Лабораторная работа №7

Коновалов Сергей Сергеевич 6204-010302D

Содержание

Задание 1

Добавил поддержку итераторов в классы ArrayTabulatedFunction и LinkedListTabulatedFunction. Реализовал метод iterator(), возвращающий объект итератора, который позволяет использовать цикл for-each для перебора точек функции. Сделал итераторы анонимными классами, работающими напрямую с внутренней структурой данных для эффективности. Также добавил обработку исключений NoSuchElementException и UnsupportedOperationException в соответствии с требованиями задания.

```
package functions;

import java.util.Iterator;

public interface TabulatedFunction extends Function, Iterable<FunctionPoint> {

    int getPointsCount();

    FunctionPoint getPoint(int index);

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

    double getPointX(int index);

    double getPointY(int index);

    void setPointX(int index, double x) throws InappropriateFunctionPointException;

    void setPointY(int index, double y);

    void deletePoint(int index);

    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;

    Object clone();
}
```

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

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("No more points available");
            }
            // Создаем копию точки для защиты инкапсуляции
            FunctionPoint point = points[currentIndex];
            FunctionPoint copy = new FunctionPoint(point.getX(), point.getY());
            currentIndex++;
            return copy;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("Remove operation is not supported");
        }
    };
}
```

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

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("No more points available");
            }
            // Создаем копию точки для защиты инкапсуляции
            FunctionPoint point = currentNode.getPoint();
            FunctionPoint copy = new FunctionPoint(point.getX(), point.getY());
            currentNode = currentNode.getNext();
            iterations++;
            return copy;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("Remove operation is not supported");
        }
    };
}
```

Задание 2

Выполнил внедрение паттерна "Фабричный метод" для динамического создания объектов табулированных функций. Создал интерфейс TabulatedFunctionFactory с тремя методами создания функций. Добавил в классы ArrayTabulatedFunction и LinkedListTabulatedFunction вложенные классы фабрик, реализующие этот интерфейс. Изменил класс TabulatedFunctions, заменив прямые вызовы конструкторов на использование фабричных методов, что позволило динамически менять тип создаваемых объектов во время выполнения программы.

```
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);
}
```

```
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);
    }
}
```

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

public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) {
    TabulatedFunctions.factory = factory;
}

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);
}
```

Задание 3

Добавил в класс TabulatedFunctions методы создания объектов с использованием механизма рефлексии. Реализовал перегруженные версии методов createTabulatedFunction, принимающие класс создаваемого объекта в качестве параметра. Сделал проверку, что переданный класс реализует интерфейс TabulatedFunction. Добавил обработку исключений рефлексии с преобразованием их в IllegalArgumentException. Также создал перегруженные методы tabulate, использующие рефлексию для создания объектов указанного класса.

```
    // Методы с рефлексией
public static TabulatedFunction createTabulatedFunction(Class<?> 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(double.class, double.class, int.class);
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
    }
}

public static TabulatedFunction createTabulatedFunction(Class<?> 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(double.class, double.class, double[].class);
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, values);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
    }
}

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

    try {
        Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
    }
}
```

Вывод Main:

```
================================================================================
Тестирование итераторов
================================================================================

1. Итератор для ArrayTabulatedFunction
------------------------------------------------
Функция: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0), (4.0; 16.0)}

Итерация с помощью for-each:
  (0.0; 0.0)
  (1.0; 1.0)
  (2.0; 4.0)
  (3.0; 9.0)
  (4.0; 16.0)

Явное использование итератора:
  (0.0; 0.0)
  (1.0; 1.0)
  (2.0; 4.0)
  (3.0; 9.0)
  (4.0; 16.0)


2. Итератор для LinkedListTabulatedFunction
------------------------------------------------------
Функция: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0), (4.0; 16.0)}

Итерация с помощью for-each:
  (0.0; 0.0)
  (1.0; 1.0)
  (2.0; 4.0)
  (3.0; 9.0)
  (4.0; 16.0)

Явное использование итератора:
  (0.0; 0.0)
  (1.0; 1.0)
  (2.0; 4.0)
  (3.0; 9.0)
  (4.0; 16.0)


3. Тестирование исключений итераторов
------------------------------------------------
Тестирование NoSuchElementException:
  Поймано NoSuchElementException: No more points available
Тестирование UnsupportedOperationException для remove():
  Поймано UnsupportedOperationException: Remove operation is not supported
Тестирование защиты инкапсуляции:
  Получена точка из итератора: (0.0; 0.0)
  Точка после изменения: (999.0; 999.0)
  Исходная функция не изменилась: {(0.0; 0.0), (1.0; 1.0)}
  Защита инкапсуляции работает: true
================================================================================
Тестирование фабрик
================================================================================
Фабрика по умолчанию: ArrayTabulatedFunction
LinkedList фабрика: LinkedListTabulatedFunction
Array фабрика: ArrayTabulatedFunction

Тестирование разных конструкторов:
Через массив значений: ArrayTabulatedFunction
Через массив точек: ArrayTabulatedFunction
================================================================================
Тестирование рефлексии
================================================================================
1. Создание ArrayTabulatedFunction через рефлексию:
   Класс: class functions.ArrayTabulatedFunction
   Функция: {(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

2. Создание ArrayTabulatedFunction с массивом значений:
   Класс: class functions.ArrayTabulatedFunction
   Функция: {(0.0; 0.0), (5.0; 50.0), (10.0; 100.0)}

3. Создание LinkedListTabulatedFunction с массивом точек:
   Класс: class functions.LinkedListTabulatedFunction
   Функция: {(0.0; 0.0), (5.0; 25.0), (10.0; 100.0)}

4. Табулирование Sin с использованием LinkedListTabulatedFunction:
   Класс: class functions.LinkedListTabulatedFunction
   Функция: {(0.0; 0.0), (0.3141592653589793; 0.3090169943749474), (0.6283185307
179586; 0.5877852522924731), (0.9424777960769379; 0.8090169943749475), (1.256637
0614359172; 0.9510565162951535), (1.5707963267948966; 1.0), (1.8849555921538759;
 0.9510565162951536), (2.199114857512855; 0.8090169943749475), (2.51327412287183
45; 0.5877852522924732), (2.827433388230814; 0.3090169943749475), (3.14159265358
9793; 1.2246467991473532E-16)}

5. Создание через массивы x и y значений:
   Класс: class functions.ArrayTabulatedFunction
   Функция: {(0.0; 0.0), (1.0; 1.0), (2.0; 4.0), (3.0; 9.0)}

6. Тестирование обработки ошибок:
   Корректно обработана ошибка неправильного класса: Класс java.lang.String не р
еализует интерфейс TabulatedFunction
   Корректно обработана ошибка неправильного количества точек: Ошибка при создан
ии объекта через рефлексию
   Корректно обработана ошибка неправильных границ: Ошибка при создании объекта
через рефлексию
PS D:\Other\Study\GitHubLab7>
```