Лабораторная №7

Болясников Вадим Андреевич 6204-010302

Задание на лабораторную:

Внести изменения в существующий набор типов табулированных функций, позволяющие обрабатывать точки функций по порядку (паттерн «Итератор»), а также выбирать тип объекта табулированной функции при его неявном создании (паттерн «Фабричный метод» и средства рефлексии).

Задание 1

Я сделал так, чтобы все объекты типа TabulatedFunction можно было использовать в качестве объекта-агрегата в «улучшенном цикле for» (вариант for-each). Извлекаемые объекты при этом имеют тип FunctionPoint.

В интерфейс TabulatedFunction добавил родительский тип Iterable<FunctionPoint>, использовав параметризованный тип (generic type). В классах, реализующих интерфейс TabulatedFunction, добавил метод iterator(), возвращающий объект итератора. Классы итераторов сделал анонимными.

Итераторы работают эффективно и используют знание о внутренней структуре объектов:

В ArrayTabulatedFunction итератор напрямую обращается к элементам массива points

В LinkedListTabulatedFunction итератор обходит узлы связного списка

Операцию удаления текущего элемента в итераторах не реализовывал.

Метод remove() всегда выбрасывает исключение UnsupportedOperationException. Метод получения следующего элемента выбрасывает исключение NoSuchElementException, если следующего элемента нет. Возвращаемый методом объект типа FunctionPoint является копией точки, что не позволяет нарушить инкапсуляцию объекта табулированной функции.

В методе main() проверил работу итераторов классов табулированных функций, используя for-each цикл для перебора всех точек функций.

Задание 2

В пакете functions описал базовый интерфейс фабрик табулированных функций TabulatedFunctionFactory. Интерфейс объявляет три перегруженных метода TabulatedFunction createTabulatedFunction(), параметры которых соответствуют параметрам конструкторов классов табулированных функций.

В классах ArrayTabulatedFunction и LinkedListTabulatedFunction создал вложенные публичные классы
фабрик ArrayTabulatedFunctionFactory и LinkedListTabulatedFunctionFactory, реализующие интерфейс фабрики и порождающие объекты соответствующих классов табулированных функций.

В классе TabulatedFunctions объявил приватное статическое поле
типа TabulatedFunctionFactory и проинициализировал его
объектом ArrayTabulatedFunctionFactory. Также объявил
метод setTabulatedFunctionFactory(), позволяющий заменить объект фабрики.

Ещё в классе TabulatedFunctions описал три перегруженных метода TabulatedFunction createTabulatedFunction(), возвращающих объекты табулированных функций, созданные с помощью текущей фабрики. Параметры методов соответствуют параметрам методов фабрики.

В остальных методах класса, где требуется создание объектов табулированных функций, заменил явное создание объектов с помощью конструкторов на вызов соответствующего метода createTabulatedFunction().
В методе main() проверил работу фабрик, изменяя тип создаваемых функций динамически в ходе работы программы.

Задание 3

В классе TabulatedFunctions добавил ещё три перегруженных версии метода createTabulatedFunction(). Их параметры повторяют параметры трёх аналогичных методов, основанных на использовании фабрики, но также эти методы получают ссылку типа Class на описание класса, объект которого требуется создать. Сделал так, чтобы в эти методы можно было передать только ссылки на классы, реализующие интерфейс TabulatedFunction (проверка через TabulatedFunction.class.isAssignableFrom(clazz)).
Новые методы создания объектов находят в предложенном классе конструктор с соответствующими типами параметров с помощью рефлексии (clazz.getConstructor(...)). С помощью найденного конструктора (в него передаются фактические параметры) создаётся объект табулированной функции. Ссылка на этот объект возвращается из метода создания.
Если в ходе выполнения рефлексивных операций возникло исключение (не найден конструктор и т.д.), оно отлавливается (используется блок try с отловом нескольких типов исключений). Вместо него выбрасывается исключение IllegalArgumentException, причём в его конструктор передаётся отловленное исключение из рефлексии. Это позволяет в случае возникновения ошибок определить реальную причину ошибки.
Также перегрузил метод tabulate(), добавив версию, принимающую ссылку типа Class на описание класса, объект которого требуется создать.
В методе main() проверил работу методов рефлексивного создания объектов, а также методов класса TabulatedFunctions, использующих создание объектов через рефлексию. Протестировал корректную обработку ошибок при передаче классов, не реализующих интерфейс TabulatedFunction.

Вывод:

```
=== ЛАБОРАТОРНАЯ РАБОТА №7 ===
=== Тестирование рефлексивного создания объектов ===

1. ArrayTabulatedFunction через границы и количество точек:
   Тип: ArrayTabulatedFunction
   Функция: {(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

2. ArrayTabulatedFunction через границы и значения:
   Тип: ArrayTabulatedFunction
   Функция: {(0.0; 0.0), (5.0; 5.0), (10.0; 10.0)}

3. LinkedListTabulatedFunction через массив точек:
   Тип: LinkedListTabulatedFunction
   Функция: {(0.0; 0.0), (5.0; 25.0), (10.0; 100.0)}

4. Табулирование Sin с LinkedListTabulatedFunction:
   Тип: LinkedListTabulatedFunction
   Функция: {(0.0; 0.0), (0.7853981633974483; 0.7071067811865475),
(1.5707963267948966; 1.0), (2.356194490192345; 0.7071067811865476),
(3.141592653589793; 1.2246467991473532E-16)}

5. Тест ошибки (класс не реализует TabulatedFunction):
   Ожидаемая ошибка: Класс java.lang.String не реализует интерфейс TabulatedFunction
=== Тестирование фабрик табулированных функций ===

1. Фабрика по умолчанию:
   Тип: ArrayTabulatedFunction

2. LinkedList фабрика:
   Тип: LinkedListTabulatedFunction

3. Array фабрика:
   Тип: ArrayTabulatedFunction

4. Тестирование разных методов создания:
   create(0, 10, 5): ArrayTabulatedFunction
   create(0, 10, values): ArrayTabulatedFunction
   create(points): ArrayTabulatedFunction
=== Тестирование итератора ===

Проверка итератора ArrayTabulatedFunction:
(0.0; 0.0)
(1.0; 1.0)
(2.0; 4.0)

Проверка итератора LinkedListTabulatedFunction:
(0.0; 0.0)
(0.0; 0.0)
(1.0; 1.0)
(2.0; 4.0)
```