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: Проверка итератора ---
```

```
1. Тест ArrayTabulatedFunction:
   Итерация по точкам (for-each):
     (0.0; 0.0)
     (1.0; 1.0)
     (2.0; 4.0)
     (3.0; 9.0)

2. Тест LinkedListTabulatedFunction:
   Итерация по точкам (for-each):
     (0.0; 0.0)
     (1.0; 1.0)
     (2.0; 4.0)
     (3.0; 9.0)

3. Тест с явным итератором:
   Точки функции от 0 до 10 (5 точек):
     X: 0.0, Y: 0.0
     X: 2.5, Y: 0.0
     X: 5.0, Y: 0.0
     X: 7.5, Y: 0.0
     X: 10.0, Y: 0.0

4. Тест исключения при удалении:
   Корректно: Удаление не поддерживается

5. Тест исключения при отсутствии следующего элемента:
   Корректно: Нет следующего элемента

--- ЗАДАНИЕ 2: Проверка фабричного метода ---

1. Тест фабрики по умолчанию (ArrayTabulatedFunction):
   Тип функции после tabulate: ArrayTabulatedFunction
   Количество точек: 11

2. Тест смены фабрики на LinkedListTabulatedFunction:
   Тип функции: LinkedListTabulatedFunction

3. Тест смены фабрики обратно на ArrayTabulatedFunction:
   Тип функции: ArrayTabulatedFunction

4. Тест всех методов createTabulatedFunction:
   createTabulatedFunction(0, 10, 5): ArrayTabulatedFunction, точек: 5
   createTabulatedFunction(0, 10, values): ArrayTabulatedFunction, точек:
6
   createTabulatedFunction(points): ArrayTabulatedFunction, точек: 3
```

```
--- ЗАДАНИЕ 3: Проверка рефлексии ---

1. Тест создания через рефлексию:
   createTabulatedFunction(ArrayTabulatedFunction.class, 0, 10, 3):
     Тип: ArrayTabulatedFunction
     Функция: {(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

   createTabulatedFunction(ArrayTabulatedFunction.class, 0, 10,
[0,5,10]):
     Тип: ArrayTabulatedFunction
     Функция: {(0.0; 0.0), (5.0; 5.0), (10.0; 10.0)}

   createTabulatedFunction(LinkedListTabulatedFunction.class, points):
     Тип: LinkedListTabulatedFunction
     Функция: {(0.0; 0.0), (5.0; 25.0), (10.0; 100.0)}

2. Тест tabulate с рефлексией:
   tabulate(LinkedListTabulatedFunction.class, sin, 0, PI, 11):
     Тип: LinkedListTabulatedFunction
     Количество точек: 11
     Первая точка: (0.0; 0.0)
     Последняя точка: (3.141592653589793; 1.2246467991473532E-16)

3. Тест SimpleArrayTabulatedFunction через рефлексию:
   createTabulatedFunction(SimpleArrayTabulatedFunction.class, 0, 5, 4):
     Тип: SimpleArrayTabulatedFunction
     Функция: {(0.0; 0.0), (1.6666666666666667; 0.0),
(3.3333333333333335; 0.0), (5.0; 0.0)}

4. Тест обработки ошибок:
   Корректно обработана ошибка: Класс functions.TabulatedFunction не
имеет конструктора (double, double, int)
   Причина: NoSuchMethodException

5. Тест вызова метода с неправильным количеством аргументов:
   Корректно создана функция: ArrayTabulatedFunction

6. Тест вызова несуществующего метода (через try-catch):

=== Все задания лабораторной работы выполнены! ===
```