

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 7

**«Реализация итераторов, фабрик и
рефлексивного создания объектов для
табулированных функций»**

по курсу
Объектно-ориентированное программирование

Выполнила: Яньшина Анастасия Юрьевна
Группа 6203-010302D

Содержание

Титульный лист	1
Содержание	2
Задание 1	3
Задание 2	5
Задание 3	9

Задание 1

Итак, добавим итераторы во все объекты типа `TabulatedFunction`. Для начала, меняем интерфейс `TabulatedFunction` (Рис. 1). Затем добавляем итератор в `ArrayTabulatedFunction` (Рис. 2) и в `LinkedListTabulatedFunction` (Рис. 3).

Сразу же тестируем работу итераторов классов табулированных функций, создавая в `main` объект функции и обращаясь к ней. Как видим, всё корректно работает (Рис. 4).

```
package functions;
import java.io.Serializable;
13 usages 2 implementations
public interface TabulatedFunction extends Function, Serializable, Cloneable, Iterable<FunctionPoint> {
    11 usages 2 implementations
    int getPointsCount();
    4 usages 2 implementations
    FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException;
    no usages 2 implementations
}
```

Рис. 1

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

        @Override
        public boolean hasNext() {return currentIndex < points.length;}

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {throw new java.util.NoSuchElementException("В табулированной функции больше нет точек!");}
            //возвращаем копию точки для защиты инкапсуляции
            return new FunctionPoint(points[currentIndex++]);
        }

        @Override
        public void remove() {throw new UnsupportedOperationException("Невозможно выполнить операцию удаления!");}
    };
}
```

Рис. 2

```

@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        4 usages
        private FunctionNode currentNode = head.getNext();

        @Override
        public boolean hasNext() {return currentNode != head;}

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {throw new java.util.NoSuchElementException("В табулированной функции больше нет точек!");}
            FunctionPoint point = new FunctionPoint(currentNode.getPoint());
            currentNode = currentNode.getNext();
            return point;
        }

        @Override
        public void remove() {throw new UnsupportedOperationException("Невозможно выполнить операцию удаления!");}
    };
}

```

Рис. 3

```

import ...

public class Main {
    public static void main(String[] args) {
        TabulatedFunction f = new ArrayTabulatedFunction( leftX: 0, rightX: 10, pointsCount: 9);
        for (FunctionPoint p : f) {
            System.out.println(p);
        }
    }
}

```

"C:\Program Files\Java\jdk-21.0.6\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=63995:C:\Progr
 (0.0; 0.0)
 (1.25; 0.0)
 (2.5; 0.0)
 (3.75; 0.0)
 (5.0; 0.0)
 (6.25; 0.0)
 (7.5; 0.0)
 (8.75; 0.0)
 (10.0; 0.0)
 Process finished with exit code 0

Рис. 4

Задание 2

Следующее задание – реализация фабричного метода. Опишем в пакете `functions` базовый интерфейс фабрик табулированных функций `TabulatedFunctionFactory` (Рис. 5). Теперь добавим в классы `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` вложенные и публичные классы фабрик `ArrayTabulatedFunctionFactory` и `LinkedListTabulatedFunctionFactory` (Рис. 6 и рис. 7).

Затем меняем класс `TabulatedFunctions()`. Добавляем поле `TabulatedFunctionFactory factory` и новые методы создания через фабрику (Рис. 8). Помимо этого, в уже существующих методах вносим изменения, чтобы возвращаемый объект был создан через фабрику. Изменения произошли в `inputTabulatedFunction()`, `readTabulatedFunction()` и `tabulate()` (Рис. 9, 10 и 11).

Меняем `main` и проверяем работу написанного. Вроде всё работает так, как и должно (Рис. 12).

```
package functions;

no usages
public interface TabulatedFunctionFactory {
    no usages
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException;
    no usages
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException;
    no usages
    TabulatedFunction createTabulatedFunction(FunctionPoint[] points) throws IllegalArgumentException;
}
```

Рис. 5

```

no usages
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {
    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {
        return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException {
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) throws IllegalArgumentException {
        return new ArrayTabulatedFunction(points);
    }
}

```

Рис. 6

```

no usages
public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory {
    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {
        return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException {
        return new LinkedListTabulatedFunction(leftX, rightX, values);
    }

    no usages
    @Override
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) throws IllegalArgumentException {
        return new LinkedListTabulatedFunction(points);
    }
}

```

Рис. 7


```

public class TabulatedFunctions {
    no usages
    private TabulatedFunctions() {throw new AssertionError( detailMessage: "Нельзя создавать экземпляры утилитного класса");}
    4 usages
    private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();
    no usages
    public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) {TabulatedFunctions.factory = factory;}

    //новые методы создания через фабрику
    no usages
    public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
        return factory.createTabulatedFunction(leftX, rightX, pointsCount);
    }

    no usages
    public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return factory.createTabulatedFunction(leftX, rightX, values);
    }

    no usages
    public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return factory.createTabulatedFunction(points);
    }
}

```

Рис. 8

```

3 usages
public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
    //проверяем возможные ошибки в параметрах и корректность границ табулирования
    if (leftX >= rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    if (pointsCount < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {throw new IllegalArgumentException("Границы табулирования выходят за пределы области определения функции");}

    //создаем табулированную функцию, используя TabulatedFunction
    TabulatedFunction result = createTabulatedFunction(leftX, rightX, pointsCount);

    //заполняем значениями исходной функции
    for (int i = 0; i < pointsCount; i++) {

```

Рис. 9

```

public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
    DataInputStream dis = new DataInputStream(in);
    //читаем количество точек
    int pointsCount = dis.readInt();

    //читаем координаты точек
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        points[i] = new FunctionPoint(dis.readDouble(), dis.readDouble());
    }

    //создаем табулированную функцию (реализация через ArrayTabulatedFunction как в методе табулирования)
    return factory.createTabulatedFunction(points);
}

```

Рис. 10

```

st.whitespaceChars( low: ' ', hi: ' ');
st.whitespaceChars( low: ',', hi: ', ');
st.whitespaceChars( low: '\t', hi: '\t');
st.whitespaceChars( low: '\n', hi: '\n');
st.whitespaceChars( low: '\r', hi: '\r');

//читаем количество точек
if (st.nextToken() != StreamTokenizer.TT_WORD) {throw new RuntimeExcept
int pointsCount = Integer.parseInt(st.sval);

//читаем координаты точек
FunctionPoint[] points = new FunctionPoint[pointsCount];
for (int i = 0; i < pointsCount; i++) {
    //читаем X
    if (st.nextToken() != StreamTokenizer.TT_WORD) {throw new RuntimeE
    double x = Double.parseDouble(st.sval);

    //читаем Y
    if (st.nextToken() != StreamTokenizer.TT_WORD) {throw new RuntimeE
    double y = Double.parseDouble(st.sval);

    points[i] = new FunctionPoint(x, y);
}

return factory.createTabulatedFunction(points);
}

```

Рис. 11

The screenshot shows the IntelliJ IDEA IDE. On the left, the 'Project' tool window displays a package structure with 'functions' at the top, followed by 'basic', 'meta', and 'ArrayTabulatedFunction'. Under 'functions', there are several classes: 'Function', 'FunctionPoint', 'FunctionPointIndexOutOfBoundsException', 'Functions', 'InappropriateFunctionPointException', 'LinkedListTabulatedFunction', 'TabulatedFunction', 'TabulatedFunctionFactory', and 'TabulatedFunctions'. The 'Main' class is highlighted. The 'Run' tool window at the bottom shows the execution of the 'Main' class, displaying the command: `"C:\Program Files\Java\jdk-21.0.6\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=64272:C:\Program F` and the output: `class functions.ArrayTabulatedFunction`, `class functions.LinkedListTabulatedFunction`, `class functions.ArrayTabulatedFunction`, and `Process finished with exit code 0`.

```

9 public class Main {
10     public static void main(String[] args) {
11         Function f = new Cos();
12         TabulatedFunction tf;
13         tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
14         System.out.println(tf.getClass());
15         TabulatedFunctions.setTabulatedFunctionFactory(new
16             LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
17         tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
18         System.out.println(tf.getClass());
19         TabulatedFunctions.setTabulatedFunctionFactory(new
20             ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
21         tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
22         System.out.println(tf.getClass());
23     }
24 }

```

Рис. 12

Задание 3

Последним заданием лабораторной реализуем рефлексивное создание объектов. Добавляем нужные методы для всех трех типов конструкторов в класс `TabulatedFunctions` (Рис. 13 и рис. 14). Реализуем эти три перегруженных метода `createTabulatedFunction()`, принимающих параметр `Class<?>`. Каждый метод проверяет, что класс реализует `TabulatedFunction`, выполняет поиск конструктора с помощью рефлексии, а также обрабатывает исключения с преобразованием в `IllegalArgumentException`.

Также перегружаем несколько существующих методов, чтобы всё работало корректно. Изменения вновь коснулись только `inputTabulatedFunction()`, `readTabulatedFunction()` и `tabulate()` (Рис. 15, 16, 17 и 18).

В завершение проверяем работу написанных методов рефлексивного создания объектов, а также методов класса `TabulatedFunctions`, использующих создание объектов. Используем буквально пример из задания лабораторной (Рис. 19). После запуска убеждаемся, что всё работает корректно (Рис. 20).

```

//методы рефлексивного создания объектов
2 usages
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, int pointsCount) {
    try {
        //проверяем, что класс реализует TabulatedFunction
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }

        //ищем конструктор с параметрами (double, double, int)
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, int.class);
        //создаём объект
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);
    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}

1 usage
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, double[] values) {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }

        //ищем нужный конструктор и создаём объект
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, double[].class);
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, values);
    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}

```

Рис. 13

```

1 usage
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, FunctionPoint[] points) {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }

        //ищем нужный конструктор и создаём объект
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);
    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}

```

Рис. 14

```

//перегруженный метод табулирования с рефлексией
1 usage
public static TabulatedFunction tabulate(Class<?> functionClass, Function function, double leftX, double rightX, int pointsCount) {
    //проверяем возможные ошибки в параметрах и корректность границ табулирования
    if (leftX > rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой!");
    if (pointsCount < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {throw new IllegalArgumentException("Границы табулирования выходят за

    //создаем табулированную функцию, используя TabulatedFunction
    //проверки в данном случае выполняются при попытке вызова конструктора, ошибки обрабатываются там же
    TabulatedFunction result = createTabulatedFunction(functionClass, leftX, rightX, pointsCount);

    //заполняем значениями исходной функции
    for (int i = 0; i < pointsCount; i++) {
        double x = result.getPointX(i);
        double y = function.getFunctionValue(x);
        result.setPointY(i, y);
    }
    return result;
}

```

Рис. 15

```

no usages
public static TabulatedFunction inputTabulatedFunction(Class<?> functionClass, InputStream in) throws IOException {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }
        DataInputStream dis = new DataInputStream(in);
        //читаем количество точек
        int pointsCount = dis.readInt();

        //читаем координаты точек
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(dis.readDouble(), dis.readDouble());
        }

        //создаем табулированную функцию через рефлексию
        java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
        return (TabulatedFunction) constructor.newInstance((Object) points);
    } catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);}
}

```

Рис. 16

```

public static TabulatedFunction readTabulatedFunction(Class<?> functionClass, Reader in) throws IOException {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction!");
        }
        StreamTokenizer st = new StreamTokenizer(in);

        //настройка токенизатора для чтения чисел
        st.resetSyntax();

        st.wordChars(low: '0', hi: '9');
        st.wordChars(low: '.', hi: '.');
        st.wordChars(low: '-', hi: '-');
        st.wordChars(low: 'E', hi: 'E');
        st.wordChars(low: 'e', hi: 'e');

        st.whitespaceChars(low: ' ', hi: ' ');
        st.whitespaceChars(low: '(', hi: '(');
        st.whitespaceChars(low: ')', hi: ')');
        st.whitespaceChars(low: ',', hi: ',');
        st.whitespaceChars(low: '\t', hi: '\t');
        st.whitespaceChars(low: '\n', hi: '\n');
        st.whitespaceChars(low: '\r', hi: '\r');

        //читаем количество точек
        if (st.nextToken() != StreamTokenizer.TT_WORD) {
            throw new RuntimeException("Ожидалось количество точек!");
        }
        int pointsCount = Integer.parseInt(st.sval);
    }
}

```

Рис. 17

```

//читаем количество точек
if (st.nextToken() != StreamTokenizer.TT_WORD) {
    throw new RuntimeException("Ожидалось количество точек!");
}
int pointsCount = Integer.parseInt(st.sval);

//читаем координаты точек
FunctionPoint[] points = new FunctionPoint[pointsCount];
for (int i = 0; i < pointsCount; i++) {
    //читаем X
    if (st.nextToken() != StreamTokenizer.TT_WORD) {
        throw new RuntimeException("Ожидалась координата X!");
    }
    double x = Double.parseDouble(st.sval);

    //читаем Y
    if (st.nextToken() != StreamTokenizer.TT_WORD) {
        throw new RuntimeException("Ожидалась координата Y!");
    }
    double y = Double.parseDouble(st.sval);

    points[i] = new FunctionPoint(x, y);
}

//создаем объект через рефлексию
java.lang.reflect.Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class);
return (TabulatedFunction) constructor.newInstance((Object) points);
} catch (Exception e) {throw new IllegalArgumentException("Невозможно создать экземпляр табулированной функции: ", e);
}

```

Рис. 18


```

System.out.println("-----Тест рефлексии-----");
TabulatedFunction function;

function = TabulatedFunctions.createTabulatedFunction(
    ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3);
System.out.println(function.getClass());
System.out.println(function);

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

function = TabulatedFunctions.createTabulatedFunction(
    LinkedListTabulatedFunction.class,
    new FunctionPoint[] {
        new FunctionPoint(x: 0, y: 0),
        new FunctionPoint(x: 10, y: 10)
    }
);
System.out.println(function.getClass());
System.out.println(function);

function = TabulatedFunctions.tabulate(
    LinkedListTabulatedFunction.class, new Sin(), leftX: 0, Math.PI, pointsCount: 11);
System.out.println(function.getClass());
System.out.println(function);
}

```

Рис. 19

```

Main
"C:\Program Files\Java\jdk-21.0.6\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\lib\idea_rt.jar=61426:C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\bin" -Didea.config.path=C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\config -Didea.copyright.path=C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\copyright -Didea.home.path=C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\bin -Didea.platform.prefix=java -Didea.vendor.id=jetbrains -Didea.version=2023.1.3 -jar C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.3\bin\idea.jar

-----Тест итераторов-----
(0.0; 0.0)
(1.25; 0.0)
(2.5; 0.0)
(3.75; 0.0)
(5.0; 0.0)
(6.25; 0.0)
(7.5; 0.0)
(8.75; 0.0)
(10.0; 0.0)

-----Тест фабричного метода-----
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction

-----Тест рефлексии-----
class functions.ArrayTabulatedFunction
{(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}
class functions.ArrayTabulatedFunction
{(0.0; 0.0), (10.0; 10.0)}
class functions.LinkedListTabulatedFunction
{(0.0; 0.0), (10.0; 10.0)}
class 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), (1.884955592153875; 0.9510565162951535), (2.199114857512854; 0.8090169943749475), (2.513274128871833; 0.5877852522924731), (2.827433399230812; 0.3090169943749474), (3.141592653589793; 0.0), (3.455751918948772; -0.3090169943749474), (3.770009154907751; -0.5877852522924731), (4.08416842026673; -0.8090169943749475), (4.398327685625709; -0.9510565162951535), (4.712486950984688; -1.0), (5.026646216343667; -0.9510565162951535), (5.340805481702646; -0.8090169943749475), (5.654964747061625; -0.5877852522924731), (5.969124012420604; -0.3090169943749474), (6.283185307179586; 0.0), (6.597344572538565; 0.3090169943749474), (6.911503837897544; 0.5877852522924731), (7.225663103256523; 0.8090169943749475), (7.539822368615502; 0.9510565162951535), (7.853981633974481; 1.0), (8.16814090433346; 0.9510565162951535), (8.482300169692439; 0.8090169943749475), (8.796459435051418; 0.5877852522924731), (9.110618700410397; 0.3090169943749474), (9.424777965769376; 0.0), (9.738937231128355; -0.3090169943749474), (10.053096496487334; -0.5877852522924731), (10.367255761846313; -0.8090169943749475), (10.681415027205292; -0.9510565162951535), (10.995574292564271; -1.0), (11.30973355792325; -0.9510565162951535), (11.623892823282229; -0.8090169943749475), (11.938052088641208; -0.5877852522924731), (12.252211354000187; -0.3090169943749474), (12.566370619359166; 0.0), (12.880529884718145; 0.3090169943749474), (13.194689150077124; 0.5877852522924731), (13.508848415436103; 0.8090169943749475), (13.823007680795082; 0.9510565162951535), (14.137166946154061; 1.0), (14.45132621151304; 0.9510565162951535), (14.765485476872019; 0.8090169943749475), (15.079644742230998; 0.5877852522924731), (15.393804007589977; 0.3090169943749474), (15.707963272948956; 0.0), (16.022122538307935; -0.3090169943749474), (16.336281803666914; -0.5877852522924731), (16.650441069025893; -0.8090169943749475), (16.964600334384872; -0.9510565162951535), (17.278759604743851; -1.0), (17.59291887010283; -0.9510565162951535), (17.907078135461809; -0.8090169943749475), (18.221237400820788; -0.5877852522924731), (18.535396666179767; -0.3090169943749474), (18.849555931538746; 0.0), (19.163715196897725; 0.3090169943749474), (19.477874462256704; 0.5877852522924731), (19.792033727615683; 0.8090169943749475), (20.106192992974662; 0.9510565162951535), (20.420352258333641; 1.0), (20.73451152369262; 0.9510565162951535), (21.048670789051599; 0.8090169943749475), (21.362830054410578; 0.5877852522924731), (21.676989319769557; 0.3090169943749474), (21.991148585128536; 0.0), (22.305307850487515; -0.3090169943749474), (22.619467115846494; -0.5877852522924731), (22.933626381205473; -0.8090169943749475), (23.247785646564452; -0.9510565162951535), (23.561944911923431; -1.0), (23.87610417728241; -0.9510565162951535), (24.190263442641389; -0.8090169943749475), (24.504422708000368; -0.5877852522924731), (24.818581973359347; -0.3090169943749474), (25.132741238718326; 0.0), (25.446900504077305; 0.3090169943749474), (25.761059769436284; 0.5877852522924731), (26.075219034795263; 0.8090169943749475), (26.389378300154242; 0.9510565162951535), (26.703537565513221; 1.0), (27.0176968308722; 0.9510565162951535), (27.331856096231179; 0.8090169943749475), (27.646015361590158; 0.5877852522924731), (27.960174626949137; 0.3090169943749474), (28.274333892308116; 0.0), (28.588493157667095; -0.3090169943749474), (28.902652423026074; -0.5877852522924731), (29.216811688385053; -0.8090169943749475), (29.530970953744032; -0.9510565162951535), (29.845130219103011; -1.0), (30.15928948446199; -0.9510565162951535), (30.473448749820969; -0.8090169943749475), (30.787608015179948; -0.5877852522924731), (31.101767280538927; -0.3090169943749474), (31.415926545897906; 0.0), (31.730085811256885; 0.3090169943749474), (32.044245076615864; 0.5877852522924731), (32.358404341974843; 0.8090169943749475), (32.672563607333822; 0.9510565162951535), (32.986722872692801; 1.0), (33.30088213805178; 0.9510565162951535), (33.615041403410759; 0.8090169943749475), (33.929200668769738; 0.5877852522924731), (34.243359934128717; 0.3090169943749474), (34.557519204487696; 0.0), (34.871678469846675; -0.3090169943749474), (35.185837735205654; -0.5877852522924731), (35.499997000564633; -0.8090169943749475), (35.814156265923612; -0.9510565162951535), (36.128315531282591; -1.0), (36.44247479664157; -0.9510565162951535), (36.756634062000549; -0.8090169943749475), (37.070793327359528; -0.5877852522924731), (37.384952592718507; -0.3090169943749474), (37.699111858077486; 0.0), (38.013271123436465; 0.3090169943749474), (38.327430388795444; 0.5877852522924731), (38.641589654154423; 0.8090169943749475), (38.955748919513402; 0.9510565162951535), (39.269908184872381; 1.0), (39.58406745023136; 0.9510565162951535), (39.898226715590339; 0.8090169943749475), (40.212385980949318; 0.5877852522924731), (40.526545246308297; 0.3090169943749474), (40.840704511667276; 0.0), (41.154863777026255; -0.3090169943749474), (41.469023042385234; -0.5877852522924731), (41.783182307744213; -0.8090169943749475), (42.097341573103192; -0.9510565162951535), (42.411500838462171; -1.0), (42.72566010382115; -0.9510565162951535), (43.03981936917913; -0.8090169943749475), (43.354038634538109; -0.5877852522924731), (43.668257904897088; -0.3090169943749474), (43.982477170256067; 0.0), (44.296696435615046; 0.3090169943749474), (44.610915700974025; 0.5877852522924731), (44.925134966333004; 0.8090169943749475), (45.239354231691983; 0.9510565162951535), (45.553573497050962; 1.0), (45.867792762409941; 0.9510565162951535), (46.18201202776892; 0.8090169943749475), (46.496231293127899; 0.5877852522924731), (46.810450558486878; 0.3090169943749474), (47.124669823845857; 0.0), (47.438889089204836; -0.3090169943749474), (47.753108354563815; -0.5877852522924731), (48.067327619922794; -0.8090169943749475), (48.381546885281773; -0.9510565162951535), (48.695766150640752; -1.0), (49.009985416099731; -0.9510565162951535), (49.32420468145871; -0.8090169943749475), (49.638423946817689; -0.5877852522924731), (49.952643212176668; -0.3090169943749474), (50.266862477535647; 0.0), (50.581081742894626; 0.3090169943749474), (50.895301008253605; 0.5877852522924731), (51.209520273612584; 0.8090169943749475), (51.523739538971563; 0.9510565162951535), (51.837958804330542; 1.0), (52.152178069689521; 0.9510565162951535), (52.4663973350485; 0.8090169943749475), (52.780616600407479; 0.5877852522924731), (53.094835865766458; 0.3090169943749474), (53.409055131125437; 0.0), (53.723274396484416; -0.3090169943749474), (54.037493661843395; -0.5877852522924731), (54.351712927202374; -0.8090169943749475), (54.665932192561353; -0.9510565162951535), (54.980151457920332; -1.0), (55.294370723279311; -0.9510565162951535), (55.60858998863829; -0.8090169943749475), (55.922809253997269; -0.5877852522924731), (56.237028519356248; -0.3090169943749474), (56.551247784715227; 0.0), (56.865467050074206; 0.3090169943749474), (57.179686315433185; 0.5877852522924731), (57.493905580792164; 0.8090169943749475), (57.808124846151143; 0.9510565162951535), (58.122344111510122; 1.0), (58.436563376869101; 0.9510565162951535), (58.75078264222808; 0.8090169943749475), (59.064997907587059; 0.5877852522924731), (59.379217172946038; 0.3090169943749474), (59.693436438305017; 0.0), (59.997655703663996; -0.3090169943749474), (60.311874969022975; -0.5877852522924731), (60.626094234381954; -0.8090169943749475), (60.940313504740933; -0.9510565162951535), (61.254532770099912; -1.0), (61.568752035458891; -0.9510565162951535), (61.88297130081787; -0.8090169943749475), (62.197190566176849; -0.5877852522924731), (62.511409831535828; -0.3090169943749474), (62.825629096894807; 0.0), (63.139848362253786; 0.3090169943749474), (63.454067627612765; 0.5877852522924731), (63.768286892971744; 0.8090169943749475), (64.082506158330723; 0.9510565162951535), (64.396725423689702; 1.0), (64.710944689048681; 0.9510565162951535), (65.02516395440766; 0.8090169943749475), (65.339383219766639; 0.5877852522924731), (65.653602485125618; 0.3090169943749474), (65.967821750484597; 0.0), (66.282041015843576; -0.3090169943749474), (66.596260281202555; -0.5877852522924731), (66.910479546561534; -0.8090169943749475), (67.224698811920513; -0.9510565162951535), (67.538918077279492; -1.0), (67.853137342638471; -0.9510565162951535), (68.16735660799745; -0.8090169943749475), (68.481575873356429; -0.5877852522924731), (68.795795138715408; -0.3090169943749474), (69.109974404074387; 0.0), (69.424193669433366; 0.3090169943749474), (69.738412934792345; 0.5877852522924731), (70.052632200151324; 0.8090169943749475), (70.366851465510303; 0.9510565162951535), (70.681070730869282; 1.0), (70.995289996228261; 0.9510565162951535), (71.30950926158724; 0.8090169943749475), (71.623728526946219; 0.5877852522924731), (71.937947792305198; 0.3090169943749474), (72.252167057664177; 0.0), (72.566386323023156; -0.3090169943749474), (72.880605588382135; -0.5877852522924731), (73.194824853741114; -0.8090169943749475), (73.509044119100093; -0.9510565162951535), (73.823263384459072; -1.0), (74.137482649818051; -0.9510565162951535), (74.45170191517703; -0.8090169943749475), (74.765921180536009; -0.5877852522924731), (75.080140445894988; -0.3090169943749474), (75.394359711253967; 0.0), (75.708578976612946; 0.3090169943749474), (76.022798241971925; 0.5877852522924731), (76.337017507330904; 0.8090169943749475), (76.651236772689883; 0.9510565162951535), (76.965456038048862; 1.0), (77.279675303407841; 0.9510565162951535), (77.59389456876682; 0.8090169943749475), (77.908113834125799; 0.5877852522924731), (78.222333104484778; 0.3090169943749474), (78.536552369843757; 0.0), (78.850771635202736; -0.3090169943749474), (79.164990900561715; -0.5877852522924731), (79.479210165920694; -0.8090169943749475), (79.793429431279673; -0.9510565162951535), (80.107648696638652; -1.0), (80.421867961997631; -0.9510565162951535), (80.73608722735661; -0.8090169943749475), (81.050306492715589; -0.5877852522924731), (81.364525758074568; -0.3090169943749474), (81.678745023433547; 0.0), (81.992964288792526; 0.3090169943749474), (82.307183554151505; 0.5877852522924731), (82.621402819510484; 0.8090169943749475), (82.935622084869463; 0.9510565162951535), (83.249841350228442; 1.0), (83.564060615587421; 0.9510565162951535), (83.8782798809464; 0.8090169943749475), (84.192499146305379; 0.5877852522924731), (84.506718411664358; 0.3090169943749474), (84.820937677023337; 0.0), (85.135156942382316; -0.3090169943749474), (85.449376207741295; -0.5877852522924731), (85.763595473100274; -0.8090169943749475), (86.077814738459253; -0.9510565162951535), (86.392034003818232; -1.0), (86.706253269177211; -0.9510565162951535), (87.02047253453619; -0.8090169943749475), (87.334691804895169; -0.5877852522924731), (87.648911070254148; -0.3090169943749474), (87.963130335613127; 0.0), (88.277349600972106; 0.3090169943749474), (88.591568866331085; 0.5877852522924731), (88.905788131690064; 0.8090169943749475), (89.219967397049043; 0.9510565162951535), (89.534186662408022; 1.0), (89.848405927767001; 0.9510565162951535), (90.16262519312598; 0.8090169943749475), (90.476844458484959; 0.5877852522924731), (90.791063723843938; 0.3090169943749474), (91.105282989202917; 0.0), (91.419502254561896; -0.3090169943749474), (91.733721519920875; -0.5877852522924731), (92.047940785279854; -0.8090169943749475), (92.362160050638833; -0.9510565162951535), (92.676379315997812; -1.0), (92.990598581356791; -0.9510565162951535), (93.30481784671577; -0.8090169943749475), (93.619037112074749; -0.5877852522924731), (93.933256377433728; -0.3090169943749474), (94.247475642792707; 0.0), (94.561694908151686; 0.3090169943749474), (94.875914173510665; 0.5877852522924731), (95.190133438869644; 0.8090169943749475), (95.504352704228623; 0.9510565162951535), (95.818571969587602; 1.0), (96.132791234946581; 0.9510565162951535), (96.44701050030556; 0.8090169943749475), (96.761229765664539; 0.5877852522924731), (97.075449031023518; 0.3090169943749474), (97.389668296382497; 0.0), (97.703887561741476; -0.3090169943749474), (98.018106827100455; -0.5877852522924731), (98.332326092459434; -0.8090169943749475
```