Лабораторная работа №7.

Внесение изменений в существующий набор типов табулированных функций, позволяющие обрабатывать точки функций по порядку (паттерн «Итератор»), а также выбор типа объекта табулированной функции при его неявном создании (паттерн «Фабричный метод» и средства рефлексии)

Цель: 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).

Филянов Кирилл Владимирович

Содержание

Задание 1: Реализация паттерна "Итератор"

Ход выполнения:

1. Модификация интерфейса TabulatedFunction:

- Добавлен родительский тип Iterable<FunctionPoint>
- Это позволяет использовать объекты TabulatedFunction в цикле for-each

2. Реализация итераторов в классах:

- В классах ArrayTabulatedFunction и LinkedListTabulatedFunction реализованы методы iterator()
- Использованы анонимные классы итераторов
- Итераторы работают напрямую с внутренними структурами данных для эффективности

3. Особенности реализации:

- Метод remove() всегда выбрасывает UnsupportedOperationException
- Метод next() выбрасывает NoSuchElementException при отсутствии следующего элемента
- Возвращаемые объекты FunctionPoint не нарушают инкапсуляцию

Результат:

```
private static void testIterators() {  1 usage
    System.out.println("\n1. Тест ArrayTabulatedFunction:");
    double[] values1 = {1.0, 4.0, 9.0, 16.0};
    TabulatedFunction arrayFunc = TabulatedFunctions.createTabulatedFunction( leftX: 0,  rightX: 3, values1);

    System.out.println("Итерация по ArrayTabulatedFunction:");
    for (FunctionPoint p : arrayFunc) {
        System.out.println(p);
    }
```

```
--- ПРОВЕРКА 1: ИТЕРАТОРЫ TABULATED FUNCTION ---

1. Тест ArrayTabulatedFunction:
Итерация по ArrayTabulatedFunction:
(0,00; 1,00)
(1,00; 4,00)
(2,00; 9,00)
(3,00; 16,00)

2. Тест LinkedListTabulatedFunction:
Итерация по LinkedListTabulatedFunction:
(1,00; 0,50)
(2,00; 1,00)
(3,00; 1,50)
(4,00; 2,00)
```

Задание 2: Реализация паттерна "Фабричный метод"

Ход выполнения:

1. Создание интерфейса фабрики:
- Создан интерфейс TabulatedFunctionFactory с тремя перегруженными методами createTabulatedFunction()

2. Реализация фабрик:
- Созданы вложенные публичные классы фабрик:
- ArrayTabulatedFunctionFactory в классе ArrayTabulatedFunction
- LinkedListTabulatedFunctionFactory в классе LinkedListTabulatedFunction

3. Интеграция фабрик:
- В классе TabulatedFunctions добавлено статическое поле типа TabulatedFunctionFactory
- Реализован метод setTabulatedFunctionFactory() для смены фабрики
- Все методы создания объектов теперь используют текущую фабрику

Результат:

```
private static void testFactories() {  1 usage
    Function f = new Cos();
    TabulatedFunction tf;

    System.out.println("\n1. TabulatedFunctions.tabulate(f, 0, Math.PI, 11)");
    tf = TabulatedFunctions.tabulate(f,  leftX: 0, Math.PI,  pointsCount: 11);
    System.out.println(tf.getClass());

    System.out.println("\n2. После установки LinkedListTabulatedFunctionFactory:");
    TabulatedFunctions.setTabulatedFunctionFactory(new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
    tf = TabulatedFunctions.tabulate(LinkedListTabulatedFunction.class, f,  leftX: 0, Math.PI,  pointsCount: 11);
    System.out.println(tf.getClass());

    System.out.println("\n3. После установки ArrayTabulatedFunctionFactory:");
    TabulatedFunctions.setTabulatedFunctionFactory(new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
    tf = TabulatedFunctions.tabulate(ArrayTabulatedFunction.class, f,  leftX: 0, Math.PI,  pointsCount: 11);
    System.out.println(tf.getClass());
}
```

```
--- ПРОВЕРКА 2: ФАБРИКИ TABULATED FUNCTION ---

1. TabulatedFunctions.tabulate(f, 0, Math.PI, 11)
class functions.tabulated.ArrayTabulatedFunction

2. После установки LinkedListTabulatedFunctionFactory:
class functions.tabulated.LinkedListTabulatedFunction

3. После установки ArrayTabulatedFunctionFactory:
class functions.tabulated.ArrayTabulatedFunction
```

Задание 3: Использование рефлексии

Ход выполнения:

1. Добавление методов с рефлексией:
- В классе TabulatedFunctions добавлены три перегруженных метода createTabulatedFunction(), принимающих параметр типа Class<?>
- Методы проверяют, что переданный класс реализует интерфейс TabulatedFunction

2. Работа с конструкторами:
- Использован метод getConstructor() для поиска конструктора с соответствующими параметрами
- Создание объектов через newInstance() с передачей фактических параметров

3. Обработка исключений:
- Исключения рефлексии отлавливаются и преобразуются в IllegalArgumentException
- Сохраняется цепочка исключений для диагностики

Результат:

```
private static void testReflection() {  1 usage
    TabulatedFunction f;

    System.out.println("\n1. TabulatedFunctions.createTabulatedFunction(" + "ArrayTabulatedFunction.class, 0, 10, 3)");
    f = TabulatedFunctions.createTabulatedFunction(ArrayTabulatedFunction.class,  leftX: 0,  rightX: 10,  pointsCount: 3);
    System.out.println(f.getClass());
    System.out.println(f);

    System.out.println("\n2. TabulatedFunctions.createTabulatedFunction(" + "ArrayTabulatedFunction.class, 0, 10, new double[] {0, 10})"
    f = TabulatedFunctions.createTabulatedFunction(ArrayTabulatedFunction.class,  leftX: 0,  rightX: 10, new double[] {0, 10});
    System.out.println(f.getClass());
    System.out.println(f);

    System.out.println("\n3. TabulatedFunctions.createTabulatedFunction(" + "LinkedListTabulatedFunction.class, new FunctionPoint[] {...
    f = TabulatedFunctions.createTabulatedFunction(LinkedListTabulatedFunction.class, new FunctionPoint[] {new FunctionPoint( x: 0,  y: 0)
    System.out.println(f.getClass());
    System.out.println(f);

    System.out.println("\n4. TabulatedFunctions.tabulate(" + "LinkedListTabulatedFunction.class, new Sin(), 0, Math.PI, 11)");
    f = TabulatedFunctions.tabulate(LinkedListTabulatedFunction.class, new Sin(),  leftX: 0, Math.PI,  pointsCount: 11);
    System.out.println(f.getClass());
    System.out.println(f);
}
```

```
--- ПРОВЕРКА 3: РЕФЛЕКСИВНОЕ СОЗДАНИЕ ---

1. TabulatedFunctions.createTabulatedFunction(ArrayTabulatedFunction.class, 0, 10, 3)
class functions.tabulated.ArrayTabulatedFunction
{(0,00; 0,00), (5,00; 0,00), (10,00; 0,00)}

2. TabulatedFunctions.createTabulatedFunction(ArrayTabulatedFunction.class, 0, 10, new double[] {0, 10})
class functions.tabulated.ArrayTabulatedFunction
{(0,00; 0,00), (10,00; 10,00)}

3. TabulatedFunctions.createTabulatedFunction(LinkedListTabulatedFunction.class, new FunctionPoint[] {...})
class functions.tabulated.LinkedListTabulatedFunction
{(0,00; 0,00), (10,00; 10,00)}

4. TabulatedFunctions.tabulate(LinkedListTabulatedFunction.class, new Sin(), 0, Math.PI, 11)
class functions.tabulated.LinkedListTabulatedFunction
{(0,00; 0,00), (0,31; 0,31), (0,63; 0,59), (0,94; 0,81), (1,26; 0,95), (1,57; 1,00), (1,88; 0,95), (2,20; 0,81), (2,51; 0,59), (2,83; 0,31), (3,14; 0,00)}
```

Перегруженные методы сериализации:

```
1. ТЕСТ ОБРАТНОЙ СОВМЕСТИМОСТИ:
================================================
Записано (старый формат): 5 0.0 0.0 1.0 1.0 2.0 4.0 3.0 9.0 4.0 16.0
Восстановлен класс: ArrayTabulatedFunction
Функции равны: true
Количество точек: 5

2. ТЕСТ С ФАБРИКОЙ:
==================
Записано (с фабрикой): 5 0.0 0.0 1.0 1.0 2.0 4.0 3.0 9.0 4.0 16.0
Восстановлен класс: LinkedListTabulatedFunction
Функции равны: true
✓ Это действительно LinkedListTabulatedFunction!
```

```
3. Array С УКАЗАНИЕМ КЛАССА:
========================================
Записано (с классом): functions.tabulated.ArrayTabulatedFunction 5 0.0 0.0 1.0 1.0 2.0 4.0 3.0 9.0 4.0 16.0
✓ Формат содержит полное имя класса!
Восстановлен класс: functions.tabulated.ArrayTabulatedFunction
Функции равны: true

4. LinkedList С УКАЗАНИЕМ КЛАССА:
==================================
Записано LinkedList (с классом): functions.tabulated.LinkedListTabulatedFunction 5 0.0 0.0 1.0 1.0 2.0 4.0 3.0 9.0 4.0 16.0
✓ Формат содержит полное имя класса LinkedListTabulatedFunction!
Восстановлен класс: functions.tabulated.LinkedListTabulatedFunction
Функции равны: true

5. ПРОВЕРКА ОСОБЕННОСТЕЙ LinkedList:
====================================
ArrayTabulatedFunction toString(): {(0,00; 0,00), (1,00; 1,00), (2,00; 4,00), (3,00; 9,00), (4,00; 16,00)}
LinkedListTabulatedFunction toString(): {(0,00; 0,00), (1,00; 1,00), (2,00; 4,00), (3,00; 9,00), (4,00; 16,00)}

Проверка интерполяции:
ArrayTabulatedFunction.getFunctionValue(2,5) = 6,5000
LinkedListTabulatedFunction.getFunctionValue(2,5) = 6,5000
Значения равны ✓ (должны быть равны)
```

```
6. ПРОВЕРКА СОДЕРЖИМОГО:
========================
Исходная функция (Array):
  [0] (0,0, 0,0)
  [1] (1,0, 1,0)
  [2] (2,0, 4,0)
  [3] (3,0, 9,0)
  [4] (4,0, 16,0)

Восстановленная функция (Array):
  [0] (0,0, 0,0)
  [1] (1,0, 1,0)
  [2] (2,0, 4,0)
  [3] (3,0, 9,0)
  [4] (4,0, 16,0)

Исходная функция (LinkedList):
  [0] (0,0, 0,0)
  [1] (1,0, 1,0)
  [2] (2,0, 4,0)
  [3] (3,0, 9,0)
  [4] (4,0, 16,0)
```

```
Восстановленная функция (LinkedList):
  [0] (0,0, 0,0)
  [1] (1,0, 1,0)
  [2] (2,0, 4,0)
  [3] (3,0, 9,0)
  [4] (4,0, 16,0)

7. ТЕСТ РАЗНЫХ ФАБРИК:
=======================================
1. Array → Array фабрика: ✓ Успешно
2. LinkedList → LinkedList фабрика: ✓ Успешно
3. Array → LinkedList фабрика: ✓ Успешно (преобразовано в LinkedList)
4. LinkedList → Array фабрика: ✓ Успешно (преобразовано в Array)
5. LinkedList → запись через Array фабрику → чтение через Array фабрику: ✓ Успешно
6. Array → запись через LinkedList фабрику → чтение через LinkedList фабрику: ✓ Успешно

8. ПРОВЕРКА НЕСОВМЕСТИМЫХ ФОРМАТОВ:
===================================
✓ Ожидаемая ошибка при несовместимом формате: Некорректное количество точек: functions.tabulated.LinkedListTabulatedFunction

Process finished with exit code 0
```