Лабораторная работа №7

Выполнил: Магера Никита Алексеевич

Студент группы 6203-010302D

Ход выполнения

Задание 1

В 1 задании реализован паттерн "Итератор" для табулированных функций. Интерфейс TabulatedFunction расширен для поддержки Iterable<FunctionPoint>, что позволяет использовать цикл for-each. В классах ArrayTabulatedFunction и LinkedListTabulatedFunction добавлены анонимные итераторы, работающие напрямую с внутренними структурами данных для эффективности. Итераторы возвращают копии точек, сохраняя инкапсуляцию. Методы соответствуют требованиям: next() бросает NoSuchElementException при отсутствии элементов, remove() всегда вызывает UnsupportedOperationException. Работа проверена в main() созданием объектов и их перебором в for-each.(Рис.1-2)

```
package functions;

import java.util.Iterator;
import functions.Function;
import functions.FunctionPoint;
import functions.InappropriateFunctionPointException;

public interface TabulatedFunction extends Function, Iterable<FunctionPoint>, Cloneable {  28 usages  2 implem

// все объекты типа клонируемыми с точки зрения JVM

    int getPointsCount();  16 usages  2 implementations  new *

    FunctionPoint getPoint(int index);  4 usages  2 implementations  new *

    double getPointX(int index);  10 usages  2 implementations  new *

    double getPointY(int index);  16 usages  2 implementations  new *

    void setPointY(int index, double y);  5 usages  2 implementations  new *

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

    void setPointX(int index, double x) throws functions.InappropriateFunctionPointException;  1 usage  2 imp

    void deletePoint(int index);  4 usages  2 implementations  new *

    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;  3 usages  2 implementations

    Object clone() throws CloneNotSupportedException;  2 implementations  new *
}
```

Рис. 1

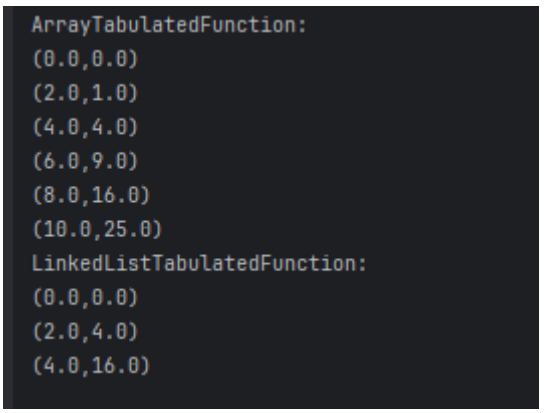


Рис. 2

Задание 2

В 2 задании требовалось реализовать систему фабрик для создания табулированных функций. Добавлен интерфейс TabulatedFunctionFactory с тремя методами создания. В классы ArrayTabulatedFunction и LinkedListTabulatedFunction добавлены вложенные классы-фабрики. В TabulatedFunctions добавлено статическое поле factory и метод setTabulatedFunctionFactory() для выбора типа создаваемых объектов. Все методы создания теперь используют фабрику, что позволяет менять реализацию динамически. (Рис.3-5)

```
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {  1 usage  new *

    // метод для создания по границам и количеству точек
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {  1 usage  new
        return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
    }
    // метод для создания по границам и массиву значений y
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {  3 usages  ne
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }
    //метод для создания по массиву точек
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {  no usages  new *
        return new ArrayTabulatedFunction(points);
    }
}
```

Рис.3

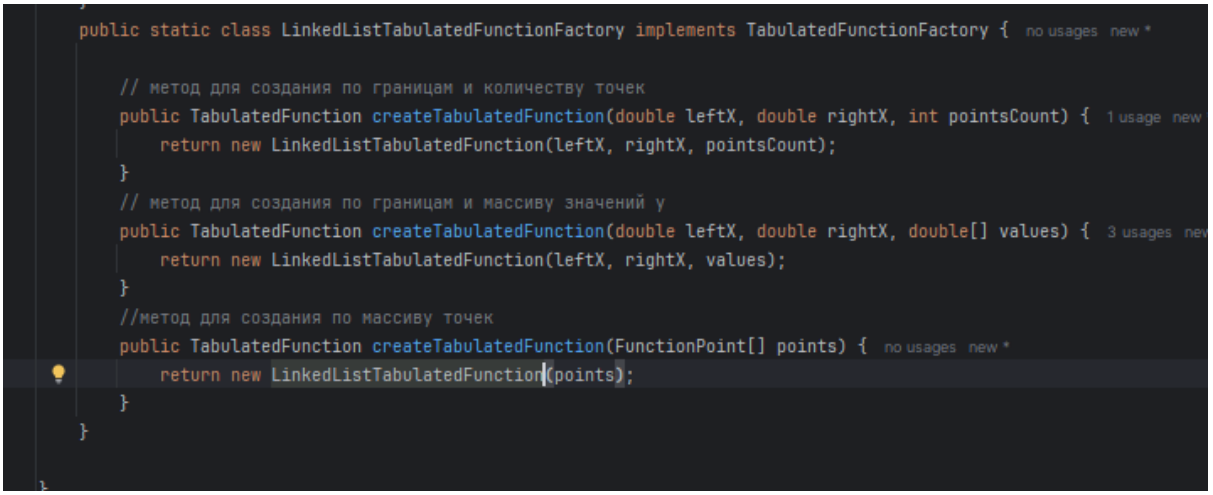


Рис.4

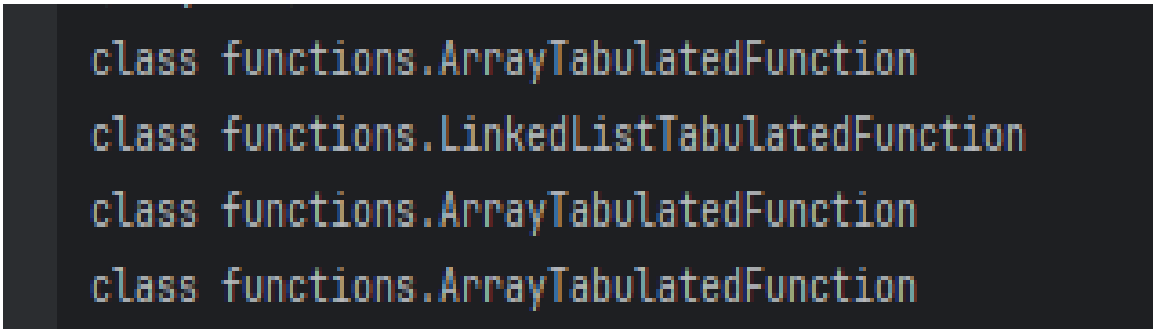


Рис.5

Задание 3

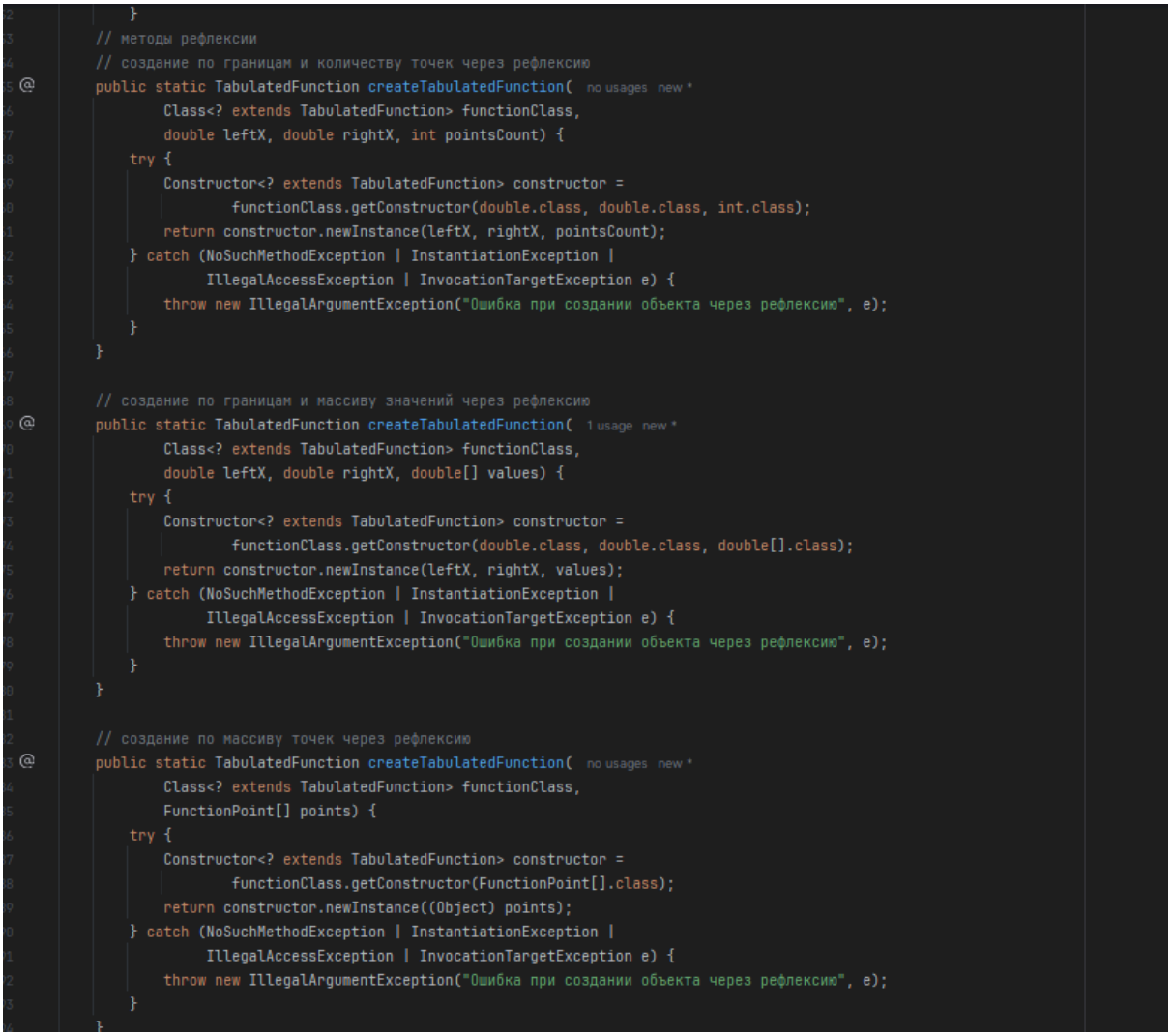
В 3 задании была добавлена возможность создания объектов через рефлексию. В TabulatedFunctions добавлены три метода createTabulatedFunction() и метод tabulate(), принимающие параметр Class<? extends TabulatedFunction>. Эти методы находят конструктор с нужными параметрами через рефлексию и создают объект. Исключения рефлексии преобразуются в IllegalArgumentException. Это позволяет создавать объекты любых классов, реализующих TabulatedFunction, без жесткого связывания.(Рис.6-9)

```
System.out.println(tf.getClass());
// тесты рефлексии
TabulatedFunction f_2;
f_2 = TabulatedFunctions.createTabulatedFunction(
        ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3);
System.out.println(f_2.getClass());
System.out.println(f_2);
f_2 = TabulatedFunctions.createTabulatedFunction(
        ArrayTabulatedFunction.class, leftX: 0, rightX: 10, new double[] {0, 10});
System.out.println(f_2.getClass());
System.out.println(f_2);
f_2 = TabulatedFunctions.createTabulatedFunction(
        LinkedListTabulatedFunction.class,
        new FunctionPoint[] {
                new FunctionPoint( x: 0, y: 0),
                new FunctionPoint( x: 10, y: 10)});
System.out.println(f_2.getClass());
System.out.println(f_2);
f_2 = TabulatedFunctions.tabulate(
        LinkedListTabulatedFunction.class, new Sin(), leftX: 0, Math.PI, pointsCount: 11);
System.out.println(f_2.getClass());
System.out.println(f_2);
```

Рис.6

```
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),
```

Рис.7

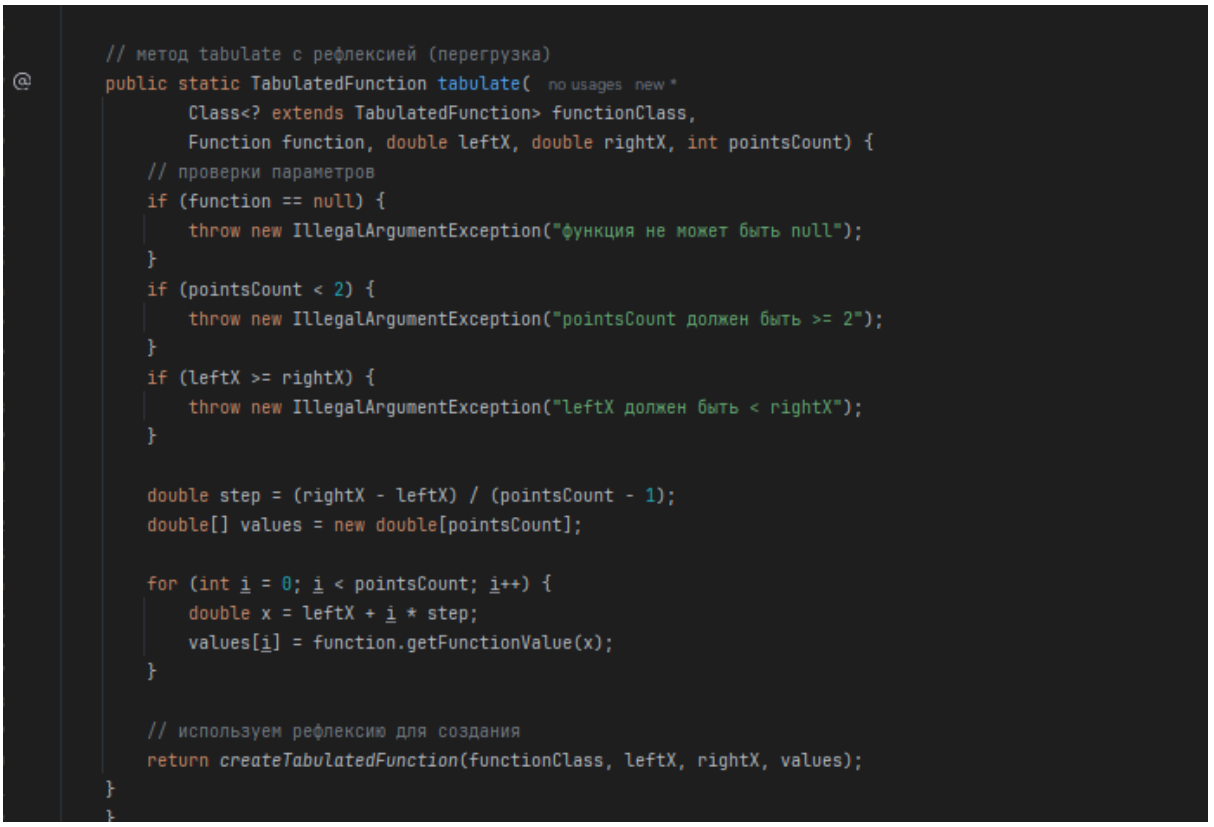


```
    }
// методы рефлексии
// создание по границам и количеству точек через рефлексию
public static TabulatedFunction createTabulatedFunction(  no usages  new *
        Class<? extends TabulatedFunction> functionClass,
        double leftX, double rightX, int pointsCount) {
    try {
        Constructor<? extends TabulatedFunction> constructor =
                functionClass.getConstructor(double.class, double.class, int.class);
        return constructor.newInstance(leftX, rightX, pointsCount);
    } catch (NoSuchMethodException | InstantiationException |
             IllegalAccessException | InvocationTargetException e) {
        throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
    }
}

// создание по границам и массиву значений через рефлексию
public static TabulatedFunction createTabulatedFunction(  1 usage  new *
        Class<? extends TabulatedFunction> functionClass,
        double leftX, double rightX, double[] values) {
    try {
        Constructor<? extends TabulatedFunction> constructor =
                functionClass.getConstructor(double.class, double.class, double[].class);
        return constructor.newInstance(leftX, rightX, values);
    } catch (NoSuchMethodException | InstantiationException |
             IllegalAccessException | InvocationTargetException e) {
        throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
    }
}

// создание по массиву точек через рефлексию
public static TabulatedFunction createTabulatedFunction(  no usages  new *
        Class<? extends TabulatedFunction> functionClass,
        FunctionPoint[] points) {
    try {
        Constructor<? extends TabulatedFunction> constructor =
                functionClass.getConstructor(FunctionPoint[].class);
        return constructor.newInstance((Object) points);
    } catch (NoSuchMethodException | InstantiationException |
             IllegalAccessException | InvocationTargetException e) {
        throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
    }
}
```

Рис.8



```java
// метод tabulate с рефлексией (перегрузка)
public static TabulatedFunction tabulate(  no usages  new *
        Class<? extends TabulatedFunction> functionClass,
        Function function, double leftX, double rightX, int pointsCount) {
    // проверки параметров
    if (function == null) {
        throw new IllegalArgumentException("функция не может быть null");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("pointsCount должен быть >= 2");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("leftX должен быть < rightX");
    }

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

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

    // используем рефлексию для создания
    return createTabulatedFunction(functionClass, leftX, rightX, values);
}
}
```

Рис.9