

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 №2

Дисциплина: «ООП»

Тема «Базовые конструкции»

Выполнил: Сучков Борис Антонович

Группа: 6201-120303D

Самара 2025

Задание на лабораторную работу

Разработать набор классов для работы с функциями одной переменной, заданными в табличной форме.

В данной работе можно не делать проверки корректности параметров методов, если только задание не требует этого в явной форме. Необходимые для этого модификации будут выполняться в следующей работе.

В ходе выполнения работы запрещено использовать классы из пакета `java.util`.

Задание 1

Создать пакет `functions`, в котором далее будут создаваться классы программы.

Задание 2

В пакете `functions` создать класс `FunctionPoint`, объект которого должен описывать одну точку табулированной функции.

Состояние объектов должно содержать два аспекта: координату точки по оси абсцисс и координату точки по оси ординат. При написании класса следует учесть особенности инкапсуляции.

В классе должны быть описаны следующие конструкторы:

- `FunctionPoint(double x, double y)` – создаёт объект точки с заданными координатами;
- `FunctionPoint(FunctionPoint point)` – создаёт объект точки с теми же координатами, что у указанной точки;
- `FunctionPoint()` – создаёт точку с координатами (0; 0).

Задание 3

В пакете `functions` создать класс `TabulatedFunction`, объект которого должен описывать табулированную функцию.

Для хранения данных о точках должен использоваться массив типа `FunctionPoint`. При этом разумно организовать работу с массивом так, чтобы точки в нём были всегда упорядочены по значению координаты x .

В классе должны быть описаны следующие конструкторы:

`TabulatedFunction(double leftX, double rightX, int pointsCount)` – создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования (значения функции в точках при этом следует считать равными 0);

`TabulatedFunction(double leftX, double rightX, double[] values)` – аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива.

В обоих случаях точки должны создаваться через равные интервалы по x .

Задание 4

В классе `TabulatedFunction` описать методы, необходимые для работы с функцией.

Метод `double getLeftDomainBorder()` должен возвращать значение левой границы области определения табулированной функции. Очевидно, что оно совпадает с абсциссой самой левой точки в описывающей функцию таблице.

Аналогично, метод `double getRightDomainBorder()` должен возвращать значение правой границы области определения табулированной функции.

Метод `double getFunctionValue(double x)` должен возвращать значение функции в точке x , если эта точка лежит в области определения функции. В противном случае метод должен возвращать значение неопределённости (оно хранится, например, в поле `NaN` класса `Double`). При расчёте значения функции следует использовать линейную интерполяцию, т.е. считать, что на интервале между заданными в таблице точками функция является прямой линией. Для написания кода метода рекомендуется воспользоваться уравнением прямой, проходящей через две заданные различающиеся точки.

Задание 5

В классе `TabulatedFunction` описать методы, необходимые для работы с точками табулированной функции. Считать, что нумерация точек начинается с нуля.

Метод `int getPointsCount()` должен возвращать количество точек.

Метод `FunctionPoint getPoint(int index)` должен возвращать копию точки, соответствующей переданному индексу. Возвращение ссылки на саму точку противоречит принципу инкапсуляции.

Метод `void setPoint(int index, FunctionPoint point)` должен заменять указанную точку табулированной функции на переданную. Для корректной инкапсуляции замените на копию переданной точки. В случае если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции, то замену точки проводить не следует. Например, для функции, определяемой точками $\{(0; 0), (1; 1), (2; 4)\}$, точку с индексом 1 нельзя заменить точкой $(-1; 5)$.

Метод `double getPointX(int index)` должен возвращать значение абсциссы точки с указанным номером.

Метод `void setPointX(int index, double x)` должен изменять значение абсциссы точки с указанным номером. Аналогично методу `setPoint()`, данный метод не должен изменять точку, если новое значение попадает в другой интервал табулирования.

Метод `double getPointY(int index)` должен возвращать значение ординаты точки с указанным номером.

Метод `void setPointY(int index, double y)` должен изменять значение ординаты точки с указанным номером.

Задание 6

В классе `TabulatedFunction` описать методы, изменяющие количество точек табулированной функции.

Метод `void deletePoint(int index)` должен удалять заданную точку табулированной функции.

Метод `void addPoint(FunctionPoint point)` должен добавлять новую точку табулированной функции. При написании метода обеспечьте корректную инкапсуляцию.

При написании методов следует учитывать, что точки в массиве должны быть упорядочены по значению координаты x .

Для копирования участков массивов рекомендуется воспользоваться методом `arraycopy()` класса `System`.

Также следует понимать, что создание нового массива каждый раз при выполнении операций удаления и вставки точки является расточительством по отношению к памяти и скорости работы программы. Поэтому длина массива в общем случае не должна совпадать с количеством точек в табулированной функции, а замена массива на массив большей длины должна производиться только в некоторых случаях.

Задание 7

Проверить работу написанных классов.

В пакете по умолчанию (вне пакета `functions`) нужно создать класс `Main`, содержащий точку входа программы.

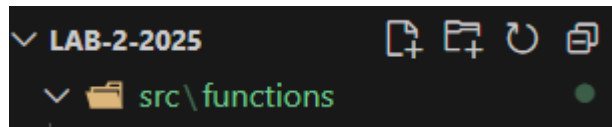
В методе `main()` создайте экземпляр класса `TabulatedFunction` и задайте для него табулированные значения какой-нибудь известной вам функции.

Выведите в консоль значения функции на ряде точек. Рекомендуется использовать такой шаг (или такие точки), чтобы среди них оказались точки вне области определения функции, а также чтобы несколько точек попали в один интервал табулированной функции.

Проверьте, как изменяется результат работы программы после изменения точек, добавления и удаления точек.

Задание 1

Я создал пакет `functions`, в котором далее будут создаваться классы программы.



Задание 2

В пакете `functions` я создал класс `FunctionPoint`, объект которого описывает одну точку табулированной функции.

Состояние объектов содержит два аспекта: координату точки по оси абсцисс и координату точки по оси ординат. При написании класса учтены особенности инкапсуляции.

В классе описаны следующие конструкторы:

- `FunctionPoint(double x, double y)` – создаёт объект точки с заданными координатами;
- `FunctionPoint(FunctionPoint point)` – создаёт объект точки с теми же координатами, что у указанной точки;
- `FunctionPoint()` – создаёт точку с координатами (0; 0).

```
FunctionPoint.java U X
src > functions > FunctionPoint.java > FunctionPoint > getY()
1 package functions; // Указываем, что класс находится в пакете functions
2
3 // Описывает одну точку табулированной функции (пара x и y).
4 public class FunctionPoint {
5     private double x;
6     private double y;
7
8     public FunctionPoint(double x, double y) { // создаёт объект точки с заданными координатами
9         this.x = x;
10        this.y = y;
11    }
12    public FunctionPoint(FunctionPoint point) { // создаёт объект точки с теми же координатами, что у указанной точки
13        this.x = point.x;
14        this.y = point.y;
15    }
16    public FunctionPoint() { // создаёт точку с координатами (0; 0)
17        this.x = 0;
18        this.y = 0;
19    }
20    public double getX() {
21        return x;
22    }
23    public double getY() {
24        return y;
25    }
26    public void setX(double x) {
27        this.x = x;
28    }
29    public void setY(double y) {
30        this.y = y;
31    }
32 }
33
```

Задание 3

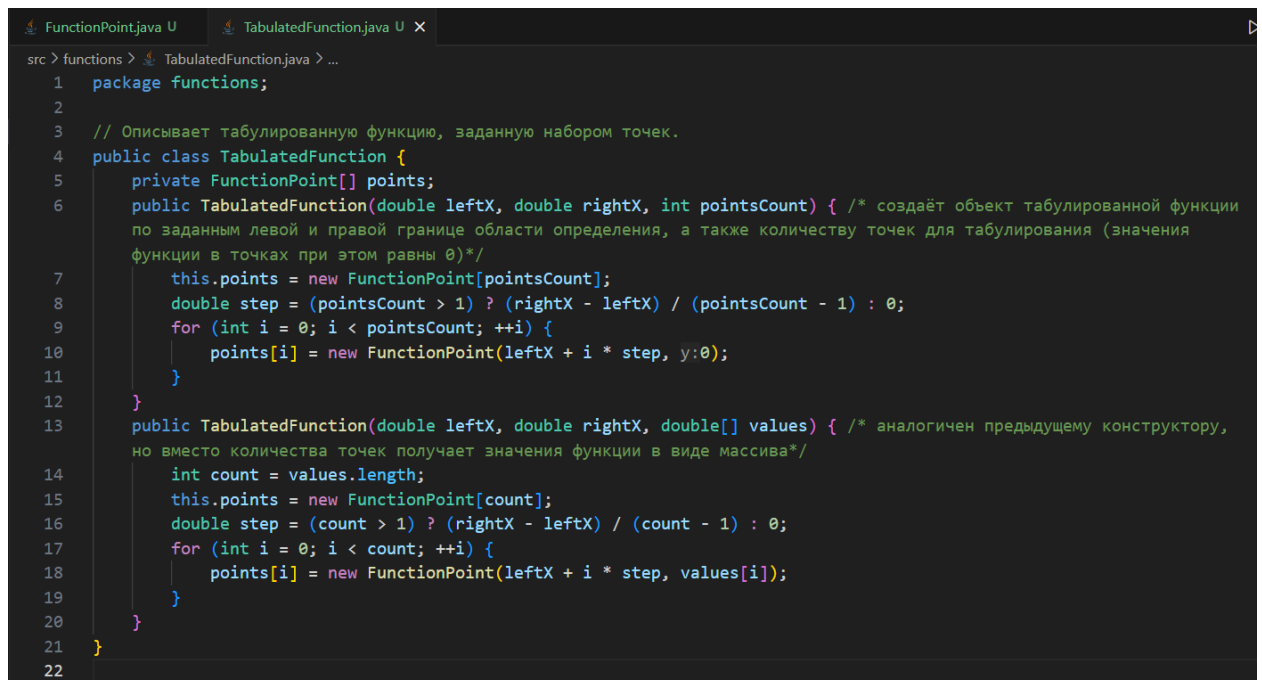
Для хранения данных о точках используется массив типа `FunctionPoint`. При этом работа с массивом организована так, что точки в нём всегда упорядочены по значению координаты x .

В классе описаны следующие конструкторы:

`TabulatedFunction(double leftX, double rightX, int pointsCount)` – создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования (значения функции в точках при этом равны 0);

`TabulatedFunction(double leftX, double rightX, double[] values)` – аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива.

В обоих случаях точки создаются через равные интервалы по x .



```
1 package functions;
2
3 // Описывает табулированную функцию, заданную набором точек.
4 public class TabulatedFunction {
5     private FunctionPoint[] points;
6     public TabulatedFunction(double leftX, double rightX, int pointsCount) { /* создаёт объект табулированной функции
7         по заданным левой и правой границе области определения, а также количеству точек для табулирования (значения
8         функции в точках при этом равны 0)*/
9         this.points = new FunctionPoint[pointsCount];
10        double step = (pointsCount > 1) ? (rightX - leftX) / (pointsCount - 1) : 0;
11        for (int i = 0; i < pointsCount; ++i) {
12            points[i] = new FunctionPoint(leftX + i * step, y:0);
13        }
14    }
15    public TabulatedFunction(double leftX, double rightX, double[] values) { /* аналогичен предыдущему конструктору,
16        но вместо количества точек получает значения функции в виде массива*/
17        int count = values.length;
18        this.points = new FunctionPoint[count];
19        double step = (count > 1) ? (rightX - leftX) / (count - 1) : 0;
20        for (int i = 0; i < count; ++i) {
21            points[i] = new FunctionPoint(leftX + i * step, values[i]);
22        }
23    }
24 }
```

Задание 4

В классе `TabulatedFunction` я описал методы, необходимые для работы с функцией.

Метод `double getLeftDomainBorder()` возвращает значение левой границы области определения табулированной функции. Оно совпадает с абсциссой самой левой точки в описывающей функцию таблице.

Аналогично, метод `double getRightDomainBorder()` возвращает значение правой границы области определения табулированной функции.

Метод `double getFunctionValue(double x)` возвращает значение функции в точке `x`, если эта точка лежит в области определения функции. В противном случае метод возвращает значение неопределённости (оно хранится в поле `NaN` класса `Double`). При расчёте значения функции используется линейная интерполяция, т.е. считается, что на интервале между заданными в таблице точками функция является прямой линией.

```
20     }
21     public double getLeftDomainBorder() {
22         return points[0].getX();
23     }
24     public double getRightDomainBorder() {
25         return points[points.length - 1].getX();
26     }
27     public double getFunctionValue(double x) {
28         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
29             return Double.NaN;
30         }
31         for (int i = 0; i < points.length - 1; ++i) {
32             if (points[i].getX() <= x && x <= points[i + 1].getX()) {
33                 double x1 = points[i].getX();
34                 double y1 = points[i].getY();
35                 double x2 = points[i + 1].getX();
36                 double y2 = points[i + 1].getY();
37                 return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
38             }
39         }
40         return Double.NaN;
41     }
42 }
```

```
FunctionPoint.java U  TabulatedFunction.java U X
src > Functions > / TabulatedFunction.java > ...
1 package Functions;
2
3 // Описывает табулированную функцию, заданную набором точек.
4 public class TabulatedFunction {
5     private FunctionPoint[] points;
6     public TabulatedFunction(double leftX, double rightX, int pointsCount) { /* создаёт объект табулированной функции по заданным левой и правой границе области определения, а также
7         количеству точек для табулирования (значения функции в точках при этом равны 0) */
8         this.points = new FunctionPoint[pointsCount];
9         double step = (pointsCount > 1) ? (rightX - leftX) / (pointsCount - 1) : 0;
10         for (int i = 0; i < pointsCount; ++i) {
11             points[i] = new FunctionPoint(leftX + i * step, 0);
12         }
13     }
14     public TabulatedFunction(double leftX, double rightX, double[] values) { /* аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива */
15         int count = values.length;
16         this.points = new FunctionPoint[count];
17         double step = (count > 1) ? (rightX - leftX) / (count - 1) : 0;
18         for (int i = 0; i < count; ++i) {
19             points[i] = new FunctionPoint(leftX + i * step, values[i]);
20         }
21     }
22     public double getLeftDomainBorder() {
23         return points[0].getX();
24     }
25     public double getRightDomainBorder() {
26         return points[points.length - 1].getX();
27     }
28     public double getFunctionValue(double x) {
29         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
30             return Double.NaN;
31         }
32         for (int i = 0; i < points.length - 1; ++i) {
33             if (points[i].getX() <= x && x <= points[i + 1].getX()) {
34                 double x1 = points[i].getX();
35                 double y1 = points[i].getY();
36                 double x2 = points[i + 1].getX();
37                 double y2 = points[i + 1].getY();
38                 return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
39             }
40         }
41         return Double.NaN;
42     }
43 }
```

Задание 5

В классе `TabulatedFunction` я описал методы, необходимые для работы с точками табулированной функции. Нумерация точек начинается с нуля.

Метод `int getPointsCount()` возвращает количество точек.

Метод `FunctionPoint getPoint(int index)` возвращает копию точки, соответствующей переданному индексу. Возвращение ссылки на саму точку противоречит принципу инкапсуляции.

Метод `void setPoint(int index, FunctionPoint point)` заменяет указанную точку табулированной функции на переданную. Для корректной инкапсуляции я заменил на копию переданной точки. В случае, если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции, то замену точки не проводится.

Метод `double getPointX(int index)` возвращает значение абсциссы точки с указанным номером.

Метод `void setPointX(int index, double x)` изменяет значение абсциссы точки с указанным номером. Аналогично методу `setPoint()`, данный метод не изменяет точку, если новое значение попадает в другой интервал табулирования.

Метод `double getPointY(int index)` возвращает значение ординаты точки с указанным номером.

Метод `void setPointY(int index, double y)` изменяет значение ординаты точки с указанным номером.

```
44 public int getPointsCount() {
45     return points.length;
46 }
47 public FunctionPoint getPoint(int index) {
48     return new FunctionPoint(points[index]);
49 }
50 public void setPoint(int index, FunctionPoint point) {
51     double newX = point.getX();
52     double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
53     double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
54
55     if (newX > leftBound && newX < rightBound) {
56         points[index] = new FunctionPoint(point);
57     }
58 }
59 public double getPointX(int index) {
60     return points[index].getX();
61 }
62 public void setPointX(int index, double x) {
63     double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
64     double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
65
66     if (x > leftBound && x < rightBound) {
67         points[index].setX(x);
68     }
69 }
70 public double getPointY(int index) {
71     return points[index].getY();
72 }
73 public void setPointY(int index, double y) {
74     points[index].setY(y);
75 }
76 }
77 }
```



```

1 package TabulatedFunction;
2
3 // Описание табулированной функции, заданной таблицей точек.
4 public class TabulatedFunction {
5     private FunctionPoint[] points;
6     public TabulatedFunction(double leftX, double rightX, int pointCount) { // Создать объект табулированной функции по заданным левым и правым границам области определения, а также количеству точек для табулирования (значения функции в точках при этом равны 0)
7         this.points = new FunctionPoint[pointCount];
8         double step = (rightX - leftX) / (pointCount - 1);
9         for (int i = 0; i < pointCount; ++i) {
10             points[i] = new FunctionPoint(leftX + i * step, 0);
11         }
12     }
13     public TabulatedFunction(double leftX, double rightX, double[] values) { // Аналогично предыдущей конструкции, но вместо количества точек получить значения функции в этих массивах!
14         int count = values.length;
15         this.points = new FunctionPoint[count];
16         double step = (count > 1) ? (rightX - leftX) / (count - 1) : 0;
17         for (int i = 0; i < count; ++i) {
18             points[i] = new FunctionPoint(leftX + i * step, values[i]);
19         }
20     }
21     public double getLeftDomainBorder() {
22         return points[0].getX();
23     }
24     public double getRightDomainBorder() {
25         return points[points.length - 1].getX();
26     }
27     public double getFunctionValue(double x) {
28         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
29             return Double.NaN;
30         }
31         for (int i = 0; i < points.length - 1; ++i) {
32             if (points[i].getX() <= x && x <= points[i + 1].getX()) {
33                 double x1 = points[i].getX();
34                 double y1 = points[i].getY();
35                 double x2 = points[i + 1].getX();
36                 double y2 = points[i + 1].getY();
37                 return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
38             }
39         }
40         return Double.NaN;
41     }
42     public int getPointCount() {
43         return points.length;
44     }
45     public FunctionPoint getPoint(int index) {
46         return new FunctionPoint(points[index]);
47     }
48     public void setPoint(int index, FunctionPoint point) {
49         double newX = point.getX();
50         double newY = point.getY();
51         double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
52         double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
53         if (newX > leftBound && newX < rightBound) {
54             points[index] = new FunctionPoint(point);
55         }
56     }
57     public double getPointX(int index) {
58         return points[index].getX();
59     }
60     public void setPointX(int index, double x) {
61         double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
62         double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
63         if (x > leftBound && x < rightBound) {
64             points[index].setX(x);
65         }
66     }
67     public double getPointY(int index) {
68         return points[index].getY();
69     }
70     public void setPointY(int index, double y) {
71         points[index].setY(y);
72     }
73 }

```

Задание 6

В классе `TabulatedFunction` я описал методы, изменяющие количество точек табулированной функции.

Метод `void deletePoint(int index)` удаляет заданную точку табулированной функции.

Метод `void addPoint(FunctionPoint point)` добавляет новую точку табулированной функции. При написании метода я обеспечил корректную инкапсуляцию.

При написании методов я учитывал, что точки в массиве должны быть упорядочены по значению координаты x .

Для копирования участков массивов я воспользовался методом `arraycopy()` класса `System`.

```

75 }
76 // Методы изменения кол-ва точек
77 public void deletePoint(int index) {
78     if (index < 0 || index >= points.length || points.length < 3) return;
79     FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
80     System.arraycopy(points, srcPos:0, newPoints, destPos:0, index);
81     System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
82     points = newPoints;
83 }
84 public void addPoint(FunctionPoint point) {
85     int insertIndex = 0;
86     while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) {
87         insertIndex++;
88     }
89     if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) {
90         return;
91     }
92     FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
93     System.arraycopy(points, srcPos:0, newPoints, destPos:0, insertIndex);
94     newPoints[insertIndex] = new FunctionPoint(point);
95     System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
96     points = newPoints;
97 }
98 }
99

```



```

FunctionPoint.java U  TabulatedFunction.java U  Main.java U X
src > Main.java > ...
1  import functions.FunctionPoint;
2  import functions.TabulatedFunction;
3
4  public class Main {
5      Run | Debug
6      public static void main(String[] args) {
7          System.out.println("Создание функции y=x^2 на отрезке [0, 10]:");
8          double[] values = new double[11];
9          for (int i = 0; i < values.length; ++i) {
10             values[i] = i * i;
11         }
12         TabulatedFunction function = new TabulatedFunction(leftX:0, rightX:10, values);
13         printFunction(function);
14
15         System.out.println("Проверка значений функции:");
16         System.out.println("Значение в точке f(2.0) = " + function.getFunctionValue(x:2.0));
17         System.out.println("Значение между точками f(2.5) = " + function.getFunctionValue(x:2.5));
18         System.out.println("Значение вне области определения f(-1.0) = " + function.getFunctionValue(-1.0));
19         System.out.println();
20
21         System.out.println("Изменение точки:");
22         function.setPoint(index:5, new FunctionPoint(x:5.5, y:30.25));
23         printFunction(function);
24
25         System.out.println("Удаление точки:");
26         function.deletePoint(index:3);
27         printFunction(function);
28
29         System.out.println("Добавление новой точки:");
30         function.addPoint(new FunctionPoint(x:8.5, y:72.25));
31         printFunction(function);
32     }
33     public static void printFunction(TabulatedFunction func) {
34         System.out.println("Табулированная функция из " + func.getPointsCount() + " точек:");
35         for (int i = 0; i < func.getPointsCount(); i++) {
36             FunctionPoint p = func.getPoint(i);
37             System.out.println("Точка " + i + ": (" + p.getX() + "; " + p.getY() + ")");
38         }
39         System.out.println();
40     }
41 }

```

```

PS C:\Users\Borus\Desktop\Буз\Программирование\ООП\Lab2\Lab-2-2025> c:: cd 'c:\Users\Borus\Desktop\Буз\Программирование\ООП\Lab2\Lab-2-2025'; & 'c:\Program Files\Java\jdk-24\bin\java.exe' '-enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'c:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\aacbf2ff35e3318c35668c5dd30fd329\redhat.java\jdt_ws\Lab-2-2025_cd70ed9a\bin' 'Main'
Создание функции y=x^2 на отрезке [0, 10]:
Табулированная функция из 11 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 4.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)
Точка 5: (5.0; 25.0)
Точка 6: (6.0; 36.0)
Точка 7: (7.0; 49.0)
Точка 8: (8.0; 64.0)
Точка 9: (9.0; 81.0)
Точка 10: (10.0; 100.0)

Проверка значений функции:
Значение в точке f(2.0) = 4.0
Значение между точками f(2.5) = 6.5
Значение вне области определения f(-1.0) = NaN

Изменение точки:
Табулированная функция из 11 точек:
Точка 0: (0.0; 0.0)
Точка 1: (1.0; 1.0)
Точка 2: (2.0; 4.0)
Точка 3: (3.0; 9.0)
Точка 4: (4.0; 16.0)
Точка 5: (5.5; 30.25)
Точка 6: (6.0; 36.0)
Точка 7: (7.0; 49.0)
Точка 8: (8.0; 64.0)
Точка 9: (9.0; 81.0)
Точка 10: (10.0; 100.0)

```

```
Удаление точки:  
Табулированная функция из 10 точек:  
Точка 0: (0.0; 0.0)  
Точка 1: (1.0; 1.0)  
Точка 2: (2.0; 4.0)  
Точка 3: (4.0; 16.0)  
Точка 4: (5.5; 30.25)  
Точка 5: (6.0; 36.0)  
Точка 6: (7.0; 49.0)  
Точка 7: (8.0; 64.0)  
Точка 8: (9.0; 81.0)  
Точка 9: (10.0; 100.0)
```

```
Добавление новой точки:  
Табулированная функция из 11 точек:  
Точка 0: (0.0; 0.0)  
Точка 1: (1.0; 1.0)  
Точка 2: (2.0; 4.0)  
Точка 3: (4.0; 16.0)  
Точка 4: (5.5; 30.25)  
Точка 5: (6.0; 36.0)  
Точка 6: (7.0; 49.0)  
Точка 7: (8.0; 64.0)  
Точка 8: (8.5; 72.25)  
Точка 9: (9.0; 81.0)  
Точка 10: (10.0; 100.0)
```

```
PS C:\Users\Borus\Desktop\язы\Программирование\ООП\Lab2\Lab-2-2025>
```

for Turbo! 0.0.0 0.0.0 Live Share Java: Ready Ln 41, Col 1 Spaces: 4 UTF-8 CRLF () Java 65 Go Live Quokka Prettier