Лабораторная работа №7

Выполнила: Аксенова Алина
Номер группы: 6204-010302D
Дата выполнения: 2025г.

Оглавление

Задание 1

Я сделала так, чтобы табулированные функции можно было перебирать в цикле for-each. Для этого добавила `Iterable<FunctionPoint>` к интерфейсу `TabulatedFunction`. В классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` написала метод `iterator()`, который возвращает анонимный итератор. Итератор работает быстро, потому что обращается напрямую к внутренним данным - к массиву или узлам списка.

```
@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("Нет следующего элемента");
            }
            // Возвращаем копию для защиты инкапсуляции
            return new FunctionPoint(points[currentIndex++]);
        }

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

Метод `remove()` всегда бросает исключение, как и требовалось. Метод `next()` тоже бросает исключение, если элементов больше нет. Чтобы не нарушать инкапсуляцию, я возвращаю копии точек.

Проверила в main-методе - всё работает, можно перебирать точки функций в цикле for-each.

Задание 2

Создала интерфейс `TabulatedFunctionFactory` с тремя методами для создания функций. Затем сделала фабрики внутри классов `ArrayTabulatedFunction` и `LinkedListTabulatedFunction`:

```
public static class ArrayTabulatedFunctionFactory implements
TabulatedFunctionFactory {
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double
rightX, int pointsCount) {
        return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
    }
    // ... ещё два метода
}
В классе TabulatedFunctions добавила поле для хранения текущей фабрики и метод
для её смены:
java
private static TabulatedFunctionFactory factory =
    new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();

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

Во всех местах, где создавались функции через `new ArrayTabulatedFunction(...)`, заменила на вызовы фабрики `createTabulatedFunction(...)`. Теперь тип создаваемых функций можно менять "на лету".

Протестировала - сначала создаются массивы, потом меняю фабрику и создаются списки, потом можно вернуться обратно к массивам.

Задание 3

Добавила в `TabulatedFunctions` методы, которые создают объекты через рефлексию. Они принимают класс, который нужно создать:

```
public static TabulatedFunction createTabulatedFunction(
    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 e) {
        throw new IllegalArgumentException("Класс " + functionClass.getName() +
                                           " не имеет конструктора (double,
double, int)", e);
    } catch (InstantiationException | IllegalAccessException |
InvocationTargetException e) {
        throw new IllegalArgumentException("Ошибка создания объекта", e);
    }
}
```

Если что-то пойдёт не так (нет такого конструктора или ошибка создания), метод бросает `IllegalArgumentException` с информацией о том, что именно сломалось.

Также сделала перегруженный метод `tabulate()`, который тоже принимает класс создаваемой функции.

Проверила - могу создавать функции любого типа через рефлексию. Если попробовать создать что-то неправильное (например, сам интерфейс), получаю понятное сообщение об ошибке.

Вывод Main:

```
=== ЛАБОРАТОРНАЯ РАБОТА №7 ===

--- ЗАДАНИЕ 1:
```

```
1. Тест ArrayTabulatedFunction:
   Итерация через for-each:
     (0.0; 0.0)
     (1.0; 1.0)
     (2.0; 4.0)

2. Тест LinkedListTabulatedFunction:
   Итерация через for-each:
     (0.0; 0.0)
     (1.0; 1.0)
     (2.0; 4.0)

3. Тест итератора вручную:
     (0.0; 0.0)
     (1.0; 1.0)
     (2.0; 4.0)

4. Тест исключений:
   Корректно: UnsupportedOperationException - Удаление не поддерживается
   Корректно: NoSuchElementException - Нет следующего элемента

--- ЗАДАНИЕ 2:

1. Тест фабрики по умолчанию:
   Тип: ArrayTabulatedFunction

2. Смена фабрики на LinkedListTabulatedFunction:
   Тип: LinkedListTabulatedFunction

3. Возврат к ArrayTabulatedFunction:
   Тип: ArrayTabulatedFunction

--- ЗАДАНИЕ 3:

1. Создание через рефлексию:
   ArrayTabulatedFunction(0, 10, 3):
     Тип: ArrayTabulatedFunction
     Данные: {(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

   ArrayTabulatedFunction(0, 10, [0,5,10]):
     Тип: ArrayTabulatedFunction

   LinkedListTabulatedFunction(points):
     Тип: LinkedListTabulatedFunction

2. Tabulate с рефлексией:
```

```
   Тип: LinkedListTabulatedFunction
   Точек: 5

3. Обработка ошибок:
   Корректно: Класс functions.TabulatedFunction не имеет конструктора
(double, double, int)

 Методы ввода/вывода с рефлексией:
   Чтение в LinkedListTabulatedFunction: успешно

 Тест перегруженных методов записи с рефлексией:
   outputTabulatedFunction с Class: успешно
   writeTabulatedFunction с Class: успешно
   Ошибка при несоответствии типа: Функция должна быть экземпляром
класса functions.LinkedListTabulatedFunction

=== Все задания выполнены ===
```