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

«Использование паттерна «Итератор», паттерна «Фабрика» и рефлексии в Java»

по курсу
Объектно-ориентированное программирование

Выполнил: Петрухин Роман,
студент группы 6203-010302D

Оглавление

Пункт №1

Для выполнения первого и второго подпункта первого пункта я сделал так, чтобы интерфейс TabulatedFunction наследовался от интефейса Iterable <FunctionPoint>.

Дальшейшая реализация данного пункта представлена на рисунках 1-2.

```
@Override  new *
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {  new *
        private FunctionNode curr = head.getNext();  4 usages
        @Override  new *
        public boolean hasNext() {
            return curr != head;
        }

        @Override  new *
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Elements is canceled!");
            }
            FunctionPoint p = new FunctionPoint(curr.getPoint());
            curr = curr.getNext();
            return p;
        }

        @Override  new *
        public void remove() {
            throw new UnsupportedOperationException("You cannot do remove operation!");
        }
    };
```

Рисунок 1

```
@Override  new *
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {  new *
        private int index;  2 usages
        @Override  new *
        public boolean hasNext() {
            return index < amountOfElements;
        }

        @Override  new *
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("NoSuchElement!");
            }
            return new FunctionPoint(massiveOfPoints[index++]);
        }

        @Override  new *
        public void remove() {
            throw new UnsupportedOperationException("You cannot do remove operation!");
        }
    };
}
```

Рисунок 2

Пример работы вышеперечисленных заданий продемонстрирован на рисунках 3-4.

```
System.out.println("---------------The First Task---------------");
TabulatedFunction f2 = new ArrayTabulatedFunction( leftX: 0, rightX: 10, pointsCount: 9);
for (FunctionPoint p : f2) {
    System.out.println(p);
}
```

Рисунок 3

```
---------------The First Task---------------
(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)
```

Рисунок 4

Пункт №2

Реализация интерфейса TabulatedFunctionFactory и вложенных классов ArrayTabulatedFunctionFactory и LinkedListTabulatedFunction представлена на рисунках далее.

```
public interface TabulatedFunctionFactory {  4 usages  2 implementations  new *
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException;
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException;
    TabulatedFunction createTabulatedFunction(FunctionPoint[] massiveOfpoints) throws IllegalArgumentException;  1 usage  2 imple
}
```

Рисунок 5

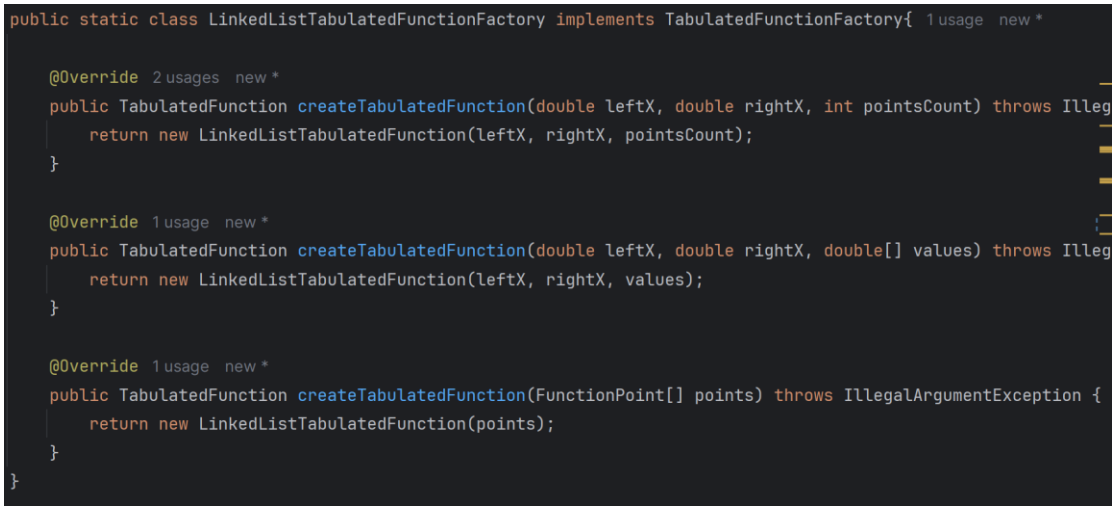


Рисунок 6

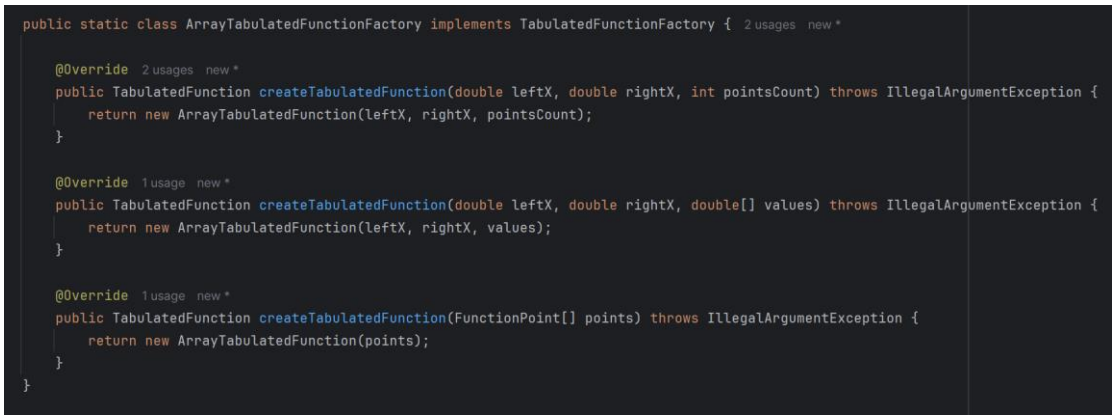


Рисунок 7

На рисунке 8 представлена реализация перегрузки нужных методов в классе TabulatedFubnctions

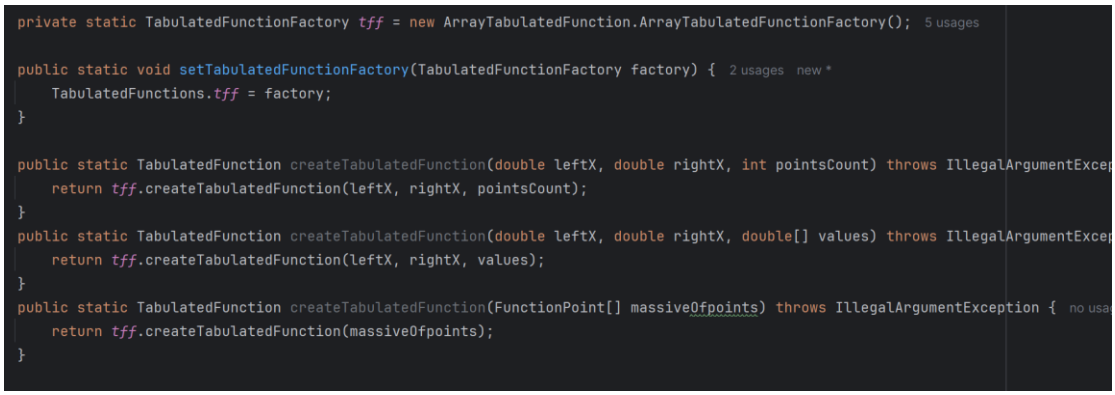


Рисунок 8

Результат работы фукнционала данного пункта будет представлен на рисунках 9-10

```
System.out.println("---------------The Second Task---------------");
Function f1 = new Cos();
TabulatedFunction tf;
tf = TabulatedFunctions.tabulate(f1, leftX: 0, Math.PI, pointsCount: 11);
System.out.println(tf.getClass());
TabulatedFunctions.setTabulatedFunctionFactory(new
        LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f1, leftX: 0, Math.PI, pointsCount: 11);
System.out.println(tf.getClass());
TabulatedFunctions.setTabulatedFunctionFactory(new
        ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f1, leftX: 0, Math.PI, pointsCount: 11);
System.out.println(tf.getClass());
```

Рисунок 9

```
---------------The Second Task---------------
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction
```

Рисунок 10

Пункт №3

Реализация данного задания будет представлена на рисунках далее.

```
public static TabulatedFunction createTabulatedFunction(Class<?> fClass, double leftX, double rightX, int pointsCount) {  2 usag
    try {
        // isAssignableFrom() проверяет, можно ли привести functionClass к TabulatedFunction
        if (!TabulatedFunction.class.isAssignableFrom(fClass)) {
            throw new IllegalArgumentException("Class must be realize TabulatedFunction interface!");
        }

        // Ищем конструктор с параметрами (double, double, int)
        java.lang.reflect.Constructor<?> constructor = fClass.getConstructor(double.class, double.class, int.class);
        // Вызывает найденный конструктор с переданными аргументами
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);

    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException("Constructor (double, double, int) not found in class: " + fClass.getName(), e);
    } catch (InstantiationException | IllegalAccessException | java.lang.reflect.InvocationTargetException e) {
        throw new IllegalArgumentException("Cannot create instance of class: " + fClass.getName(), e);
    }
}

/**
```

Рисунок 11

На рисунке 11 представдена реализация метода createTabulatedFunction с получаемым аргументом Class<?> class в сигнатуре функции. 2 оставшиеся функции реализованы подобным образом. Все исключения корректно обрабатываются через try-catch.

На последующих рисунках будет приведена реализация методов старых методов, создающих объект табулированной функции.

```
public static TabulatedFunction tabulate(Class<?> fClass, Function function, double leftX, double rightX, int pointsCount) {
    // Проверка входных параметров
    if (function == null) {
        throw new IllegalArgumentException("Function cannot be null");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("LeftX cannot be greater or equal to rightX");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Points count must be greater than 1");
    }

    // Проверка, что отрезок табулирования принадлежит области определения
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
        throw new IllegalArgumentException("Tabulation interval is outside function domain");
    }

    // Создание табулированной функции
    TabulatedFunction tabulatedFunc = createTabulatedFunction(fClass, leftX, rightX, pointsCount);

    // Заполнение значений функции
    for (int i = 0; i < pointsCount; i++) {
        double x = tabulatedFunc.getPointX(i);
        double y = function.getFunctionValue(x);
        tabulatedFunc.setPointY(i, y);
    }

    return tabulatedFunc;
}
```

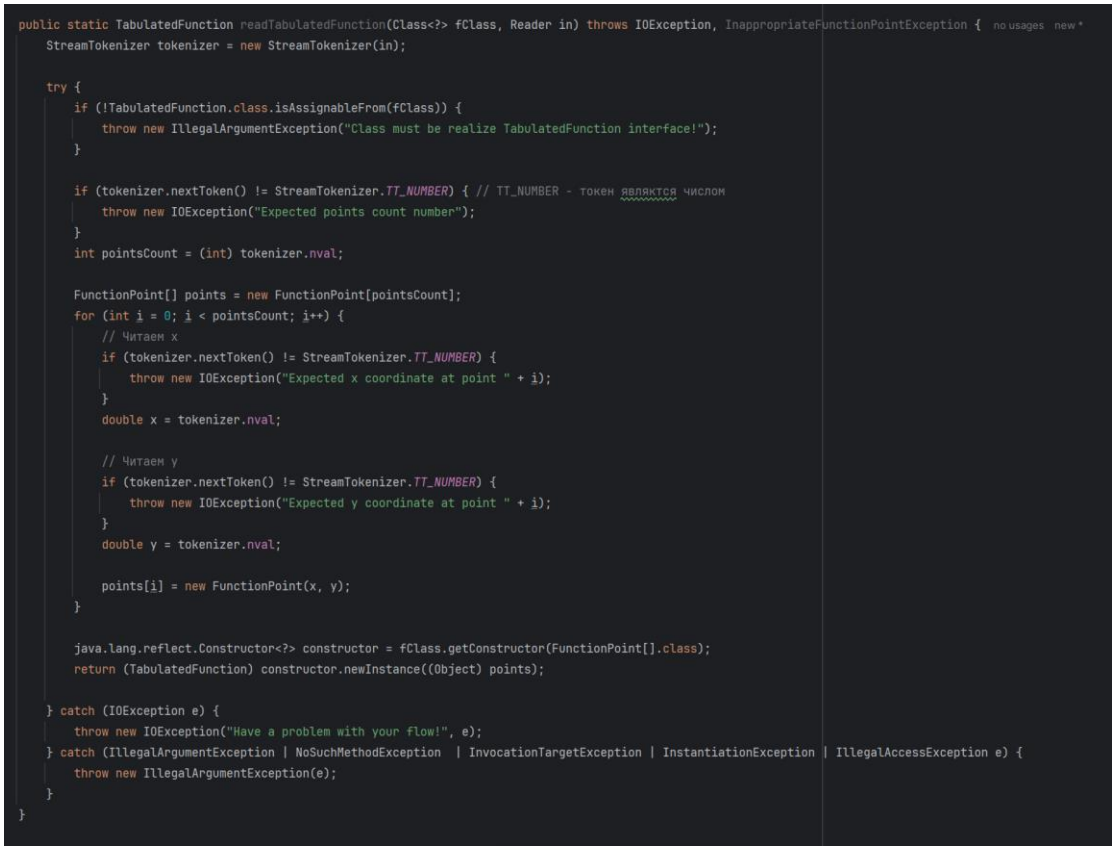
Рисунок 12

На рисунке 12 представлена перегрузка функции tabulate().

```
public static TabulatedFunction inputTabulatedFunction(Class<?> fClass, InputStream in) throws IOException,
    if (!TabulatedFunction.class.isAssignableFrom(fClass)) {
        throw new IllegalArgumentException("Class must be realize TabulatedFunction interface!");
    }
    DataInputStream data = new DataInputStream(in);

    FunctionPoint[] points;
    try {
        int pCount = data.readInt();
        points = new FunctionPoint[pCount];
        for (int i = 0; i < pCount; i++) {
            points[i] = new FunctionPoint(data.readDouble(), data.readDouble());
        }
    } catch (IOException e) {
        throw new IOException("Have a problem with your flow!", e);
    }
    java.lang.reflect.Constructor<?> constructor = fClass.getConstructor(FunctionPoint[].class);
    return (TabulatedFunction) constructor.newInstance((Object) points);
}
```

Рисунок 13

На рисунке 13 представлена реализация перегрузки функции inputTabulatedFunction().



```java
public static TabulatedFunction readTabulatedFunction(Class<?> fClass, Reader in) throws IOException, InappropriateFunctionPointException {  no usages  new *
    StreamTokenizer tokenizer = new StreamTokenizer(in);

    try {
        if (!TabulatedFunction.class.isAssignableFrom(fClass)) {
            throw new IllegalArgumentException("Class must be realize TabulatedFunction interface!");
        }

        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) { // TT_NUMBER - токен являются числом
            throw new IOException("Expected points count number");
        }
        int pointsCount = (int) tokenizer.nval;

        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            // Читаем x
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Expected x coordinate at point " + i);
            }
            double x = tokenizer.nval;

            // Читаем y
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Expected y coordinate at point " + i);
            }
            double y = tokenizer.nval;

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

        java.lang.reflect.Constructor<?> constructor = fClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);

    } catch (IOException e) {
        throw new IOException("Have a problem with your flow!", e);
    } catch (IllegalArgumentException | NoSuchMethodException  | InvocationTargetException | InstantiationException | IllegalAccessException e) {
        throw new IllegalArgumentException(e);
    }
}
```

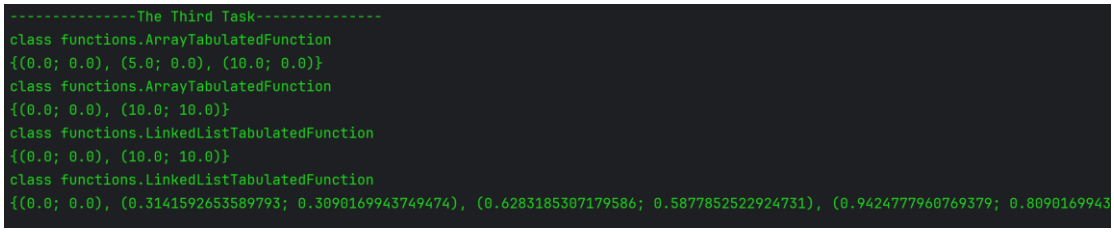
Рисунок 14

На рисунке 14 представлена реализация перегрузки функции readTabulatedFunction

Далее, на рисунках 15-16 будет представлен результат работы функций данного пункта.

```java
System.out.println("---------------The Third Task---------------");
TabulatedFunction f;

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

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

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

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

Рисунок 15


---------------The Third Task---------------
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; 0.8090169943
Рисунок 16