Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет имени
академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №7

Дисциплина: «ООП»

Тема «Внесение изменений в существующий набор типов табулированных функций»

Выполнил: Куликов Степан
Дмитриевич

Группа: 6201-120303D

Самара, 2025

Задание на лабораторную работу

Задание 1

Сделать объекты TabulatedFunction итерируемыми (Iterable<FunctionPoint>) для использования в for-each. Итератор должен быть анонимным, возвращать FunctionPoint, выбрасывать NoSuchElementException при окончании, UnsupportedOperationException при удалении, и не нарушать инкапсуляцию. Проверить в main().

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

```
public Iterator<FunctionPoint> iterator() {
return new Iterator<FunctionPoint>() {
private int index = 0;

    public boolean hasNext() {
return index < pointsCount;
    }

    public FunctionPoint next() {          if
(!hasNext()) {              throw new
NoSuchElementException();
        }
        FunctionPoint p = point[index++];
return new FunctionPoint(p.getX(), p.getY());
    }

    public void remove() {          throw new
UnsupportedOperationException();
    }
  };
}
```

```
public Iterator<FunctionPoint> iterator() {
return new Iterator<FunctionPoint>() {

    private FunctionNode current = head.next;

    public boolean hasNext() {
```

```
            return current != head;
        }

        public FunctionPoint next() {            if
(!hasNext()) {                throw new
NoSuchElementException();
            }

            FunctionPoint p = current.data;
current = current.next;

            return new FunctionPoint(p.getX(), p.getY());
        }

        public void remove() {            throw new
UnsupportedOperationException();
        }
    }; }


for (FunctionPoint p : f) {
    System.out.println(p);
}
```

Задание 2

Реализовать фабрику табулированных функций (TabulatedFunctionFactory) с методами createTabulatedFunction(). Сделать вложенные фабрики для ArrayTabulatedFunction и LinkedListTabulatedFunction. В TabulatedFunctions хранить текущую фабрику и методы для создания объектов через неё. Проверить работу фабрик в main().

```
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);
}
public static class ArrayTabulatedFunctionFactory implements
TabulatedFunctionFactory {

    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
int pointsCount) {        return new ArrayTabulatedFunction(leftX, rightX,
pointsCount);
    }

    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
double[] values) {        return new ArrayTabulatedFunction(leftX, rightX, values);
    }

    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
return new ArrayTabulatedFunction(points);
    }
}
public static class LinkedListTabulatedFunctionFactory implements
TabulatedFunctionFactory {

    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
int pointsCount) {        return new LinkedListTabulatedFunction(leftX, rightX,
pointsCount);
    }
```

```
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
double[] values) {         return new LinkedListTabulatedFunction(leftX, rightX,
values);
    }

    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
return new LinkedListTabulatedFunction(points);
    }
}
private static TabulatedFunctionFactory factory = new
ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();
public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory)
{
    TabulatedFunctions.factory = factory;
} 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);
}
```

Заменили return new ArrayTabulatedFunction(leftX,

rightX, values); на return createTabulatedFunction(leftX,

rightX, values); в main:

```
Function cosFunc = new Cos();
TabulatedFunction tf;

double left = Math.max(0, cosFunc.getLeftDomainBorder()); double
right = Math.min(Math.PI, cosFunc.getRightDomainBorder());

tf = TabulatedFunctions.tabulate(cosFunc, left, right, 11);
System.out.println(tf.getClass());

TabulatedFunctions.setTabulatedFunctionFactory(         new
LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory()); tf =
TabulatedFunctions.tabulate(cosFunc, left, right, 11);
System.out.println(tf.getClass());

TabulatedFunctions.setTabulatedFunctionFactory(         new
ArrayTabulatedFunction.ArrayTabulatedFunctionFactory()); tf =
TabulatedFunctions.tabulate(cosFunc, left, right, 11);
System.out.println(tf.getClass());
```

Задание 3

В классе TabulatedFunctions добавить перегруженные методы createTabulatedFunction(), которые, помимо обычных параметров, принимают ссылку Class на класс табулированной функции. Методы используют рефлексию для поиска конструктора с соответствующими параметрами и

создают объект нужного класса. Исключения рефлексии обрабатываются и пробрасываются как IllegalArgumentException.

Проверка работы методов выполняется в main() через создание объектов ArrayTabulatedFunction и LinkedListTabulatedFunction с разными параметрами, а также через метод tabulate(), который тоже принимает класс функции.

```
   public static <T extends TabulatedFunction> T
createTabulatedFunction(Class<T> clazz, double leftX, double rightX, int
pointsCount) {
      try {
         Constructor<T> constructor = clazz.getConstructor(double.class,
double.class, int.class);            return constructor.newInstance(leftX,
rightX, pointsCount);        } catch (NoSuchMethodException |
InstantiationException |                IllegalAccessException |
InvocationTargetException e) {            throw new
IllegalArgumentException(e);
      }
   }


   public static <T extends TabulatedFunction> T
createTabulatedFunction(Class<T> clazz, double leftX, double rightX, double[]
values) {
      try {
         Constructor<T> constructor = clazz.getConstructor(double.class,
double.class, double[].class);          return constructor.newInstance(leftX,
rightX, values);
      } catch (NoSuchMethodException | InstantiationException |
IllegalAccessException | InvocationTargetException e) {            throw
new IllegalArgumentException(e);
```

```
        }
    }

    public static <T extends TabulatedFunction> T
createTabulatedFunction(Class<T> clazz, FunctionPoint[] points) {
        try {
            Constructor<T> constructor = clazz.getConstructor(FunctionPoint[].class);
return constructor.newInstance((Object) points);
        } catch (NoSuchMethodException | InstantiationException |
IllegalAccessException | InvocationTargetException e) {            throw
new IllegalArgumentException(e);
        }
    }


    public static <T extends TabulatedFunction> T tabulate(Class<T> clazz,
Function function, double leftX, double rightX, int pointsCount) {
        if (leftX < function.getLeftDomainBorder() || rightX >
function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы табулирования выходят
за область определения функции");
        }
        if (pointsCount < 2) {            throw new IllegalArgumentException("Число
точек должно быть >= 2");
        }
```

```
        double step = (rightX - leftX) / (pointsCount - 1);
double[] values = new double[pointsCount];        for
(int i = 0; i < pointsCount; i++) {            double x =
leftX + step * i;            values[i] =
function.getFunctionValue(x);
        }

        return createTabulatedFunction(clazz, leftX, rightX, values);
    }
}
```

В main:

```
Function cosFunc = new Cos();
TabulatedFunction tf;

double left = Math.max(0, cosFunc.getLeftDomainBorder()); double
right = Math.min(Math.PI, cosFunc.getRightDomainBorder());

tf = TabulatedFunctions.tabulate(ArrayTabulatedFunction.class, cosFunc, left,
right, 11);
System.out.println(tf.getClass());

tf = TabulatedFunctions.tabulate(LinkedListTabulatedFunction.class, cosFunc,
left, right, 11);
System.out.println(tf.getClass());

tf = TabulatedFunctions.tabulate(ArrayTabulatedFunction.class, cosFunc, left,
right, 11);
System.out.println(tf.getClass());
```

Команды в консоли:

```
C:\Users\fael7\Desktop\лр7\Lab-7-2025>javac Main.java

C:\Users\fael7\Desktop\лр7\Lab-7-2025>java Main
```

(0.0; 0.0)

(1.0; 1.0)

(2.0; 4.0)

(0.0; 0.0)

(1.0; 1.0)

(2.0; 4.0)

class functions.ArrayTabulatedFunction

class functions.LinkedListTabulatedFunction

class functions.ArrayTabulatedFunction