Отчет по лабораторной работе №7

Студент: Стоколяс Юрий Юрьевич
Группа: 6201-120303D

Выполненные задания

Задание 1: Реализация паттерна «Итератор»

В интерфейсе `TabulatedFunction` добавлено наследование от `Iterable<FunctionPoint>`, что позволяет использовать объекты табулированных функций в улучшенном цикле for-each.

В классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` реализован метод `iterator()`, возвращающий анонимный класс, реализующий интерфейс `Iterator<FunctionPoint>`.

Итератор для `ArrayTabulatedFunction` использует внутренний массив `points` и индекс `currentIndex` для эффективного доступа к элементам. Итератор для `LinkedListTabulatedFunction` использует внутреннюю структуру связанного списка через поле `head` и навигацию по узлам.

Метод `remove()` в обоих итераторах выбрасывает исключение `UnsupportedOperationException`, как требуется по заданию. Метод `next()` выбрасывает `NoSuchElementException`, если следующего элемента нет. Возвращаемые объекты `FunctionPoint` создаются как копии, что обеспечивает инкапсуляцию.

В методе `main()` добавлен тест `testLab7Task1()`, который проверяет работу итераторов для обоих типов табулированных функций.

Задание 2: Реализация паттерна «Фабричный метод»

В пакете `functions` создан интерфейс `TabulatedFunctionFactory`, объявляющий три перегруженных метода `createTabulatedFunction()` с параметрами, соответствующими конструкторам классов табулированных функций.

В классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` созданы вложенные публичные классы фабрик `ArrayTabulatedFunctionFactory` и `LinkedListTabulatedFunctionFactory`, реализующие интерфейс `TabulatedFunctionFactory`.

В классе `TabulatedFunctions` добавлено приватное статическое поле `factory` типа `TabulatedFunctionFactory`, инициализированное объектом `ArrayTabulatedFunctionFactory`. Также добавлен метод `setTabulatedFunctionFactory()` для замены фабрики.

В классе `TabulatedFunctions` реализованы три перегруженных метода `createTabulatedFunction()`, которые делегируют создание объектов текущей фабрике. Все места в классе, где происходило явное создание объектов табулированных функций через конструкторы, заменены на вызовы соответствующих методов `createTabulatedFunction()`.

В методе `main()` добавлен тест `testLab7Task2()`, который демонстрирует работу фабрик: создает объекты через `tabulate()`, меняет фабрику на `LinkedListTabulatedFunctionFactory`, затем обратно на `ArrayTabulatedFunctionFactory`, и выводит типы созданных объектов.

Задание 3: Использование рефлексии

В классе `TabulatedFunctions` добавлены три перегруженных версии метода `createTabulatedFunction()`, принимающие параметр типа `Class<? extends TabulatedFunction>`. Перед использованием класса проверяется, что он реализует интерфейс `TabulatedFunction` через метод `isAssignableFrom()`.

Методы используют рефлексию для поиска конструктора с соответствующими типами параметров через `getConstructor()`. Создание объекта выполняется через `newInstance()`. Если в ходе рефлексивных операций возникает исключение (не найден конструктор, ошибка создания объекта и т.д.), оно отлавливается и выбрасывается как `IllegalArgumentException` с причиной в виде отловленного исключения.

Также перегружен метод `tabulate()`, который принимает параметр типа `Class<? extends TabulatedFunction>` и использует рефлексивный метод `createTabulatedFunction()` для создания объекта указанного типа.

В методе `main()` добавлен тест `testLab7Task3()`, который проверяет работу рефлексивных методов создания объектов для обоих типов табулированных функций с различными наборами параметров, а также работу перегруженного метода `tabulate()`.

Результаты тестирования

Все три задания успешно реализованы и протестированы. Программа корректно компилируется и выполняется. Итераторы работают для обоих типов табулированных функций, фабрики позволяют динамически менять тип создаваемых объектов, а рефлексия обеспечивает создание объектов по указанному классу.

Фрагмент консольного вывода:

```
=== Задание 1: Итераторы ===
ArrayTabulatedFunction:
(0.0; 0.0)
(5.0; 0.0)
(10.0; 0.0)
LinkedListTabulatedFunction:
(0.0; 0.0)
(5.0; 0.0)
(10.0; 0.0)

=== Задание 2: Фабричный метод ===
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction

=== Задание 3: Рефлексия ===
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),
(0.6283185307179586; 0.5877852522924731),
(0.9424777960769379; 0.8090169943749475),
(1.2566370614359172; 0.9510565162951535),
(1.5707963267948966; 1.0), (1.8849555921538759;
 0.9510565162951535), (2.199094857512855;
 0.8090169943749475), (2.5132541228718345;
 0.5877852522924731), (2.827413388230814;
 0.3090169943749474), (3.141592653589793;
1.2246467991473532E-16)}
```