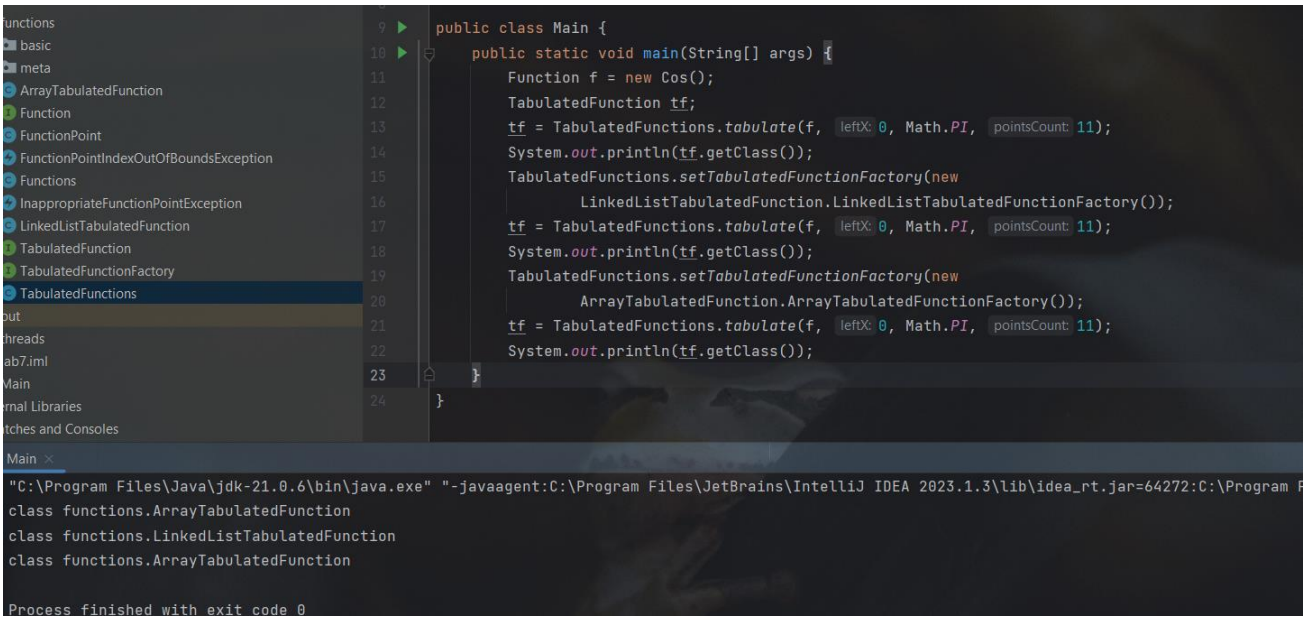


Рис. 12

Задание 3

Последним заданием лабораторной реализуем рефлексивное создание объектов. Добавляем нужные методы для всех трех типов конструкторов в класс `TabulatedFunctions` (Рис. 13 и рис. 14). Реализуем эти три перегруженных метода `createTabulatedFunction()`, принимающих параметр `Class<?>`. Каждый метод проверяет, что класс реализует `TabulatedFunction`, выполняет поиск конструктора с помощью рефлексии, а также обрабатывает исключения с преобразованием в `IllegalArgumentException`.

Также перегружаем несколько существующих методов, чтобы всё работало корректно. Изменения вновь коснулись только `inputTabulatedFunction()`, `readTabulatedFunction()` и `tabulate()` (Рис. 15, 16, 17 и 18).

В завершение проверяем работу написанных методов рефлексивного создания объектов, а также методов класса `TabulatedFunctions`, использующих создание объектов. Используем буквально пример из задания лабораторной (Рис. 19). После запуска убеждаемся, что всё работает корректно (Рис. 20).

```java
//методы рефлексивного создания объектов
2 usages
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, int pointsCount) {
    try {
        //проверяем, что класс реализует TabulatedFunction
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }

        //ищем конструктор с параметрами (double, double, int)
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, int.class);
        //создаём объект
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);

    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}

1 usage
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, double[] values) {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }

        //ищем нужный конструктор и создаём объект
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, double[].class);
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, values);

    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}
```

Рис. 13

```java
1 usage
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, FunctionPoint[] points) {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }

        //ищем нужный конструктор и создаём объект
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);

    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}
```

Рис. 14

```java
//перегруженный метод табулирования с рефлексией
1 usage
public static TabulatedFunction tabulate(Class<?> functionClass, Function function, double leftX, double rightX, int pointsCount) {
    //проверяем возможные ошибки в параметрах и корректность границ табулирования
    if (leftX>=rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой!");
    if (pointsCount < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {throw new IllegalArgumentException("Границы табулирования выходят за

    //создаем табулированную функцию, используя TabulatedFunction
    //проверки в данном случае выполняются при попытке вызова конструктора, ошибки обрабатыва.тся там же
    TabulatedFunction result = createTabulatedFunction(functionClass, leftX, rightX, pointsCount);

    //заполняем значениями исходной функции
    for (int i = 0; i < pointsCount; i++) {
        double x = result.getPointX(i);
        double y = function.getFunctionValue(x);
        result.setPointY(i, y);
    }
    return result;
}
```

Рис. 15

```java
no usages
public static TabulatedFunction inputTabulatedFunction(Class<?> functionClass, InputStream in) throws IOException {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }
        DataInputStream dis = new DataInputStream(in);
        //читаем количество точек
        int pointsCount = dis.readInt();

        //читаем координаты точек
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(dis.readDouble(), dis.readDouble());
        }

        //создаем табулированную функцию через рефлексию
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);
    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}
```

Рис. 16

```java
public static TabulatedFunction readTabulatedFunction(Class<?> functionClass, Reader in) throws IOException {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }
        StreamTokenizer st = new StreamTokenizer(in);

        //настройка токенизатора для чтения чисел
        st.resetSyntax();

        st.wordChars( low: '0',  hi: '9');
        st.wordChars( low: '.',  hi: '.');
        st.wordChars( low: '-',  hi: '-');
        st.wordChars( low: 'E',  hi: 'E');
        st.wordChars( low: 'e',  hi: 'e');

        st.whitespaceChars( low: ' ',  hi: ' ');
        st.whitespaceChars( low: '(',  hi: '(');
        st.whitespaceChars( low: ')',  hi: ')');
        st.whitespaceChars( low: ',',  hi: ',');
        st.whitespaceChars( low: '\t',  hi: '\t');
        st.whitespaceChars( low: '\n',  hi: '\n');
        st.whitespaceChars( low: '\r',  hi: '\r');

        //читаем количество точек
        if (st.nextToken() != StreamTokenizer.TT_WORD) {
            throw new RuntimeException("Ожидалось количество точек!");
        }
        int pointsCount = Integer.parseInt(st.sval);
```

Рис. 17

```java
        //читаем количество точек
        if (st.nextToken() != StreamTokenizer.TT_WORD) {
            throw new RuntimeException("Ожидалось количество точек!");
        }
        int pointsCount = Integer.parseInt(st.sval);

        //читаем координаты точек
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            //читаем X
            if (st.nextToken() != StreamTokenizer.TT_WORD) {
                throw new RuntimeException("Ожидалась координата X!");
            }
            double x = Double.parseDouble(st.sval);

            //читаем Y
            if (st.nextToken() != StreamTokenizer.TT_WORD) {
                throw new RuntimeException("Ожидалась координата Y!");
            }
            double y = Double.parseDouble(st.sval);

            points[i] = new FunctionPoint(x, y);
        }

        //создаем объект через рефлексию
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);
    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);
}
```

Рис. 18

```
    System.out.println("--------------Тест рефлексии--------------");
    TabulatedFunction function;

    function = TabulatedFunctions.createTabulatedFunction(
            ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3);
    System.out.println(function.getClass());
    System.out.println(function);

    function = TabulatedFunctions.createTabulatedFunction(
            ArrayTabulatedFunction.class, leftX: 0, rightX: 10, new double[] {0, 10});
    System.out.println(function.getClass());
    System.out.println(function);

    function = TabulatedFunctions.createTabulatedFunction(
            LinkedListTabulatedFunction.class,
            new FunctionPoint[] {
                    new FunctionPoint( x: 0, y: 0),
                    new FunctionPoint( x: 10, y: 10)
            }
    );
    System.out.println(function.getClass());
    System.out.println(function);

    function = TabulatedFunctions.tabulate(
            LinkedListTabulatedFunction.class, new Sin(), leftX: 0, Math.PI, pointsCount: 11);
    System.out.println(function.getClass());
    System.out.println(function);
}
```

Рис. 19

```
Main
"C:\Program Files\Java\jdk-21.0.6\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=61426:C:\Program Files\JetBrains\IntelliJ IDEA 2023.
--------------Тест итераторов--------------
(0.0; 0.0)
(1.25; 0.0)
(2.5; 0.0)
(3.75; 0.0)
(5.0; 0.0)
(6.25; 0.0)
(7.5; 0.0)
(8.75; 0.0)
(10.0; 0.0)

------------Тест фабричного метода------------
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction

-------------Тест рефлексии-------------
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

Process finished with exit code 0
```

Рис. 20