

Отчёт по лабораторной работе №7

Студент: Прокопенко А. П.

Группа: 6201-120303D

Задание 1. Реализация паттерна "Итератор"

1.1. Изменения в интерфейсе TabulatedFunction

```
package functions;

import java.io.Serializable;

public interface TabulatedFunction extends Function, Serializable, Cloneable, Iterable<FunctionPoint> { 49 usages 2 implementations
    int getPointsCount(); 26 usages 2 implementations
    FunctionPoint getPoint(int index); 9 usages 2 implementations
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; 1 usage 2 implementations
    double getPointX(int index); 15 usages 2 implementations
    void setPointX(int index, double x) throws InappropriateFunctionPointException; no usages 2 implementations
    double getPointY(int index); 13 usages 2 implementations
    void setPointY(int index, double y); no usages 2 implementations
    void deletePoint(int index); no usages 2 implementations
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; no usages 2 implementations
    Object clone(); 2 implementations
}
```

1.2. Реализация итератора в ArrayTabulatedFunction

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

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Нет следующего элемента");
            }
            // Возвращаем копию точки для защиты инкапсуляции
            return new FunctionPoint(points[currentIndex++]);
        }

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

1.3. Реализация итератора в LinkedListTabulatedFunction

```
@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private FunctionNode currentNode = head.getNext(); 3 usages
        private int currentIndex = 0; 2 usages

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Нет следующего элемента");
            }
            // Возвращаем копию точки для защиты инкапсуляции
            FunctionPoint point = new FunctionPoint(currentNode.getPoint());
            currentNode = currentNode.getNext();
            currentIndex++;
            return point;
        }

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

Результаты тестирования:

1. ТЕСТИРОВАНИЕ ИТЕРАТОРОВ (for-each):

=====

a) ArrayTabulatedFunction:

Точки функции (через for-each):

1. (0.0; 0.0)
2. (1.0; 1.0)
3. (2.0; 4.0)
4. (3.0; 9.0)
5. (4.0; 16.0)

Проверка значений через итератор:

Все точки совпадают с ожидаемыми

b) LinkedListTabulatedFunction:

Точки функции (через for-each):

1. (0.0; 0.0)
2. (1.0; 1.0)
3. (2.0; 4.0)
4. (3.0; 9.0)
5. (4.0; 16.0)

Проверка значений через итератор:

Все точки совпадают с ожидаемыми

c) Тестирование исключений итератора:

NoSuchElementException поймано: Нет следующего элемента

UnsupportedOperationException поймано: Удаление не поддерживается

Задание 2. Реализация паттерна "Фабричный метод"

2.1. Интерфейс фабрики

```
package functions;

public interface TabulatedFunctionFactory { 4 usages 2 implementations
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount); 1 usage 2 implementations
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values); 1 usage 2 implementations
    TabulatedFunction createTabulatedFunction(FunctionPoint[] points); 1 usage 2 implementations
}
```

2.2. Фабрики в классах табулированных функций

```
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory { 2 usages
    @Override 1 usage
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {...}
    @Override 1 usage
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {...}
    @Override 1 usage
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {...}
}

public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory { 1 usage
    @Override 1 usage
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {...}
    @Override 1 usage
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {...}
    @Override 1 usage
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {...}
}
```

2.3. Изменения в классе TabulatedFunctions

```
private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory(); 4 usages

// Метод для установки фабрики
public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) { 2 usages
    TabulatedFunctions.factory = factory;
}

public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) { 2 usages
    return factory.createTabulatedFunction(leftX, rightX, pointsCount);
}

public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) { 2 usages
    return factory.createTabulatedFunction(leftX, rightX, values);
}

public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) { 3 usages
    return factory.createTabulatedFunction(points);
}
```

Результаты тестирования

2. ТЕСТИРОВАНИЕ ФАБРИЧНОГО МЕТОДА:

=====

a) Фабрика по умолчанию:

Тип: ArrayTabulatedFunction

Количество точек: 11

Левая граница: 0.0

Правая граница: 3.141592653589793

Проверка значений косинуса:

Все значения косинуса вычислены корректно

b) Смена фабрики на LinkedListTabulatedFunctionFactory:

Тип: LinkedListTabulatedFunction

Количество точек: 11

c) Возвращаем ArrayTabulatedFunctionFactory:

Тип: ArrayTabulatedFunction

d) Методы createTabulatedFunction:

Метод 1 (границы + количество точек):

Тип: ArrayTabulatedFunction, точек: 5

Проверка равномерности точек:

Точки распределены равномерно

Метод 2 (границы + значения):

Тип: ArrayTabulatedFunction, точек: 5

Проверка значений:

Все значения сохранены корректно

Метод 3 (массив точек):

Тип: ArrayTabulatedFunction, точек: 3

Проверка точек:

Все точки сохранены корректно

Задание 3. Использование рефлексии

3.1. Методы создания через рефлексию в TabulatedFunctions

createdTabulatedFunction

```
public static TabulatedFunction createTabulatedFunction( 4 usages
    Class<?> clazz, double leftX, double rightX, int pointsCount) {

    // Проверяем, что класс реализует TabulatedFunction
    if (!TabulatedFunction.class.isAssignableFrom(clazz)) {
        throw new IllegalArgumentException(
            "Класс " + clazz.getName() + " не реализует интерфейс TabulatedFunction");
    }

    try {
        // Находим конструктор с параметрами (double, double, int)
        Constructor<?> constructor = clazz.getConstructor(double.class, double.class, int.class);

        // Создаем объект с помощью конструктора
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);
    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException(
            "Класс " + clazz.getName() + " не имеет конструктора (double, double, int)", e);
    } catch (Exception e) {
        throw new IllegalArgumentException(
            "Ошибка при создании объекта класса " + clazz.getName(), e);
    }
}
```

```
public static TabulatedFunction createTabulatedFunction( 2 usages
    Class<?> clazz, double leftX, double rightX, double[] values) {

    // Проверяем, что класс реализует TabulatedFunction
    if (!TabulatedFunction.class.isAssignableFrom(clazz)) {
        throw new IllegalArgumentException(
            "Класс " + clazz.getName() + " не реализует интерфейс TabulatedFunction");
    }

    try {
        // Находим конструктор с параметрами (double, double, double[])
        Constructor<?> constructor = clazz.getConstructor(double.class, double.class, double[].class);

        // Создаем объект с помощью конструктора
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, values);
    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException(
            "Класс " + clazz.getName() + " не имеет конструктора (double, double, double[])", e);
    } catch (Exception e) {
        throw new IllegalArgumentException(
            "Ошибка при создании объекта класса " + clazz.getName(), e);
    }
}
```

```

public static TabulatedFunction createTabulatedFunction( 4 usages
    Class<?> clazz, FunctionPoint[] points) {

    // Проверяем, что класс реализует TabulatedFunction
    if (!TabulatedFunction.class.isAssignableFrom(clazz)) {
        throw new IllegalArgumentException(
            "Класс " + clazz.getName() + " не реализует интерфейс TabulatedFunction");
    }

    try {
        // Находим конструктор с параметрами (FunctionPoint[])
        Constructor<?> constructor = clazz.getConstructor(FunctionPoint[].class);

        // Создаем объект с помощью конструктора
        return (TabulatedFunction) constructor.newInstance((Object) points);
    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException(
            "Класс " + clazz.getName() + " не имеет конструктора (FunctionPoint[])", e);
    } catch (Exception e) {
        throw new IllegalArgumentException(
            "Ошибка при создании объекта класса " + clazz.getName(), e);
    }
}

```

tabulated

```

public static TabulatedFunction tabulate( 2 usages
    Class<?> clazz, Function function, double leftX, double rightX, int pointsCount) {

    // Проверка корректности параметров
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2: " + pointsCount);
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой: " + leftX + " >= " + rightX);
    }

    // Проверка области определения
    double functionLeftBorder = function.getLeftDomainBorder();
    double functionRightBorder = function.getRightDomainBorder();
    if (leftX < functionLeftBorder || rightX > functionRightBorder) {
        throw new IllegalArgumentException("Границы табулирования [" + leftX + ", " + rightX + "] " +
            "выходят за область определения функции [" + functionLeftBorder + ", " + functionRightBorder + "]"
        );
    }

    // Создание массива значений Y путем вычисления функции в равномерно распределенных точках
    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        values[i] = function.getFunctionValue(x);
    }

    // Используем рефлексия для создания объекта
    return createTabulatedFunction(clazz, leftX, rightX, values);
}

```

3.2. Тестирование рефлексии

3. ТЕСТИРОВАНИЕ РЕФЛЕКСИИ:

=====

a) Создание ArrayTabulatedFunction (границы + количество точек):

Тип: ArrayTabulatedFunction

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

Проверка точек:

Точки созданы корректно

b) Создание ArrayTabulatedFunction (границы + значения):

Тип: ArrayTabulatedFunction

Данные: {(0.0; 0.0), (5.0; 10.0), (10.0; 20.0)}

Проверка значений:

Значения сохранены корректно

c) Создание LinkedListTabulatedFunction (массив точек):

Тип: LinkedListTabulatedFunction

Данные: {(0.0; 0.0), (10.0; 10.0)}

Проверка точек:

Точки сохранены корректно

d) Табулирование функции Sin через рефлексию:

Тип: LinkedListTabulatedFunction

Количество точек: 11

Проверка значений синуса (первые 3 точки):

Точка 0: $x=0,0000$, $\sin(x)=0,000000$

Точка 1: $x=0,3142$, $\sin(x)=0,309017$

Точка 2: $x=0,6283$, $\sin(x)=0,587785$

Первые 3 точек вычислены корректно

e) Тестирование обработки ошибок рефлексии:

e.1) Класс не реализует `TabulatedFunction`:

Класс `java.lang.Integer` не реализует интерфейс `TabulatedFunction`

e.2) Ошибка в конструкторе (`leftX >= rightX`):

Ошибка при создании объекта класса `functions.ArrayTabulatedFunction`

Исключение в конструкторе: `null`

f) Сравнение фабрики и рефлексии:

Через фабрику (текущая):

Тип: `ArrayTabulatedFunction`

Через рефлексию (`LinkedListTabulatedFunction`):

Тип: `LinkedListTabulatedFunction`

Сравнение координат X:

Координаты X совпадают

g) Тестирование сериализации с рефлексией:

Исходная функция: `LinkedListTabulatedFunction`

Восстановленная функция: `ArrayTabulatedFunction`

Все точки совпадают с точностью `1.0E-10`

h) Комплексный тест всех механизмов:

Создана функция: `LinkedListTabulatedFunction`

Точек: 9

Проверка значений косинуса:

Корректно вычислено: 9/9 точек

Вывод через `for-each`:

$x = 0,0000, \cos(x) = 1,000000$

$x = 0,7854, \cos(x) = 0,707107$

$x = 1,5708, \cos(x) = 0,000000$

$x = 2,3562, \cos(x) = -0,707107$

$x = 3,1416, \cos(x) = -1,000000$

$x = 3,9270, \cos(x) = -0,707107$

$x = 4,7124, \cos(x) = -0,000000$

$x = 5,4978, \cos(x) = 0,707107$

$x = 6,2832, \cos(x) = 1,000000$

Проверка границ области определения:

Границы корректны: $[0, 2\pi]$