

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЁВА»
КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ
«Объектно-ориентированное программирование»
ЛАБОРАТОРНАЯ РАБОТА №7

Студент _____ Чячяков А.И.

Группа _____ 6301-030301D

Руководитель _____ Борисов Д. С.

Оценка

Задание 1

Сделал так, чтобы все объекты типа TabulatedFunction можно было использовать в качестве объекта-агрегата в for-each, извлекаемые объекты при этом должны иметь тип FunctionPoint.

В интерфейсе TabulatedFunction добавил необходимый родительский тип, использовал при этом параметризованный тип (generic type).

В классах, реализующих интерфейс TabulatedFunction, добавил требующийся метод, возвращающий объект итератора.

Классы итераторов сделал анонимными. В соответствии с паттерном «Итератор».

Метод получения следующего элемента должен выбрасывать исключение NoSuchElementException, если следующего элемента нет. Возвращаемый методом объект типа FunctionPoint не должен позволять нарушить инкапсуляцию объекта табулированной функции.

Код 1.1

```
4
5 public interface TabulatedFunction extends Iterable<FunctionPoint> { 13 usages 2 implementations
6     double getLeftDomainBorder(); 2 implementations
7
8     double getRightDomainBorder(); 2 implementations
9
10    double getFunctionValue(double x); 2 implementations
11
12    int getPointsCount(); 6 usages 2 implementations
13
14    FunctionPoint getPoint(int index); no usages 2 implementations
15
16    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
17
18    double getPointX(int index); 5 usages 2 implementations
19
20    void setPointX(int index, double x) throws InappropriateFunctionPointException; no usages
21
22    double getPointY(int index); 5 usages 2 implementations
23
24    void setPointY(int index, double y); no usages 2 implementations
25
26    void deletePoint(int index); no usages 2 implementations
27
28    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; no usages
29
30    @Override 2 implementations
31    String toString();
32
33    @Override 2 implementations
34    boolean equals(Object o);
35
36    @Override 2 implementations
37    int hashCode();
38
39    Object clone(); 2 implementations
40 }
```

Код 1.2

```
@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private int currentIndex = 0; 2 usages

        @Override
        public boolean hasNext() {
            return currentIndex < pointsCount;
        }

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Больше элементов нет");
            }
            return new FunctionPoint(points[currentIndex++]);
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("Удаление не поддерживается");
        }
    };
};
```

Код 1.3

```
11 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable, Cloneable { 4 u
12     public Object clone() {
13         node.point = new FunctionPoint(current.point);
14         current = current.next;
15     }
16     return copy;
17 }
18
19 @Override
20 public java.util.Iterator<FunctionPoint> iterator() {
21     return new java.util.Iterator<FunctionPoint>() {
22         private FunctionNode current = head.next; 4 usages
23
24         @Override
25         public boolean hasNext() {
26             return current != head;
27         }
28
29         @Override
30         public FunctionPoint next() {
31             if (!hasNext()) {
32                 throw new java.util.NoSuchElementException("Больше элементов нет");
33             }
34             FunctionPoint p = new FunctionPoint(current.point);
35             current = current.next;
36             return p;
37         }
38
39         @Override
40         public void remove() {
41             throw new UnsupportedOperationException("Удаление не поддерживается");
42         }
43     };
44 }
```

Код 1.4

```

public class Main {
    private static void complicatedThreads() { 1 usage
        Thread.currentThread().interrupt();
    }

    System.out.println("Оба потока завершили работу (прерваны)");
}

private static void testIterators() { 1 usage
    double[] values = {0, 0.5, 1.0, 1.5, 2.0};
    TabulatedFunction arrayFunc = new ArrayTabulatedFunction( leftX: 0, rightX: 4, values);
    System.out.println("ArrayTabulatedFunction c iterator:");
    for (FunctionPoint p : arrayFunc) {
        System.out.println(p);
    }

    TabulatedFunction listFunc = new LinkedListTabulatedFunction( leftX: 0, rightX: 4, values);
    System.out.println("\nLinkedListTabulatedFunction c iterator:");
    for (FunctionPoint p : listFunc) {
        System.out.println(p);
    }

    System.out.println("\nПроверка UnsupportedOperationException при удалении:");
    try {
        Iterator<FunctionPoint> it = arrayFunc.iterator();
        if (it.hasNext()) {
            it.next();
            it.remove();
        }
    } catch (UnsupportedOperationException e) {
        System.out.println("Поймано: " + e.getMessage());
    }

    System.out.println("\nПроверка NoSuchElementException при исчерпании:");
    try {
        Iterator<FunctionPoint> it = arrayFunc.iterator();
        while (it.hasNext()) {
            it.next();
        }
        it.next();
    } catch (NoSuchElementException e) {
        System.out.println("Поймано: " + e.getMessage());
    }
}
}

```

Вывод:

```
Итераторы
ArrayTabulatedFunction с iterator:
(0.0; 0.0)
(1.0; 0.5)
(2.0; 1.0)
(3.0; 1.5)
(4.0; 2.0)

LinkedListTabulatedFunction с iterator:
(0.0; 0.0)
(1.0; 0.5)
(2.0; 1.0)
(3.0; 1.5)
(4.0; 2.0)

Проверка UnsupportedOperationException при удалении:
Поймано: Удаление не поддерживается

Проверка NoSuchElementException при исчерпании:
Поймано: Больше элементов нет

Process finished with exit code 0
```

Задание 2

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

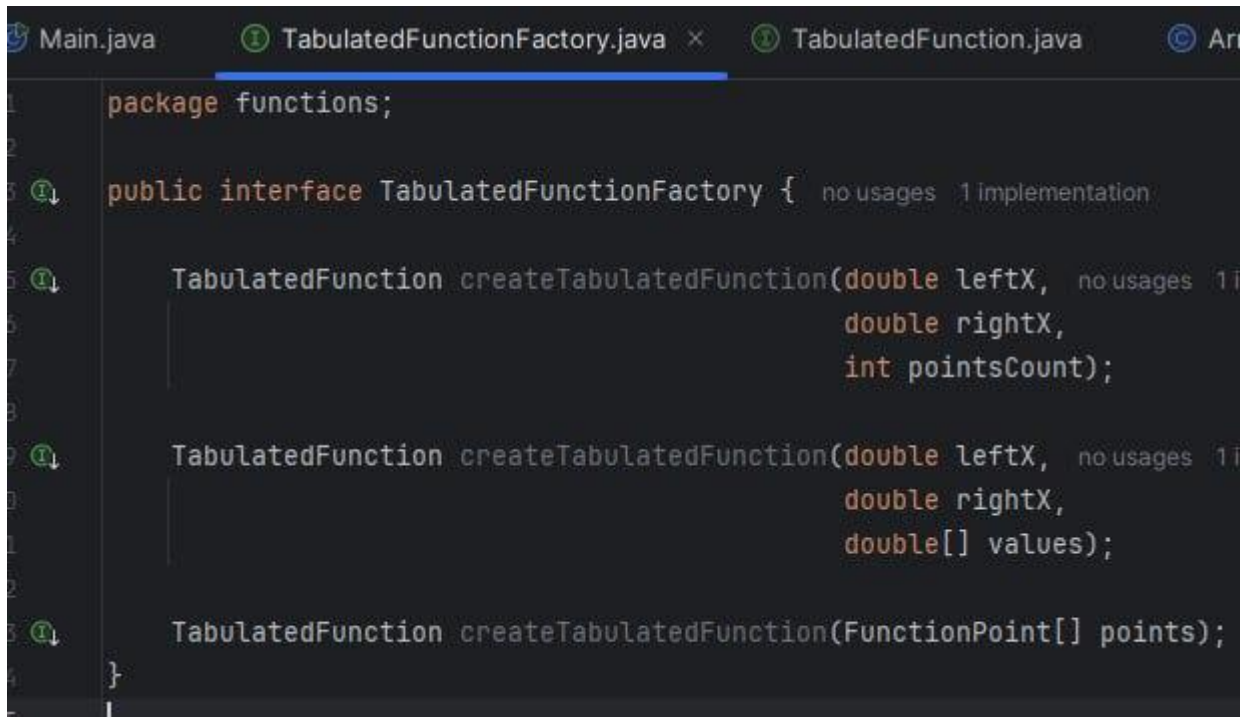
Для удобства опишу в классах ArrayTabulatedFunction и LinkedListTabulatedFunction классы фабрик ArrayTabulatedFunctionFactory и LinkedListTabulatedFunctionFactory, реализующие интерфейс фабрики и порождающие объекты соответствующих классов табулированных функций. Сделал классы фабрик вложенными и публичными.

В классе TabulatedFunctions объявил приватное статическое поле типа TabulatedFunctionFactory и проинициализировал его объектом одного из описанных классов фабрик. 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 должны соответствовать 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()`.

Код 2.1



```
1 package functions;
2
3 public interface TabulatedFunctionFactory { no usages 1 implementation
4
5     TabulatedFunction createTabulatedFunction(double leftX, no usages 1 i
6         double rightX,
7         int pointsCount);
8
9     TabulatedFunction createTabulatedFunction(double leftX, no usages 1 i
10        double rightX,
11        double[] values);
12
13     TabulatedFunction createTabulatedFunction(FunctionPoint[] points);
14 }
15
```

Код 2.2


```

Main.java  TabulatedFunctionFactory.java  TabulatedFunction.java  ArrayTabulatedFunction.java x
8      public class ArrayTabulatedFunction implements TabulatedFunction, Serializable, Cloneable {
310
311      public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {
312
313          @Override no usages
314      public TabulatedFunction createTabulatedFunction(double leftX,
315                                                         double rightX,
316                                                         int pointsCount) {
317          return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
318      }
319
320          @Override no usages
321      public TabulatedFunction createTabulatedFunction(double leftX,
322                                                         double rightX,
323                                                         double[] values) {
324          return new ArrayTabulatedFunction(leftX, rightX, values);
325      }
326
327          @Override no usages
328      public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
329          return new ArrayTabulatedFunction(points);
330      }
331  }

```

Код 2.3

```

Main.java  TabulatedFunctionFactory.java  TabulatedFunction.java  ArrayTabulatedFunction.java  LinkedListTabulatedFunction.java x
11     public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable, Cloneable { 8 usages
432     }
433
434     public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory { no usages
435
436         @Override no usages
437     public TabulatedFunction createTabulatedFunction(double leftX,
438                                                         double rightX,
439                                                         int pointsCount) {
440         return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
441     }
442
443         @Override no usages
444     public TabulatedFunction createTabulatedFunction(double leftX,
445                                                         double rightX,
446                                                         double[] values) {
447         return new LinkedListTabulatedFunction(leftX, rightX, values);
448     }
449
450         @Override no usages
451     public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
452         return new LinkedListTabulatedFunction(points);
453     }
454     }
455 }
456

```

Код 2.4

```

public class TabulatedFunctions { no usages
}

private static TabulatedFunctionFactory factory = 4 usages
    new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();

public static void setTabulatedFunctionFactory(TabulatedFunctionFactory newFactory) { no usages
    if (newFactory == null) {
        throw new IllegalArgumentException("Фабрика не может быть null");
    }
    factory = newFactory;
}

public static TabulatedFunction createTabulatedFunction(double leftX, no usages
    double rightX,
    int pointsCount) {
    return factory.createTabulatedFunction(leftX, rightX, pointsCount);
}

public static TabulatedFunction createTabulatedFunction(double leftX, 1 usage
    double rightX,
    double[] values) {
    return factory.createTabulatedFunction(leftX, rightX, values);
}

public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) { 2 usages
    return factory.createTabulatedFunction(points);
}

public static TabulatedFunction tabulate(Function function, no usages
    double leftX,
    double rightX,
    int pointsCount) {
    if (leftX < function.getLeftDomainBorder()
        || rightX > function.getRightDomainBorder()
        || leftX >= rightX
        || pointsCount < 2) {
        throw new IllegalArgumentException("Некорректные границы табулирования или количество точек");
    }

    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    double x = leftX;

```

Код 2.5


```

public class Main {
}

private static void testFactories() { 1 usage
    System.out.println("\nПроверка фабрик TabulatedFunctionFactory:");
    Function f = new Cos();
    TabulatedFunction tf;

    tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
    System.out.println("Текущая фабрика, класс: " + tf.getClass().getName());

    TabulatedFunctions.setTabulatedFunctionFactory(
        new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
    tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
    System.out.println("После установки LinkedList фабрики, класс: " + tf.getClass().getName());

    TabulatedFunctions.setTabulatedFunctionFactory(
        new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
    tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
    System.out.println("После возврата Array фабрики, класс: " + tf.getClass().getName());
}

```

Вывод:

```

Фабрики табулированных функций

Проверка фабрик TabulatedFunctionFactory:
Текущая фабрика, класс: functions.ArrayTabulatedFunction
После установки LinkedList фабрики, класс: functions.LinkedListTabulatedFunction
После возврата Array фабрики, класс: functions.ArrayTabulatedFunction

Process finished with exit code 0

```

Задание 3

В классе TabulatedFunctions добавил ещё три перегруженных версии метода createTabulatedFunction(). Их параметры должны повторять параметры трёх аналогичных методов, основанных на использовании фабрики, но также эти методы должны получать 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.

Новые методы создания объектов должны найти в предложенном классе конструктор с соответствующими типами параметров. С помощью найденного конструктора (в него должны быть переданы фактические параметры) должен быть создан объект табулированной функции. Ссылка на этот объект и должна быть возвращена из метода создания.

В классе TabulatedFunctions перегрузил методы, создающие объекты табулированных функций, добавил версии, принимающие также 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.

Код 3.1

```
5 public class TabulatedFunctions { 5 usages
42 public static TabulatedFunction createTabulatedFunction(Class<? extends TabulatedFunction> clazz, 1 usage
51     throw new IllegalArgumentException("Не удалось создать объект класса " + clazz.getName(), e);
52 }
53 }
54
55 @ public static TabulatedFunction createTabulatedFunction(Class<? extends TabulatedFunction> clazz, no usages
56     FunctionPoint[] points) {
57     try {
58         var constructor = clazz.getConstructor(FunctionPoint[].class);
59         return constructor.newInstance((Object) points);
60     } catch (NoSuchMethodException | InstantiationException | IllegalAccessException |
61         java.lang.reflect.InvocationTargetException e) {
62         throw new IllegalArgumentException("Не удалось создать объект класса " + clazz.getName(), e);
63     }
64 }
65
66 @ public static TabulatedFunction tabulate(Class<? extends TabulatedFunction> clazz, no usages
67     Function function,
68     double leftX,
69     double rightX,
70     int pointsCount) {
71     if (leftX < function.getLeftDomainBorder()
72         || rightX > function.getRightDomainBorder()
73         || leftX >= rightX
74         || pointsCount < 2) {
75         throw new IllegalArgumentException("Некорректные границы табулирования или количество точек");
76     }
77
78     double[] values = new double[pointsCount];
79     double step = (rightX - leftX) / (pointsCount - 1);
80     double x = leftX;
81
82     for (int i = 0; i < pointsCount; i++) {
83         values[i] = function.getFunctionValue(x);
84         x += step;
85     }
86
87     return createTabulatedFunction(clazz, leftX, rightX, values);
88 }
```

Код 3.2

```

0 public class Main {
1     private static void testReflection() { 1 usage
2         System.out.println("\nТестирование рефлексивного создания объектов:");
3
4         TabulatedFunction f;
5
6         f = TabulatedFunctions.createTabulatedFunction(
7             ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3);
8         System.out.println("Класс: " + f.getClass().getName());
9         System.out.println(f);
10
11        f = TabulatedFunctions.createTabulatedFunction(
12            ArrayTabulatedFunction.class, leftX: 0, rightX: 10, new double[]{0, 5, 10});
13        System.out.println("\nКласс: " + f.getClass().getName());
14        System.out.println(f);
15
16        f = TabulatedFunctions.createTabulatedFunction(
17            LinkedListTabulatedFunction.class,
18            new FunctionPoint[]{
19                new FunctionPoint(x: 0, y: 0),
20                new FunctionPoint(x: 10, y: 10)
21            }
22        );
23        System.out.println("\nКласс: " + f.getClass().getName());
24        System.out.println(f);
25
26        f = TabulatedFunctions.tabulate(
27            LinkedListTabulatedFunction.class, new functions.basic.Sin(), leftX: 0, Math.PI, pointsCount: 11);
28        System.out.println("\nКласс: " + f.getClass().getName());
29        System.out.println(f);
30    }

```

Вывод:

```

Рефлексия
Тестирование рефлексивного создания объектов:
Класс: functions.ArrayTabulatedFunction
{(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

Класс: functions.ArrayTabulatedFunction
{(0.0; 0.0), (5.0; 5.0), (10.0; 10.0)}

Класс: functions.LinkedListTabulatedFunction
{(0.0; 0.0), (10.0; 10.0)}

Класс: 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), (1.8849555921539178; 0.9510565162951535), (2.1991148575124575; 0.8090169943749475), (2.5132741268717948; 0.5877852522924731), (2.827433395731975; 0.3090169943749474), (3.141592653589793; 0.0)}

Process finished with exit code 0

```