

Лабораторная работа №7 по дисциплине «Объектно-ориентированное программирование»

Выполнил: Чичкин Данила Александрович

Группа: 6204-010302D

Оглавление

Задание 1 3

Задание 2 5

Задание 3 7

Задание 1

В интерфейс `TabulatedFunction` было добавлено наследование от `Iterable<FunctionPoint>`, что позволяет использовать объекты этого типа в цикле `foreach`. Затем в классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` я реализовал метод `iterator()`, который возвращает анонимный объект итератора (см. скрины 1, 2). Итератор работает напрямую с внутренними структурами данных, не вызывая публичные методы доступа к точкам, что соответствует паттерну «Итератор» и повышает эффективность. Метод `next()` возвращает копию текущей точки, чтобы не нарушить инкапсуляцию объекта функции. Если следующего элемента нет, выбрасывается исключение `NoSuchElementException`. Метод `remove()` не поддерживается и всегда выбрасывает `UnsupportedOperationException`.

```
205         @Override
206         public Iterator<FunctionPoint> iterator() {
207             return new Iterator<FunctionPoint>() {
208                 private int index = 0; 2 usages
209                 @Override
210                 public boolean hasNext() {
211                     return index < size;
212                 }
213                 @Override
214                 public FunctionPoint next() {
215                     if (!hasNext()) {
216                         throw new java.util.NoSuchElementException("Нет следующего элемента");
217                     }
218                     return new FunctionPoint(points[index++]);
219                 }
220                 @Override
221                 public void remove() {
222                     throw new UnsupportedOperationException("Удаление не поддерживается");
223                 }
224             };
225         }
```

Скрин 1 – метод `Iterator` класса `ArrayTabulatedFunction`

```
224         @Override
225         public Iterator<FunctionPoint> iterator() {
226             return new Iterator<FunctionPoint>() {
227                 private FunctionNode current = head.next; 3 usages
228                 private int returned = 0; 2 usages
229                 @Override
230                 public boolean hasNext() {
231                     return returned < size;
232                 }
233                 @Override
234                 public FunctionPoint next() {
235                     if (!hasNext()) {
236                         throw new java.util.NoSuchElementException("Нет следующего элемента");
237                     }
238                     FunctionPoint point = new FunctionPoint(current.data);
239                     current = current.next;
240                     returned++;
241                     return point;
242                 }
243                 @Override
244                 public void remove() {
245                     throw new UnsupportedOperationException("Удаление не поддерживается");
246                 }
247             };
248         }
```

Скрин 2 – метод `Iterator` класса `LinkedListTabulatedFunction`

Для проверки работы итератора я написал метод task1, который создаёт объекты ArrayTabulatedFunction и LinkedListTabulatedFunction и выводит все их точки с помощью цикла for-each (см. скрин 3). Точки выводятся в правильном порядке, исключений не возникает (см. скрин 4). Таким образом, реализация итератора корректна.

```
11 public static void task1() { 1 usage
12     TabulatedFunction arrayFunc = new ArrayTabulatedFunction( leftX: 0, rightX: 10, new double[] {0, 1, 4, 9, 16});
13     System.out.println("ArrayTabulatedFunction:");
14     for (FunctionPoint p : arrayFunc) {
15         System.out.println(p);
16     }
17
18     TabulatedFunction listFunc = new LinkedListTabulatedFunction( leftX: 0, rightX: 10, new double[] {0, 1, 4, 9, 16});
19     System.out.println("\nLinkedListTabulatedFunction:");
20     for (FunctionPoint p : listFunc) {
21         System.out.println(p);
22     }
23 }
```

Скрин 3 –метод task1

```
ArrayTabulatedFunction:
(0.0; 0.0)
(2.5; 1.0)
(5.0; 4.0)
(7.5; 9.0)
(10.0; 16.0)

LinkedListTabulatedFunction:
(0.0; 0.0)
(2.5; 1.0)
(5.0; 4.0)
(7.5; 9.0)
(10.0; 16.0)
```

Скрин 4 – результат работы метода task1

Задание 2

Я добавил интерфейс `TabulatedFunctionFactory` с тремя методами `createTabulatedFunction`, которые соответствуют конструкторам классов табулированных функций (скрин 5). Затем в классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` я создал вложенные публичные классы фабрик, которые реализуют этот интерфейс и создают объекты соответствующих типов (см. скрины 6, 7).

```
1 package functions;
2
3 public interface TabulatedFunctionFactory { 4 usages 2 implementations
4     TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount);
5     TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values);
6     TabulatedFunction createTabulatedFunction(FunctionPoint[] points); 4 usages 2 implementations
7 }
```

Скрин 5 – интерфейс `TabulatedFunctionFactory`

```
13 public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory { 2 usages
14     @Override 1 usage
15     public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
16         return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
17     }
18
19     @Override 1 usage
20     public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
21         return new ArrayTabulatedFunction(leftX, rightX, values);
22     }
23
24     @Override 4 usages
25     public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
26         return new ArrayTabulatedFunction(points);
27     }
28 }
29
30 private FunctionPoint[] points; 45 usages
31 private int size; 46 usages
```

Скрин 6 – класс `ArrayTabulatedFunctionFactory`

```
12 public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory { 1 usage
13     @Override 1 usage
14     public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
15         return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
16     }
17
18     @Override 1 usage
19     public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
20         return new LinkedListTabulatedFunction(leftX, rightX, values);
21     }
22
23     @Override 4 usages
24     public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
25         return new LinkedListTabulatedFunction(points);
26     }
27 }
```

Скрин 7 – класс `LinkedListTabulatedFunction`

В классе `TabulatedFunctions` я объявил статическое поле `factory` и инициализировал его фабрикой для `ArrayTabulatedFunction` по умолчанию. Также добавил метод `setTabulatedFunctionFactory` для замены фабрики во время выполнения программы (см. скрин 8). В существующих методах `TabulatedFunctions`, где создаются объекты функций, я заменил прямое создание через конструкторы на вызовы методов фабрики. Теперь методы `tabulate`, `inputTabulatedFunction` и `readTabulatedFunction` используют текущую установленную фабрику для создания объектов.

```
22     private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory(); 7 usages
23
24     private TabulatedFunctions() {}; no usages
25
26     public static void setTabulatedFunctionFactory(TabulatedFunctionFactory newFactory) { 2 usages
27         factory = newFactory;
28     }
29
30     public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) { no usages
31         return factory.createTabulatedFunction(leftX, rightX, pointsCount);
32     }
33
34     public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) { no usages
35         return factory.createTabulatedFunction(leftX, rightX, values);
36     }
37
38     public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) { no usages
39         return factory.createTabulatedFunction(points);
40     }
```

Скрин 8 – методы в классе `TabulatedFunctions`

Для проверки я написал метод `task2`, который создаёт функцию косинуса, табулирует её с разными фабриками и выводит типы созданных объектов (см. скрин 9). Сначала используется фабрика по умолчанию, которая создаёт `ArrayTabulatedFunction`. Потом я устанавливаю фабрику для `LinkedListTabulatedFunction` и создаю ту же функцию — получается связный список. После этого возвращаю фабрику для массива и снова получаю `ArrayTabulatedFunction`. Все три раза функция создаётся корректно, но с разной внутренней реализацией.

```
11     public static void task2() { 1 usage
12         Function f = new Cos();
13         TabulatedFunction tf;
14
15         tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
16         System.out.println("По умолчанию: " + tf.getClass().getSimpleName());
17
18         TabulatedFunctions.setTabulatedFunctionFactory(
19             new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
20         tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
21         System.out.println("После установки LinkedList фабрики: " + tf.getClass().getSimpleName());
22
23         TabulatedFunctions.setTabulatedFunctionFactory(
24             new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
25         tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
26         System.out.println("После установки Array фабрики: " + tf.getClass().getSimpleName());
27     }
```

Скрин 9 – метод `task2`

Результат работы метода:

```
По умолчанию: ArrayTabulatedFunction
После установки LinkedList фабрики: LinkedListTabulatedFunction
После установки Array фабрики: ArrayTabulatedFunction
```

Задание 3

Я добавил в класс `TabulatedFunctions` три новых перегруженных метода `createTabulatedFunction`, которые вместо фабрики используют рефлексию Java. Эти методы принимают первым параметром класс функции, который нужно создать, а остальные параметры такие же, как у конструкторов. Методы проверяют, что переданный класс действительно реализует интерфейс `TabulatedFunction`, иначе выбрасывают исключение (см. скрин 10).

```
43 @ public static TabulatedFunction createTabulatedFunction( no usages
44     Class<?> functionClass, double leftX, double rightX, int pointsCount) {
45     if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
46         throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction");
47     }
48
49     try {
50         Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, int.class);
51         return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);
52     } catch (Exception e) {
53         throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
54     }
55 }
56
57 @ public static TabulatedFunction createTabulatedFunction( no usages
58     Class<?> functionClass, double leftX, double rightX, double[] values) {
59     if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
60         throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction");
61     }
62
63     try {
64         Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, double[].class);
65         return (TabulatedFunction) constructor.newInstance(leftX, rightX, values);
66     } catch (Exception e) {
67         throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
68     }
69 }
70
71 @ public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, FunctionPoint[] points) { no usages
72     if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
73         throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction");
74     }
75
76     try {
77         Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
78         return (TabulatedFunction) constructor.newInstance((Object) points);
79     } catch (Exception e) {
80         throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
81     }
82 }
```

Скрин 10 – три новые перегрузки `createTabulatedFunction`

Внутри методов с помощью рефлексии находится нужный конструктор класса. Например, для создания функции по границам и количеству точек ищется конструктор с параметрами `double`, `double`, `int`. Если конструктор найден, он вызывается с переданными аргументами, и создаётся объект. Если возникает ошибка (например, конструктор не найден), она перехватывается и выбрасывается как `IllegalArgumentException` с сохранением исходной причины.

Также я добавил перегруженный метод `tabulate`, который работает похожим образом: он принимает класс функции, создаёт массив точек, а затем через рефлексию создаёт объект указанного класса (см. скрин 11).

```
115 @ public static TabulatedFunction tabulate( no usages
116     Class<?> functionClass, Function function, double leftX, double rightX, int pointsCount) {
117     if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
118         throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction");
119     }
120
121     if (leftX >= rightX)
122         throw new IllegalArgumentException("Левая граница области определения leftX должна быть строго меньше правой границы rightX");
123     if (pointsCount < 2)
124         throw new IllegalArgumentException("Количество точек pointsCount должно быть не меньше двух");
125     if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder())
126         throw new IllegalArgumentException("Указанные границы для табулирования выходят за область определения функции");
127
128     FunctionPoint[] points = new FunctionPoint[pointsCount];
129     double step = (rightX - leftX) / (pointsCount - 1);
130     for (int i = 0; i < pointsCount - 1; i++) {
131         points[i] = new FunctionPoint(x: leftX + i * step, function.getFunctionValue(x: leftX + i * step));
132     }
133     points[pointsCount - 1] = new FunctionPoint(rightX, function.getFunctionValue(rightX));
134
135     try {
136         Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
137         return (TabulatedFunction) constructor.newInstance((Object) points);
138     } catch (Exception e) {
139         throw new IllegalArgumentException("Ошибка при создании объекта через рефлексию", e);
140     }
141 }
```

Скрин 11 – перегрузка метода `tabulate`

Для проверки я написал метод `task3`, который создаёт функции разных типов через рефлексию (см. скрин 12). Убедился, что `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` создаются корректно, их точки выводятся правильно. Рефлексия позволяет указывать конкретный тип создаваемого объекта прямо при вызове метода, что даёт ещё один способ управления созданием объектов помимо фабрик.

```
11 public static void task3() { 1 usage
12     System.out.println("\n=== Проверка рефлексии ===");
13
14     TabulatedFunction f;
15
16     f = TabulatedFunctions.createTabulatedFunction(ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3);
17     System.out.println("Создан через рефлексию: " + f.getClass().getSimpleName());
18     System.out.println("Точки: " + f);
19
20     f = TabulatedFunctions.createTabulatedFunction(
21         LinkedListTabulatedFunction.class,
22         new FunctionPoint[] {
23             new FunctionPoint(x: 0, y: 0),
24             new FunctionPoint(x: 5, y: 25),
25             new FunctionPoint(x: 10, y: 100)
26         }
27     );
28     System.out.println("\nСоздан через рефлексию: " + f.getClass().getSimpleName());
29     System.out.println("Точки: " + f);
30
31     f = TabulatedFunctions.tabulate(LinkedListTabulatedFunction.class, new Sin(), leftX: 0, Math.PI, pointsCount: 5);
32     System.out.println("\nТабулирован через рефлексию: " + f.getClass().getSimpleName());
33     System.out.println("Точки: " + f);
34 }
```

Скрин 12 – метод `task3`

Результат работы метода:

```
=== Проверка рефлексии ===  
Создан через рефлексия: ArrayTabulatedFunction  
Точки: {(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}  
  
Создан через рефлексия: LinkedListTabulatedFunction  
Точки: {(0.0; 0.0), (5.0; 25.0), (10.0; 100.0)}  
  
Табулирован через рефлексия: LinkedListTabulatedFunction  
Точки: {(0.0; 0.0), (0.7853981633974483; 0.7071067811865475),  
(1.5707963267948966; 1.0), (2.356194490192345;  
0.7071067811865476), (3.141592653589793; 1.2246467991473532E-16)}
```