Лабораторная работа №7

Выполнила: Сингатуллина Алина Марсовна

Группа 6204-010302D

Оглавление

Задание 1

В интерфейс TabulatedFunction добавила наследование от Iterable<FunctionPoint>:

```
public interface TabulatedFunction extends Function, Cloneable,
Iterable<FunctionPoint> {
```

В классе ArrayTabulatedFunction реализовала метод iterator():

```
@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 java.util.NoSuchElementException("No more
elements");
            }
                        FunctionPoint original = points[currentIndex];
            FunctionPoint copy = new FunctionPoint(original.getX(),
original.getY());
            currentIndex++;
            return copy;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("Remove operation is not
supported");
        }
    };
}
```

В классе LinkedListTabulatedFunction реализовала метод iterator():

```
@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private FunctionNode currentNode = head.next;
        @Override
        public boolean hasNext() {
            return currentNode != head;         }

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new java.util.NoSuchElementException("No more
elements");
            }

            FunctionPoint original = currentNode.point;
            FunctionPoint copy = new FunctionPoint(original.getX(),
original.getY());
            currentNode = currentNode.next;
            return copy;
```

```
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("Remove operation is not
supported");
        }
    };
}
```

Вывод:

```
Тестирование итераторов
ArrayTabulatedFunction:
  (0.0; 0.0)
  (1.0; 1.0)
  (2.0; 4.0)
  (3.0; 9.0)
  (4.0; 16.0)
LinkedListTabulatedFunction:
  (0.0; 0.0)
  (1.0; 1.0)
  (2.0; 4.0)
  (3.0; 9.0)
  (4.0; 16.0)
Явный итератор:
  (0.0; 0.0)
  (1.0; 1.0)
  (2.0; 4.0)
  (3.0; 9.0)
  (4.0; 16.0)
```

Задание 2

В пакете functions описала базовый интерфейс фабрик табулированных функций TabulatedFunctionFactory.

TabulatedFunctionFactory.java

```
package functions;

public interface TabulatedFunctionFactory {
    TabulatedFunction createTabulatedFunction(double leftX, double rightX,
int pointsCount);
    TabulatedFunction createTabulatedFunction(double leftX, double rightX,
double[] values);
    TabulatedFunction createTabulatedFunction(FunctionPoint[] points);
}
```

В классе TabulatedFunctions объявила приватное статическое поле типа TabulatedFunctionFactory. Также объявила метод setTabulatedFunctionFactory(), позволяющий заменить объект фабрики.

```
private static TabulatedFunctionFactory factory =
      new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();

public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) {
   TabulatedFunctions.factory = factory;
}
```

В классе TabulatedFunctions описала три перегруженных метода TabulatedFunction createTabulatedFunction().

```
public static TabulatedFunction createTabulatedFunction(double leftX, double
rightX, int pointsCount) {
    return factory.createTabulatedFunction(leftX, rightX, pointsCount);
}

public static TabulatedFunction createTabulatedFunction(double leftX, double
rightX, double[] values) {
    return factory.createTabulatedFunction(leftX, rightX, values);
}

public static TabulatedFunction createTabulatedFunction(FunctionPoint[]
points) {
    return factory.createTabulatedFunction(points);
}
```

Вывод:

```
Тестирование фабрик
Фабрика по умолчанию (Array):
  Функция: {(0.0; 1.0), (0.7853981633974483; 0.7071067811865476), (1.5707963267948966;
6.123233995736766E-17), (2.356194490192345; -0.7071067811865475), (3.141592653589793;
-1.0)}
Установка LinkedList фабрики:
  Функция: {(0.0; 1.0), (0.7853981633974483; 0.7071067811865476), (1.5707963267948966;
6.123233995736766E-17), (2.356194490192345; -0.7071067811865475), (3.141592653589793;
-1.0)}
Возврат к Array фабрике:
  Функция: {(0.0; 1.0), (0.7853981633974483; 0.7071067811865476), (1.5707963267948966;
6.123233995736766E-17), (2.356194490192345; -0.7071067811865475), (3.141592653589793;
-1.0)}
```

Задание 3

В классе TabulatedFunctions добавила методы с рефлексией.

```
public static TabulatedFunction createTabulatedFunction(Class<?>
functionClass,
                                                         double leftX, double
rightX, int pointsCount) {
    if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
        throw new IllegalArgumentException("Class must implement
TabulatedFunction interface");
    }

    try {
        Constructor<?> constructor =
functionClass.getConstructor(double.class, double.class, int.class);
        return (TabulatedFunction) constructor.newInstance(leftX, rightX,
```

```
pointsCount);
    } catch (Exception e) {
        throw new IllegalArgumentException("Error creating tabulated
function", e);
    }
}

public static TabulatedFunction createTabulatedFunction(Class<?>
functionClass,
                                                         double leftX, double
rightX, double[] values) {
    if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
        throw new IllegalArgumentException("Class must implement
TabulatedFunction interface");
    }

    try {
        Constructor<?> constructor =
functionClass.getConstructor(double.class, double.class, double[].class);
        return (TabulatedFunction) constructor.newInstance(leftX, rightX,
values);
    } catch (Exception e) {
        throw new IllegalArgumentException("Error creating tabulated
function", e);
    }
}

public static TabulatedFunction createTabulatedFunction(Class<?>
functionClass, FunctionPoint[] points) {
    if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
        throw new IllegalArgumentException("Class must implement
TabulatedFunction interface");
    }

    try {
        Constructor<?> constructor =
functionClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);
    } catch (Exception e) {
        throw new IllegalArgumentException("Error creating tabulated
function", e);
    }
}
```

Перегрузила метод tabulate() с рефлексией.

```
public static TabulatedFunction tabulate(Class<?> functionClass, Function
function,
                                         double leftX, double rightX, int
pointsCount) {
    if (leftX < function.getLeftDomainBorder() || rightX >
function.getRightDomainBorder()) {
        throw new IllegalArgumentException("Границы выходят за область
определения функции");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не
менее 2");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше
правой");
    }
```

```
    TabulatedFunction tabulatedFunc = createTabulatedFunction(functionClass,
leftX, rightX, pointsCount);

    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        double y = function.getFunctionValue(x);
        tabulatedFunc.setPointY(i, y);
    }

    return tabulatedFunc;
}
```

Перегрузила методы чтения с помощью рефлексии:

```
public static TabulatedFunction inputTabulatedFunction(Class<?>
functionClass, InputStream in) throws IOException {
    DataInputStream dis = new DataInputStream(in);
    int pointsCount = dis.readInt();
    FunctionPoint[] points = new FunctionPoint[pointsCount];

    for (int i = 0; i < pointsCount; i++) {
        double x = dis.readDouble();
        double y = dis.readDouble();
        points[i] = new FunctionPoint(x, y);
    }

    return createTabulatedFunction(functionClass, points);
}

public static TabulatedFunction readTabulatedFunction(Class<?> functionClass,
Reader in) throws IOException {
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    tokenizer.parseNumbers();

    tokenizer.nextToken();
    int pointsCount = (int) tokenizer.nval;
    FunctionPoint[] points = new FunctionPoint[pointsCount];

    for (int i = 0; i < pointsCount; i++) {
        tokenizer.nextToken();
        double x = tokenizer.nval;
        tokenizer.nextToken();
        double y = tokenizer.nval;
        points[i] = new FunctionPoint(x, y);
    }

    return createTabulatedFunction(functionClass, points);
}
```

Вывод:

```
Тестирование рефлексии
Тестирование рефлексии
1. Создание ArrayTabulatedFunction через рефлексию:
  Класс: ArrayTabulatedFunction
  Значения: {(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}
2. Создание ArrayTabulatedFunction через рефлексию:
  Класс: ArrayTabulatedFunction
```

Значения: {(0.0; 0.0), (10.0; 10.0)}
3. Создание через рефлексию из массива точек:
Класс: LinkedListTabulatedFunction
Значения: {(0.0; 0.0), (10.0; 10.0)}
4. Табулирование с рефлексией:
Класс: LinkedListTabulatedFunction
Значения: {(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; 0.9510565162951536), (2.199114857512855; 0.8090169943749475), (2.5132741228718345; 0.5877852522924732), (2.827433388230814; 0.3090169943749475), (3.141592653589793; 1.2246467991473532E-16)}