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 № 3

«Доработка пакета для работы с табулированными функциями: исключения, связные списки и интерфейсы»

по курсу
Объектно-ориентированное программирование

Выполнила: Яньшина Анастасия Юрьевна
Группа 6203-010302D

Содержание

Задание 1

Перед началом работы непосредственно с кодом, я, как и прописано в задании, изучила следующие классы исключений, входящие в API Java:

- `java.lang.Exception` — базовый класс для всех проверяемых исключений;
- `java.lang.IndexOutOfBoundsException` — выход за границы индекса;
- `java.lang.ArrayIndexOutOfBoundsException` — специализированное для массивов;
- `java.lang.IllegalArgumentException` — неверный аргумент метода;
- `java.lang.IllegalStateException` — неверное состояние объекта.

После этого стали понятны иерархия исключений и их назначение для различных сценариев ошибок.

Задание 2

В пакете `functions` создаём два класса исключений: `FunctionPointIndexOutOfBoundsException`, логично наследует от класса `IndexOutOfBoundsException`, и `InappropriateFunctionPointException` , который наследует от `Exception` (Рис. 1 и рис. 2).

Первая ошибка возникает, когда пытаемся обратиться к несуществующей точке, вторая - когда нарушаем порядок точек.

```
package functions;

28 usages
public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException{

    11 usages
    public FunctionPointIndexOutOfBoundsException() { super(); }

    no usages
    public FunctionPointIndexOutOfBoundsException(String message) { super(message); }

}
```

Рис. 1

```
package functions;

23 usages
public class InappropriateFunctionPointException extends Exception {

    2 usages
    public InappropriateFunctionPointException() { super(); }

    8 usages
    public InappropriateFunctionPointException(String message) { super(message); }
}
```

Рис. 2

Задание 3

В разработанный ранее класс `TabulatedFunction` вносим изменения, обеспечивающие выбрасывание исключений методами класса во всех критических местах. Изменения произошли в конструкторах (Рис. 3), а также практически во всех методах (по сути, кроме тех, которые не принимают на вход никакие параметры и `getFunctionValue`) (Рис. 4, рис.5 и рис.6).

Теперь программа не молча игнорирует ошибки, а сообщает о них.

```
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) throws IllegalArgumentException {

    if (leftX>=rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой!");
    if (pointsCount < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");

    this.points = new FunctionPoint[pointsCount];
    //вычисляем шаг между точками и заполняем массив точек
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        points[i] = new FunctionPoint( x: leftX + i * step,  y: 0);
    }

}

//создаёт объект табулированной функции по заданным левой и правой границе области определения, а также значениям функции
2 usages
public ArrayTabulatedFunction(double leftX, double rightX, double[] values) throws IllegalArgumentException {

    if (leftX>=rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой!");
    if (values.length < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");

    this.points = new FunctionPoint[values.length];
    //вычисляем шаг между точками и заполняем массив точек
    double step = (rightX - leftX) / (values.length - 1);
    for (int i = 0; i < values.length; i++) {
        points[i] = new FunctionPoint( x: leftX + i * step, values[i]);
    }
```

Рис. 3

```java
//возвращает копию точки по указанному индексу
2 usages
public FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index > points.length-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    return new FunctionPoint(points[index]);
}

//устанавливает точку по указанному индексу (с проверкой порядка по X)
4 usages
public void setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
    if (index < 0 || index > points.length-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    double newX = point.getX();
    if (index > 0 && newX<=points[index-1].getX()) throw new InappropriateFunctionPointException("Значение X должно быть больше предыдущей точки!");
    if (index < points.length-1 && newX >= points[index + 1].getX()) throw new InappropriateFunctionPointException("Значение X должно быть меньше следующей точки!");
    points[index] = new FunctionPoint(point);
}

//возвращает координату X точки по указанному индексу
4 usages
public double getPointX(int index) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index > points.length-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    return points[index].getX();
}
```

Рис. 4

```java
//устанавливает координату X точки по указанному индексу (с проверкой порядка)
1 usage
public void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
    if (index < 0 || index > points.length-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    if (index > 0 && x <= points[index - 1].getX()) throw new InappropriateFunctionPointException("Значение X должно быть больше предыдущей точки!");
    if (index<points.length-1 && x >= points[index + 1].getX()) throw new InappropriateFunctionPointException("Значение X должно быть меньше следующей точки!");
    points[index].setX(x);
}

//возвращает координату Y точки по указанному индексу
1 usage
public double getPointY(int index)throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index > points.length-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    return points[index].getY();
}

//устанавливает координату Y точки по указанному индексу
5 usages
public void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index > points.length-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    //if (index < points.length && index >= 0)
    points[index].setY(y);
}
```

Рис. 5

```java
//удаляет точку по указанному индексу
3 usages
public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException {
    if (index < 0 || index > points.length-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    if (points.length-1 < 3) throw new IllegalStateException("Невозможно удалить точку! Точек должно быть минимум две!");
    FunctionPoint[] newPoints = new FunctionPoint[points.length-1];
    System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, index);
    System.arraycopy(points, srcPos: index+1, newPoints, index, length: newPoints.length-index);
    points  = newPoints;
}

//добавляет новую точку в массив (с сохранением порядка по X)
2 usages
public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
    //находим позицию для вставки (точки должны быть отсортированы по X)
    int insertIndex = 0;
    while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) insertIndex++;

    //если такая точка уже существует - ничего не добавляем и выбрасываем сообщение об ошибке
    if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) throw new InappropriateFunctionPointException("Точка со значением X = " + point.getX() + " уже существует!");

    FunctionPoint[] newPoints = new FunctionPoint[points.length+1];
    System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, insertIndex);
    newPoints[insertIndex] = new FunctionPoint(point);
    System.arraycopy(points, insertIndex, newPoints, destPos: insertIndex + 1, length: points.length - insertIndex);
    points = newPoints;
}
```

Рис. 6

Задание 4

Это было самое сложное задание. Нужно было создать новую реализацию табулированной функции, но вместо массива использовать связный список. По заданию лабораторной список должен быть двусвязным и кольцевым, причём с выделенной головой, которая не хранит в себе данные, но служит маркером начала списка.

Для начала создаём внутренний класс `FunctionNode` (Рис. 7). Этот класс решено было сделать `private static` – доступен только внутри внешнего класса и не зависит от его экземпляра. Все поля `private`, с доступом с помощью геттеров и сеттеров, что обеспечивает инкапсуляцию данных. Также в нём реализовано два конструктора: один для создания узла с данными, другой для пустого узла (Рис. 8). Важным является то, что в пустом конструкторе узел ссылалается сам на себя. Это важно для корректной работы циклического списка.

После этого создаём основной класс. Поля в нём: `head` – это голова списка; `pointsCount` нужно для быстрого определения количества точек без обхода всего списка; А также, в отличии от первого класса табулированной функции `lastAccessedNode` и `lastAccessedIndex` – для оптимизации: чтобы запоминать последний доступный узел, чтобы при последовательном доступе не бегать каждый раз с начала списка (Рис. 9).

Третьим этапом реализуем базовые операции для работы со списком. Исходя их технического задания, в пустом списке голова должна ссылаться сама на себя, поэтому при создании объекта списка я использую отдельный метод для инициализации пустого списка (Рис. 10).

Самым сложным для реализации оказался метод `getNodeByIndex()`, поскольку нужно было сделать его оптимизированно: при поиске нужного узла идти не с самого начала списка, а отталкиваясь от ближайшего доступного узла (Рис. 11).

Также тонкости были при реализации `deleteNodeByIndex(int index)`. Важно было «изолироватнь», отделить удалённый узел от списка и не забыть поменять связи соседей, чтобы не возникало неожиданных ситуаций и метод работал правильно (Рис 12).

Остальные же методы были несколько проще в реализации (Рис 13 и рис. 14).

```
private static class FunctionNode {
    4 usages
    private FunctionPoint point;
    3 usages
    private FunctionNode next;
    3 usages
    private FunctionNode prev;
```

Рис. 7

```
public FunctionNode(FunctionPoint point) { this.point = point; }
3 usages
public FunctionNode() {
    this.point = null;
    this.prev = this;
    this.next = this;
}
17 usages
public FunctionPoint getPoint(){return this.point;}
6 usages
public void setPoint(FunctionPoint point){this.point=point;}

13 usages
public FunctionNode getNext() {return next;}
7 usages
public void setNext(FunctionNode next) {this.next = next;}
6 usages
public FunctionNode getPrev() {return prev;}
7 usages
public void setPrev(FunctionNode prev) {this.prev = prev;}
```

Рис. 8

```
public class LinkedListTabulatedFunction  implements TabulatedFunction {

    21 usages
    private FunctionNode head;
    23 usages
    private int pointsCount;
    6 usages
    private FunctionNode lastAccessedNode;
    10 usages
    private int lastAccessedIndex;
```

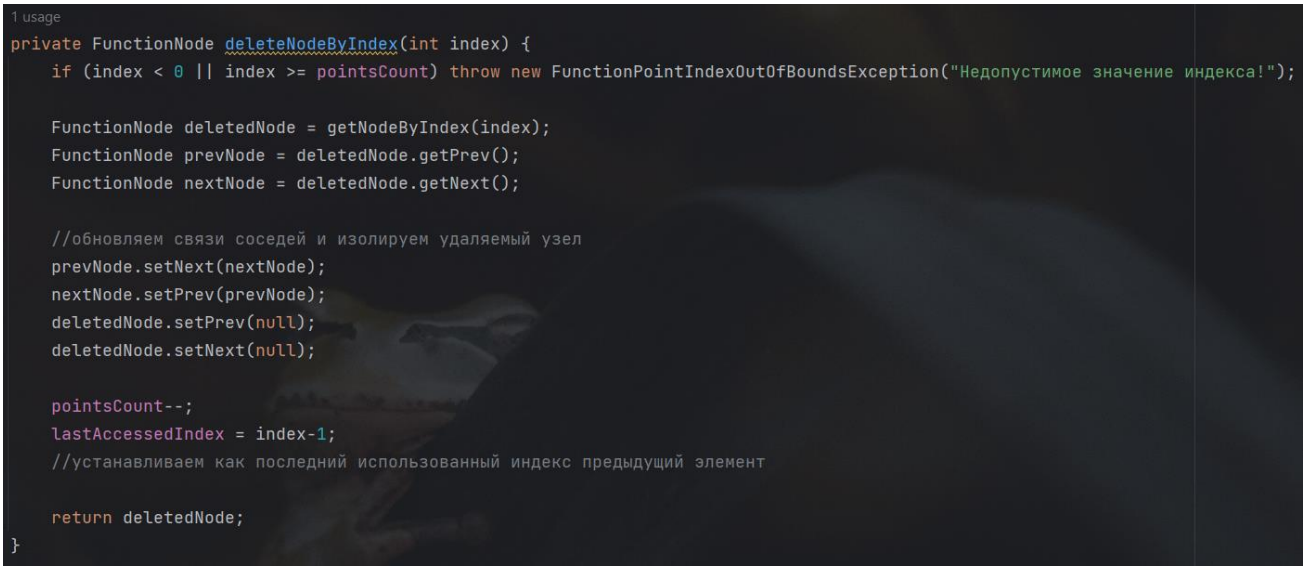
Рис. 9

```java
//инициализация пустого списка - голова указывает сама на себя
2 usages
private void initializeList() {
    head = new FunctionNode(); // голова ссылается сама на себя
    head.setPrev(head);
    head.setNext(head);
    pointsCount = 0;
    lastAccessedNode = head;
    lastAccessedIndex = -1;
}
```

Рис. 10

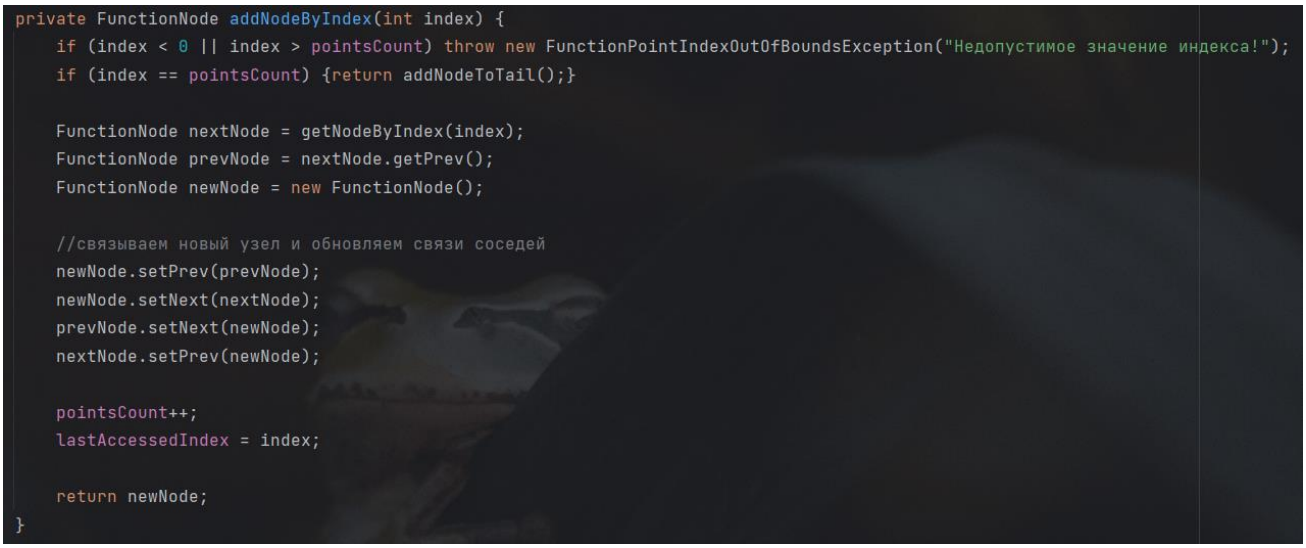
```java
12 usages
private FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");

    //для оптимимзации начинаем с последнего доступного узла
    FunctionNode current;
    int startIndex;

    if (lastAccessedIndex != -1 && Math.abs(index - lastAccessedIndex) < Math.min(index, pointsCount - index)) {
        current = lastAccessedNode;
        startIndex = lastAccessedIndex;
    } else if (index < pointsCount / 2) {
        current = head.getNext();
        startIndex = 0;
    } else {
        current = head.getPrev();
        startIndex = pointsCount - 1;
    }

    //продвигаемся по списку вперед или назад
    if (index > startIndex) {
        for (int i = startIndex; i < index; i++) {
            current = current.getNext();
        }
    } else if (index < startIndex) {
        for (int i = startIndex; i > index; i--) {
            current = current.getPrev();
        }
    }

    //сохраняем данные для следующего доступа
    lastAccessedNode = current;
    lastAccessedIndex = index;

    return current;
}
```

Рис. 11



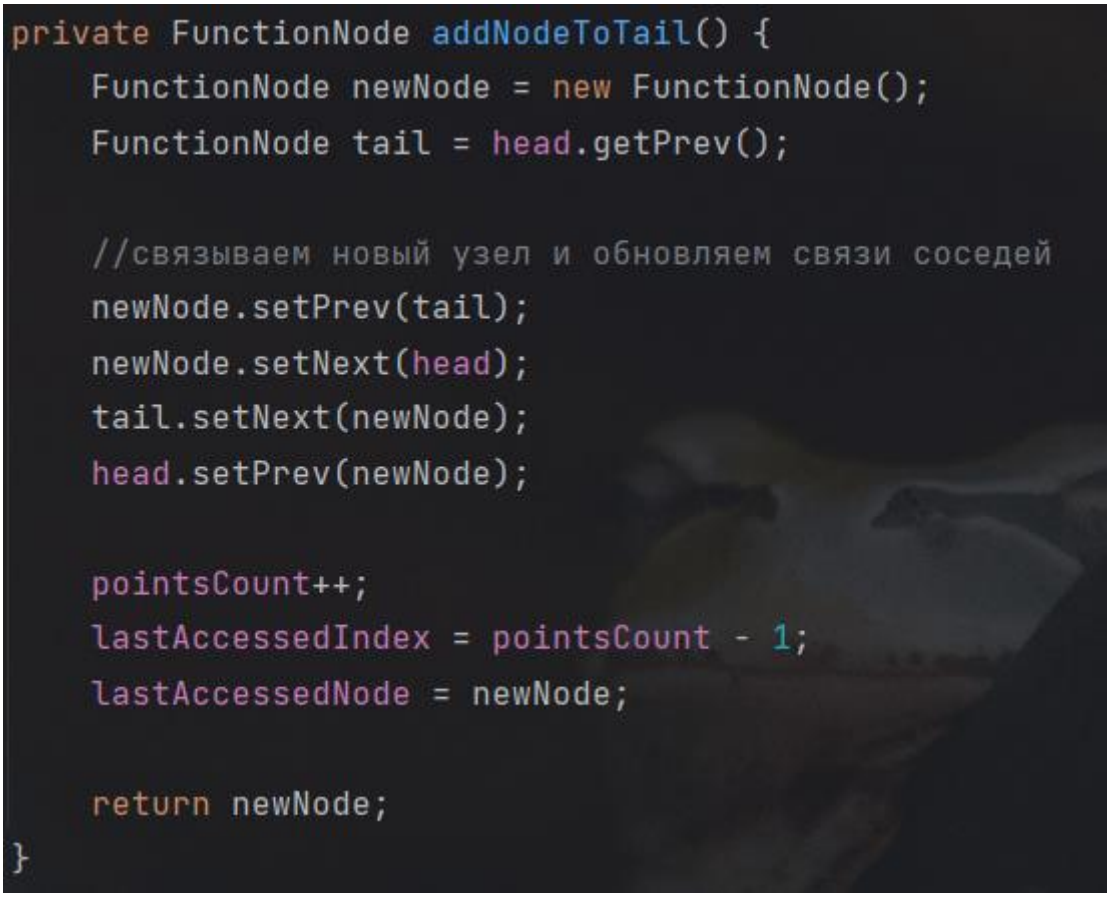
```
1 usage
private FunctionNode deleteNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");

    FunctionNode deletedNode = getNodeByIndex(index);
    FunctionNode prevNode = deletedNode.getPrev();
    FunctionNode nextNode = deletedNode.getNext();

    //обновляем связи соседей и изолируем удаляемый узел
    prevNode.setNext(nextNode);
    nextNode.setPrev(prevNode);
    deletedNode.setPrev(null);
    deletedNode.setNext(null);

    pointsCount--;
    lastAccessedIndex = index-1;
    //устанавливаем как последний использованный индекс предыдущий элемент

    return deletedNode;
}
```

Рис. 12



```
private FunctionNode addNodeByIndex(int index) {
    if (index < 0 || index > pointsCount) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    if (index == pointsCount) {return addNodeToTail();}

    FunctionNode nextNode = getNodeByIndex(index);
    FunctionNode prevNode = nextNode.getPrev();
    FunctionNode newNode = new FunctionNode();

    //связываем новый узел и обновляем связи соседей
    newNode.setPrev(prevNode);
    newNode.setNext(nextNode);
    prevNode.setNext(newNode);
    nextNode.setPrev(newNode);

    pointsCount++;
    lastAccessedIndex = index;

    return newNode;
}
```

Рис. 13



```
private FunctionNode addNodeToTail() {
    FunctionNode newNode = new FunctionNode();
    FunctionNode tail = head.getPrev();

    //связываем новый узел и обновляем связи соседей
    newNode.setPrev(tail);
    newNode.setNext(head);
    tail.setNext(newNode);
    head.setPrev(newNode);

    pointsCount++;
    lastAccessedIndex = pointsCount - 1;
    lastAccessedNode = newNode;

    return newNode;
}
```

Рис. 14

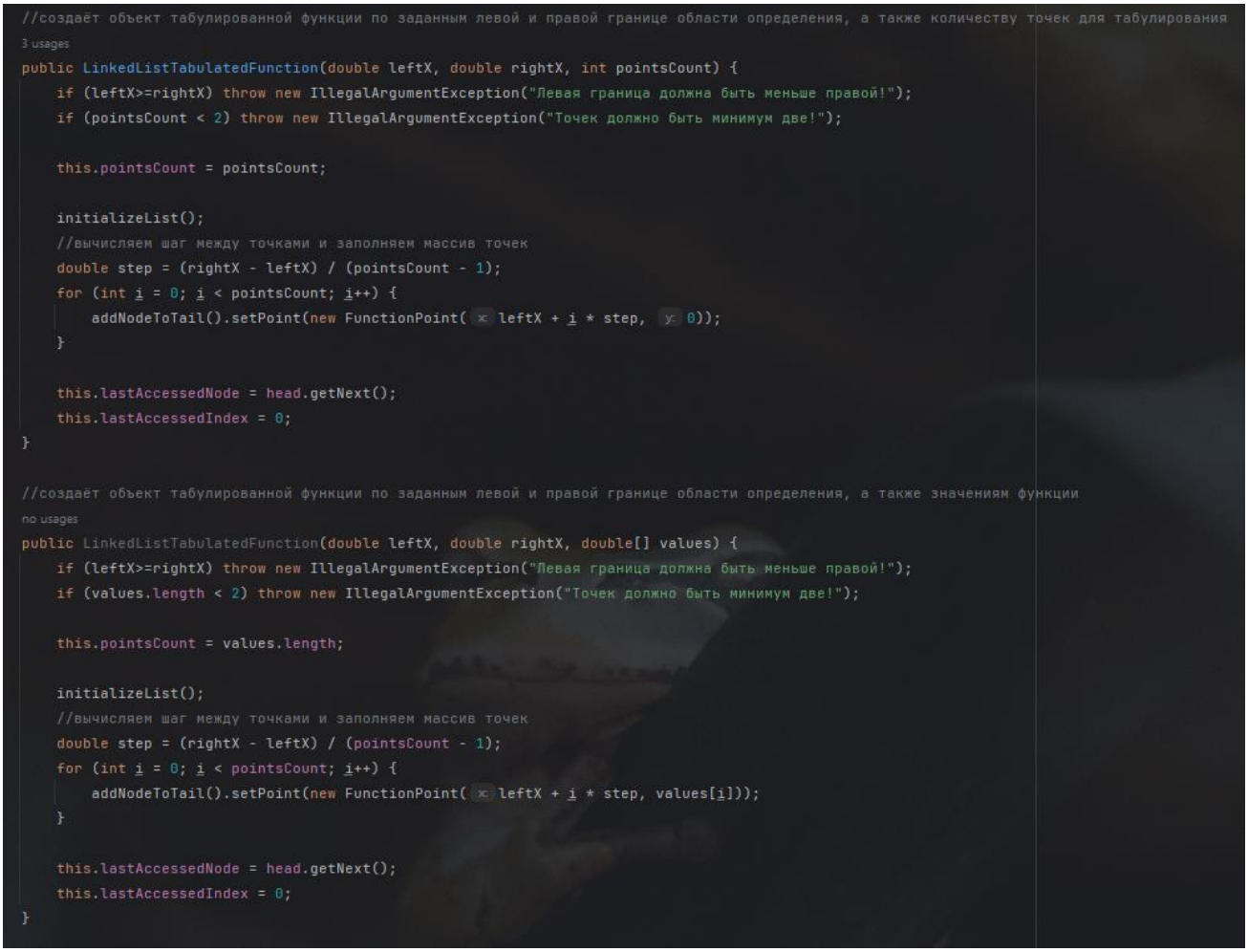
Задание 5

Также добавляем конструкторы и методы, аналогичные конструкторам и методам класса `TabulatedFunction`.

С одной стороны, во многом они похожи на методы уже написанной функции, с другой же – имеют и множество отличий. В конструкторах, например, перед заполнением списка точками, мы инициализируем пустой список с помощью упомянутой выше вспомогательной функции `initializeList()` (Рис. 15).

В методах, возвращающих границы области определения, мы пользуемся не первым и последним индексами, а обращаемся к голове, поскольку она связана с концом и началом списка (Рис. 16). А в методах `getFunctionValue(double x)` и `addPoint(FunctionPoint point)` используем цикл `while`, т.к. список в отличие от массива циклический (Рис. 17 и рис. 18).

Остальные же методы в большинстве своём реализованы с помощью похожих методов именно для работы с узлами списка (например, удаление и добавление узла) (Так же рис. 17 и рис. 18).



```java
//создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования
3 usages
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
    if (leftX>=rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой!");
    if (pointsCount < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");

    this.pointsCount = pointsCount;

    initializeList();
    //вычисляем шаг между точками и заполняем массив точек
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        addNodeToTail().setPoint(new FunctionPoint( x: leftX + i * step,  y: 0));
    }

    this.lastAccessedNode = head.getNext();
    this.lastAccessedIndex = 0;
}

//создаёт объект табулированной функции по заданным левой и правой границе области определения, а также значениям функции
no usages
public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
    if (leftX>=rightX) throw new IllegalArgumentException("Левая граница должна быть меньше правой!");
    if (values.length < 2) throw new IllegalArgumentException("Точек должно быть минимум две!");

    this.pointsCount = values.length;

    initializeList();
    //вычисляем шаг между точками и заполняем массив точек
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        addNodeToTail().setPoint(new FunctionPoint( x: leftX + i * step, values[i]));
    }

    this.lastAccessedNode = head.getNext();
    this.lastAccessedIndex = 0;
}
```

Рис. 15

```java
public double getLeftDomainBorder() {
    if (pointsCount == 0) throw new IllegalStateException("Функция пуста!");
    return head.getNext().getPoint().getX();
}

//возвращает правую границу области определения функции
3 usages
public double getRightDomainBorder() {
    if (pointsCount == 0) throw new IllegalStateException("Функция пуста!");
    return head.getPrev().getPoint().getX();
}

//вычисляет значение функции в заданной точке x с помощью линейной интерполяции
//если x вне области определения, то возвращаем Double.NaN (Not-a-Number)
1 usage
public double getFunctionValue(double x) {
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) return Double.NaN;

    // Поиск интервала, содержащего x
    FunctionNode current = head.getNext();
    //реализуем через while, т.к. список циклический
    while (current != head && current.getNext() != head) {
        double x1 = current.getPoint().getX();
        double x2 = current.getNext().getPoint().getX();
        if (x1 <= x && x <= x2) {
            double y1 = current.getPoint().getY();
            double y2 = current.getNext().getPoint().getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
        current = current.getNext();
    }

    return Double.NaN;
}
```

Рис. 16

```
1 usage
public double getPointY(int index) {return getNodeByIndex(index).getPoint().getY();}

5 usages
public void setPointY(int index, double y) {
    FunctionNode node = getNodeByIndex(index);
    FunctionPoint oldPoint = node.getPoint();
    node.setPoint(new FunctionPoint(oldPoint.getX(), y));
}

3 usages
public void deletePoint(int index) {
    if (pointsCount < 3) throw new IllegalStateException("Невозможно удалить точку! Точек должно быть минимум две!");
    deleteNodeByIndex(index);
}

2 usages
public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
    //поиск позиции для вставки
    int insertIndex = 0;
    FunctionNode current = head.getNext();

    while (current != head && current.getPoint().getX() < point.getX()) {
        insertIndex++;
        current = current.getNext();
    }

    //проверка на дубликат точки
    if (current != head && current.getPoint().getX() == point.getX()) throw new InappropriateFunctionPointException("Точка со значением X = " + point.getX() + " уже существует!");

    //вставка узла
    FunctionNode newNode = addNodeByIndex(insertIndex);
    newNode.setPoint(new FunctionPoint(point));
}
```

Рис. 17

```
//возвращает количество точек табулирования
3 usages
public int getPointsCount() {return this.pointsCount;}

//возвращает копию точки по указанному индексу
2 usages
public FunctionPoint getPoint(int index) {return new FunctionPoint(getNodeByIndex(index).getPoint());}

4 usages
public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
    if (index < 0 || index > pointsCount-1) throw new FunctionPointIndexOutOfBoundsException("Недопустимое значение индекса!");
    double newX = point.getX();

    //проверка порядка
    if (index > 0 && newX <= getNodeByIndex(index - 1).getPoint().getX()) throw new InappropriateFunctionPointException("Значение X должно быть больше предыдущей точки!");
    if (index < pointsCount - 1 && newX >= getNodeByIndex(index + 1).getPoint().getX()) throw new InappropriateFunctionPointException("Значение X должно быть меньше следующей точки!");

    getNodeByIndex(index).setPoint(new FunctionPoint(point));
}

4 usages
public double getPointX(int index) {
    return getNodeByIndex(index).getPoint().getX();
}

1 usage
public void setPointX(int index, double x) throws InappropriateFunctionPointException {
    FunctionNode node = getNodeByIndex(index);

    //проверка порядка
    if (index > 0 && x <= getNodeByIndex(index - 1).getPoint().getX()) throw new InappropriateFunctionPointException("Значение X должно быть больше предыдущей точки!");
    if (index < pointsCount - 1 && x >= getNodeByIndex(index + 1).getPoint().getX()) throw new InappropriateFunctionPointException("Значение X должно быть меньше следующей точки!");

    FunctionPoint oldPoint = node.getPoint();
    node.setPoint(new FunctionPoint(x, oldPoint.getY()));
}
```

Рис. 18

Задание 6

Чтобы унифицировать работу с разными реализациями, создаём общий интерфейс `TabulatedFunction` (Рис. 19). После нужно переименовать `TabulatedFunction` в `ArrayTabulatedFunction`, а также очень важно указать, что оба класса реализуют интерфейс (Рис. 20 и рис. 21).

Теперь можно использовать оба класса одинаково, что очень упрощает работу с ними.

```
package functions;

14 usages  2 implementations
public interface TabulatedFunction {
    3 usages  2 implementations
    double getLeftDomainBorder();
    3 usages  2 implementations
    double getRightDomainBorder();
    1 usage  2 implementations
    double getFunctionValue(double x);
    3 usages  2 implementations
    int getPointsCount();
    2 usages  2 implementations
    FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException;
    4 usages  2 implementations
    void setPoint(int index, FunctionPoint point) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
    4 usages  2 implementations
    double getPointX(int index) throws FunctionPointIndexOutOfBoundsException;
    1 usage  2 implementations
    void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
    1 usage  2 implementations
    double getPointY(int index) throws FunctionPointIndexOutOfBoundsException;
    5 usages  2 implementations
    void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;
    3 usages  2 implementations
    void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
    2 usages  2 implementations
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Рис. 19

```
4 usages
public class ArrayTabulatedFunction implements TabulatedFunction {
```

Рис. 20

```
3 usages
public class LinkedListTabulatedFunction  implements TabulatedFunction {
```

Рис. 21

Задание 7

В последнем задании данной лабораторной требуется проверить работу написанных классов. Для этого я вынесла большую часть кода, которая ранее находилась в `main`, в отдельную функцию проверяющую работу объекта функции в нормальных условиях (без ошибочных параметров) (Рис. 22).

Поскольку у нас есть интерфейс, я реализовала всё это через него – так функция может проверять как работу класса `ArrayTabulatedFunction`, так и работу `LinkedListTabulatedFunction` (Рис. 23 и рис. 24).

Как и в предыдущей лабораторной, для удобства вывода значений я использовала отдельный метод `printFunction(TabulatedFunction function)`, который последовательно выводит значения всех точек переданной в него функции (Рис. 25).

Для проверки исключений реализуем отдельный метод `testExceptions()` (Рис. 26). Я разбила его на четыре части, которые также реализую как отдельные методы:

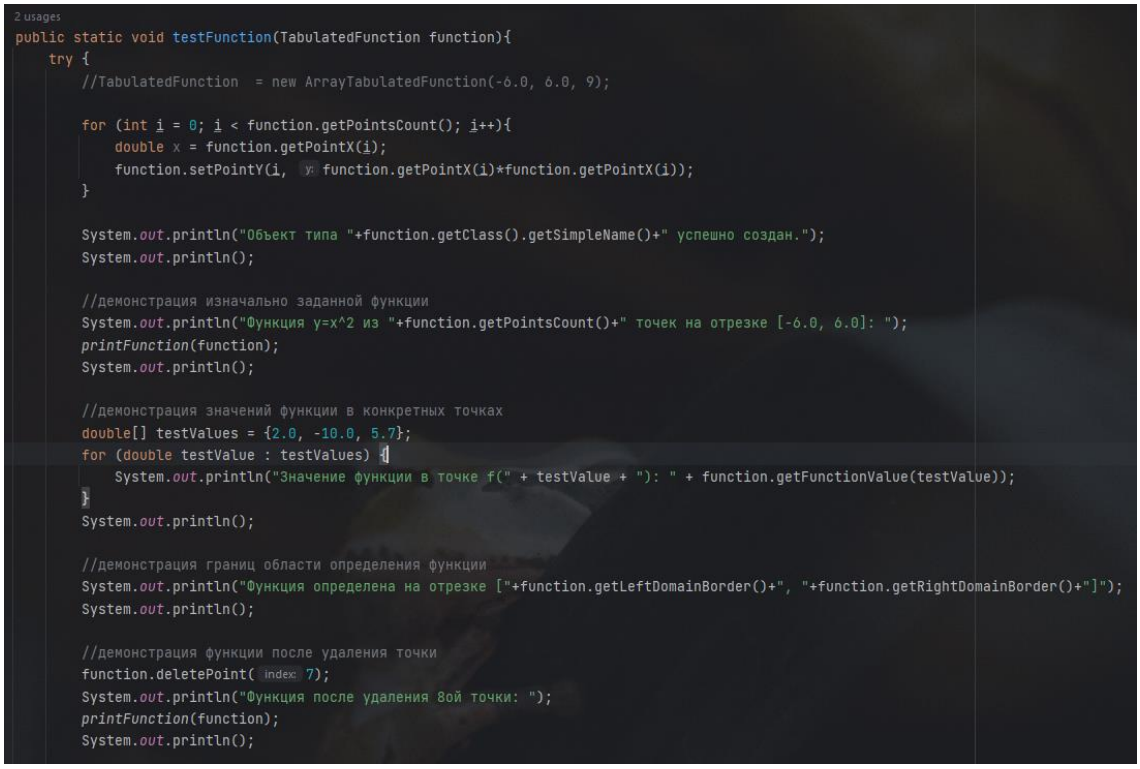
- `testConstructorExceptions()` для тестирования неверных параметров конструктора (Рис. 27);
- `testIndexExceptions()` для тестирования неверных индексов (Рис. 28);
- `testOrderExceptions()` для тестирования нарушения порядка точек (Рис. 29);

- `testMinimumPointsExceptions()` для тестирования операций с минимальным количеством точек (Рис. 30).

При успешном тестировании в консоль выводится текст, аналогичный выводу в прошлой лабораторной, поскольку по сути мы делаем всё то же самое: проверяем последовательно все методы и выводим нужные значения в консоль (Рис. 31, 32 для первого класса и рис 33, 34 для второго). Отличается только то, при успешном выполнении мы увидим данный текст два раза, поскольку вызываем `testFunction(TabulatedFunction function)` сначала для экземпляра типа `ArrayTabulatedFunction`, а затем для `LinkedListTabulatedFunction`. При тестировании же исключительных ситуаций я создавала объекты обоих классов в случайных тестах, поэтому и вывод лишь один (Рис. 35).

```
import functions.*;

public class Main {
    public static void main(String[] args) {
        //создаём экземпляры класса табулированной функции и заполняем их значениями для y=x^2
        TabulatedFunction arrayFunction = new ArrayTabulatedFunction( leftX: -6.0, rightX: 6.0, pointsCount: 9);
        TabulatedFunction listFunction = new LinkedListTabulatedFunction( leftX: -6.0, rightX: 6.0, pointsCount: 9);
        testFunction(arrayFunction);
        testFunction(listFunction);
        testExceptions();
    }
```

Рис. 22

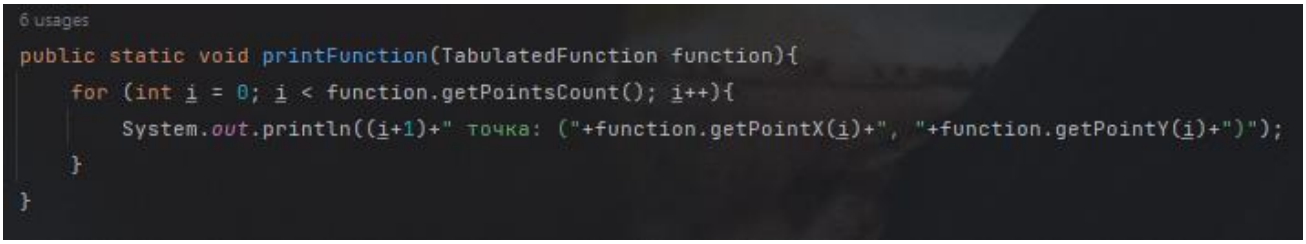


```
2 usages
public static void testFunction(TabulatedFunction function){
    try {
        //TabulatedFunction  = new ArrayTabulatedFunction(-6.0, 6.0, 9);

        for (int i = 0; i < function.getPointsCount(); i++){
            double x = function.getPointX(i);
            function.setPointY(i,  y: function.getPointX(i)*function.getPointX(i));
        }

        System.out.println("Объект типа "+function.getClass().getSimpleName()+" успешно создан.");
        System.out.println();

        //демонстрация изначально заданной функции
        System.out.println("Функция y=x^2 из "+function.getPointsCount()+" точек на отрезке [-6.0, 6.0]: ");
        printFunction(function);
        System.out.println();

        //демонстрация значений функции в конкретных точках
        double[] testValues = {2.0, -10.0, 5.7};
        for (double testValue : testValues) {
            System.out.println("Значение функции в точке f(" + testValue + "): " + function.getFunctionValue(testValue));
        }
        System.out.println();

        //демонстрация границ области определения функции
        System.out.println("Функция определена на отрезке ["+function.getLeftDomainBorder()+", "+function.getRightDomainBorder()+"]");
        System.out.println();

        //демонстрация функции после удаления точки
        function.deletePoint( index: 7);
        System.out.println("Функция после удаления 8ой точки: ");
        printFunction(function);
        System.out.println();
```

Рис. 23

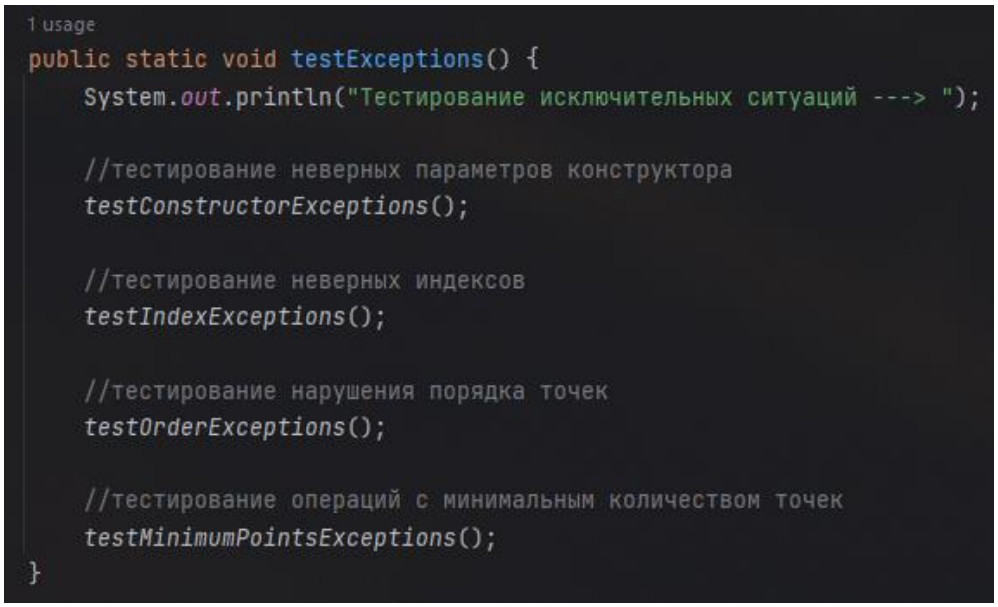
```
        //демонстрация функции после вставки точки
        FunctionPoint testPoint1 = new FunctionPoint( x: -5.8, y: 21);
        function.addPoint(testPoint1);
        System.out.println("Функция после вставки точки с координатами (-5.8, 18.33):");
        printFunction(function);
        System.out.println();

        //демонстрация функции после замены точки
        FunctionPoint testPoint2 = new FunctionPoint( x: 2.8, y: 100);
        function.setPoint( index: 6, testPoint2);
        System.out.println("Функция после замены 7ой точки:");
        printFunction(function);
        System.out.println();

        //демонстрация функции после замены конкретных координат (отдельно x и отдельно y)
        function.setPointX( index: 2,  x: -4.0);
        function.setPointY( index: 5,  y: 12.345);
        System.out.println("Функция после замены координат X у 3 точки и Y у 6 точки: ");
        printFunction(function);
        System.out.println();
    }
    catch(Exception e) {
        System.out.println("Ошибка создания"+function.getClass().getSimpleName()+": " + e.getMessage());
    }
}
```

Рис. 24



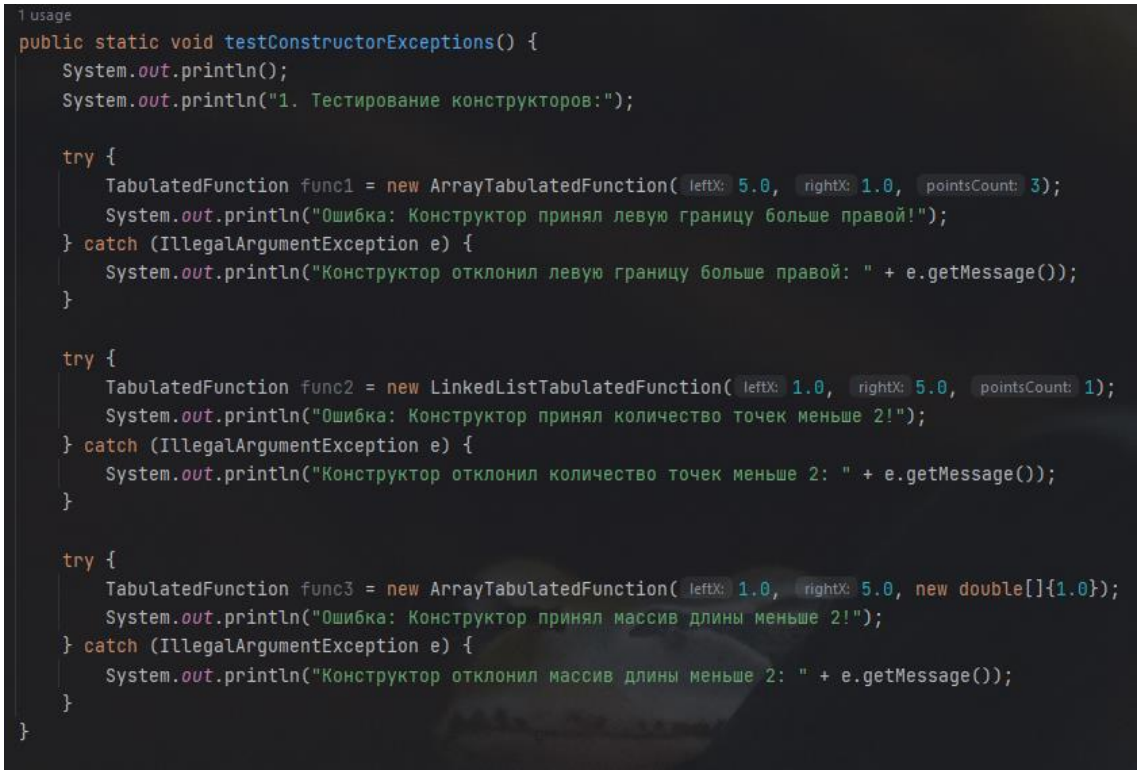
```
6 usages
public static void printFunction(TabulatedFunction function){
    for (int i = 0; i < function.getPointsCount(); i++){
        System.out.println((i+1)+" точка: ("+function.getPointX(i)+", "+function.getPointY(i)+")");
    }
}
```

Рис. 25



```
1 usage
public static void testExceptions() {
    System.out.println("Тестирование исключительных ситуаций ---> ");

    //тестирование неверных параметров конструктора
    testConstructorExceptions();

    //тестирование неверных индексов
    testIndexExceptions();

    //тестирование нарушения порядка точек
    testOrderExceptions();

    //тестирование операций с минимальным количеством точек
    testMinimumPointsExceptions();
}
```

Рис. 26



```
1 usage
public static void testConstructorExceptions() {
    System.out.println();
    System.out.println("1. Тестирование конструкторов:");

    try {
        TabulatedFunction func1 = new ArrayTabulatedFunction( leftX: 5.0,  rightX: 1.0,  pointsCount: 3);
        System.out.println("Ошибка: Конструктор принял левую границу больше правой!");
    } catch (IllegalArgumentException e) {
        System.out.println("Конструктор отклонил левую границу больше правой: " + e.getMessage());
    }

    try {
        TabulatedFunction func2 = new LinkedListTabulatedFunction( leftX: 1.0,  rightX: 5.0,  pointsCount: 1);
        System.out.println("Ошибка: Конструктор принял количество точек меньше 2!");
    } catch (IllegalArgumentException e) {
        System.out.println("Конструктор отклонил количество точек меньше 2: " + e.getMessage());
    }

    try {
        TabulatedFunction func3 = new ArrayTabulatedFunction( leftX: 1.0,  rightX: 5.0, new double[]{1.0});
        System.out.println("Ошибка: Конструктор принял массив длины меньше 2!");
    } catch (IllegalArgumentException e) {
        System.out.println("Конструктор отклонил массив длины меньше 2: " + e.getMessage());
    }
}
```

Рис. 27

```
1 usage
public static void testIndexExceptions() {
    System.out.println();
    System.out.println("2. Тестирование неверных индексов:");

    TabulatedFunction function = createTestFunction();

    try {
        function.getPoint( index: -1);
        System.out.println("Ошибка: Принят отрицательный индекс!");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Отклонён отрицательный индекс: " + e.getMessage());
    }

    try {
        function.getPoint( index: 10);
        System.out.println("Ошибка: Принят слишком большой индекс!");
    } catch (FunctionPointIndexOutOfBoundsException e) {
        System.out.println("Отклонён индекс больше допустимого: " + e.getMessage());
    }

    try {
        function.setPoint( index: 10, new FunctionPoint( x: 3.0,  y: 9.0));
        System.out.println("Ошибка: Принят неверный индекс в setPoint!");
    } catch (FunctionPointIndexOutOfBoundsException | InappropriateFunctionPointException e) {
        System.out.println("Отклонён неверный индекс в setPoint: " + e.getMessage());
    }
}
```

Рис. 28

```
1 usage
public static void testOrderExceptions() {
    System.out.println();
    System.out.println("3. Тестирование нарушения порядка:");

    TabulatedFunction function = createTestFunction();

    try {
        // Пытаемся установить точку с X меньше предыдущей
        function.setPoint( index: 2, new FunctionPoint( x: 0.5,  y: 1.0));
        System.out.println("Ошибка: Принято нарушение порядка (X меньше предыдущего)!");
    } catch (IllegalArgumentException | InappropriateFunctionPointException e) {
        System.out.println("Отклонено нарушение порядка (X меньше предыдущего): " + e.getMessage());
    }

    try {
        // Пытаемся установить точку с X больше следующей
        function.setPoint( index: 1, new FunctionPoint( x: 2.5,  y: 1.0));
        System.out.println("Ошибка: Принято нарушение порядка (X больше следующего)!");
    } catch (IllegalArgumentException | InappropriateFunctionPointException e) {
        System.out.println("Отклонено нарушение порядка (X больше следующего): " + e.getMessage());
    }

    try {
        // Пытаемся добавить точку с существующим X
        function.addPoint(new FunctionPoint( x: 1.0,  y: 5.0));
        System.out.println("Ошибка: Принято добавление точки с существующим X!");
    } catch (InappropriateFunctionPointException e) {
        System.out.println("Отклонено добавление точки с существующим X: " + e.getMessage());
    }
}
```

Рис. 29

```
1 usage
public static void testMinimumPointsExceptions() {
    System.out.println();
    System.out.println("4. Тестирование минимального количества точек:");

    // Создаем функцию с минимальным количеством точек (2)
    TabulatedFunction function = new ArrayTabulatedFunction( leftX: 0.0,  rightX: 2.0, new double[]{0.0, 4.0});

    try {
        function.deletePoint( index: 0);
        System.out.println("Ошибка: Принято удаление при количестве точек равном 2!");
    } catch (IllegalStateException e) {
        System.out.println("Отклонено удаление при количестве точек равном 2: " + e.getMessage());
    }

    try {
        function.deletePoint( index: 1);
        System.out.println("Ошибка: Принято удаление при количестве точек равном 2!");
    } catch (IllegalStateException e) {
        System.out.println("Отклонено удаление при количестве точек равном 2: " + e.getMessage());
    }

    // Проверяем, что функция осталась неизменной
    System.out.println("Функция после попыток удаления (должна остаться неизменной):");
    printFunction(function);
}

//вспомогательный метод для создания тестовой функции
2 usages
public static TabulatedFunction createTestFunction() {
    TabulatedFunction function = new LinkedListTabulatedFunction( leftX: 0.0,  rightX: 2.0,  pointsCount: 3);
    function.setPointY( index: 0,  y: 0.0);
    function.setPointY( index: 1,  y: 1.0);
    function.setPointY( index: 2,  y: 4.0);
    return function;
}
```

Рис. 30

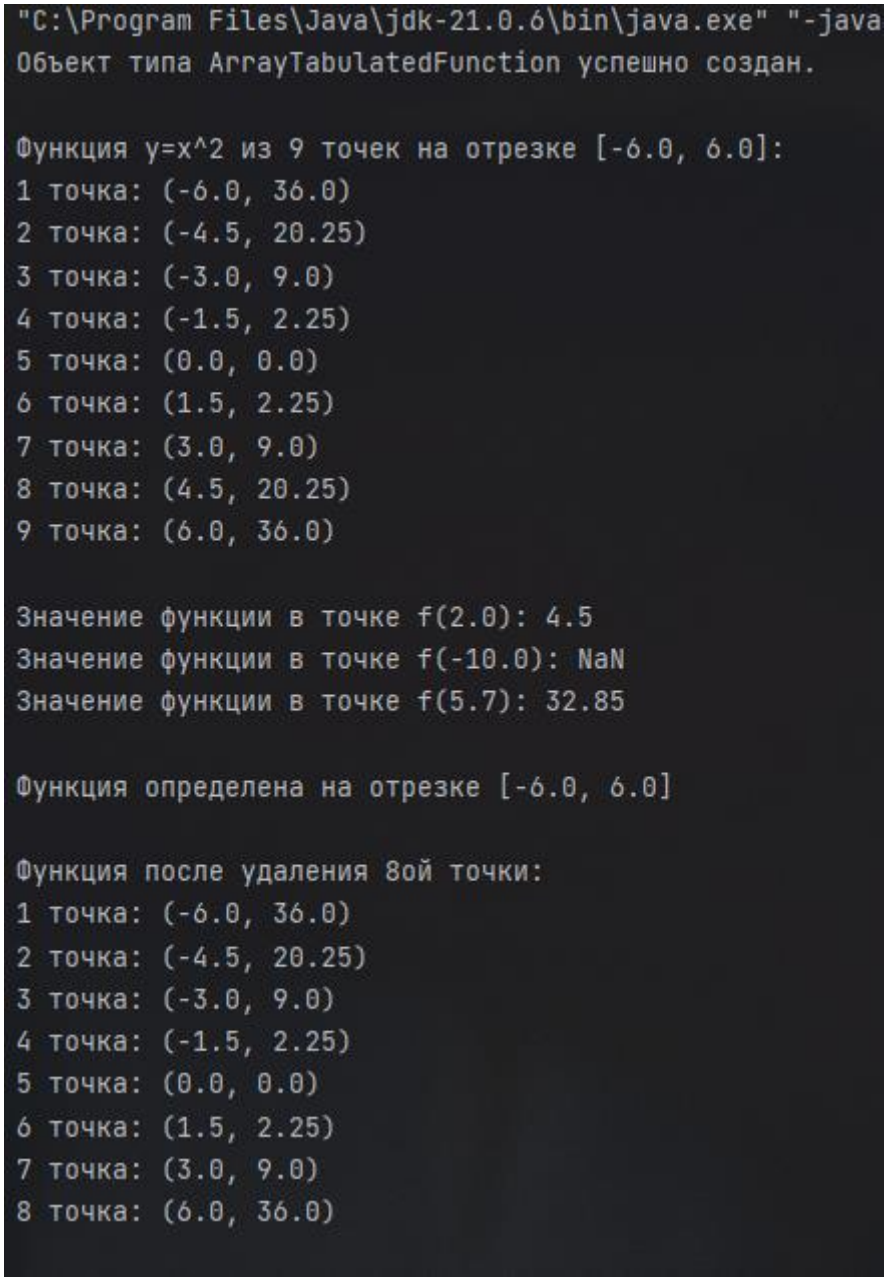

"C:\Program Files\Java\jdk-21.0.6\bin\java.exe" "-java
Объект типа ArrayTabulatedFunction успешно создан.
Функция y=x^2 из 9 точек на отрезке [-6.0, 6.0]:
1 точка: (-6.0, 36.0)
2 точка: (-4.5, 20.25)
3 точка: (-3.0, 9.0)
4 точка: (-1.5, 2.25)
5 точка: (0.0, 0.0)
6 точка: (1.5, 2.25)
7 точка: (3.0, 9.0)
8 точка: (4.5, 20.25)
9 точка: (6.0, 36.0)
Значение функции в точке f(2.0): 4.5
Значение функции в точке f(-10.0): NaN
Значение функции в точке f(5.7): 32.85
Функция определена на отрезке [-6.0, 6.0]
Функция после удаления 8ой точки:
1 точка: (-6.0, 36.0)
2 точка: (-4.5, 20.25)
3 точка: (-3.0, 9.0)
4 точка: (-1.5, 2.25)
5 точка: (0.0, 0.0)
6 точка: (1.5, 2.25)
7 точка: (3.0, 9.0)
8 точка: (6.0, 36.0)

Рис. 31

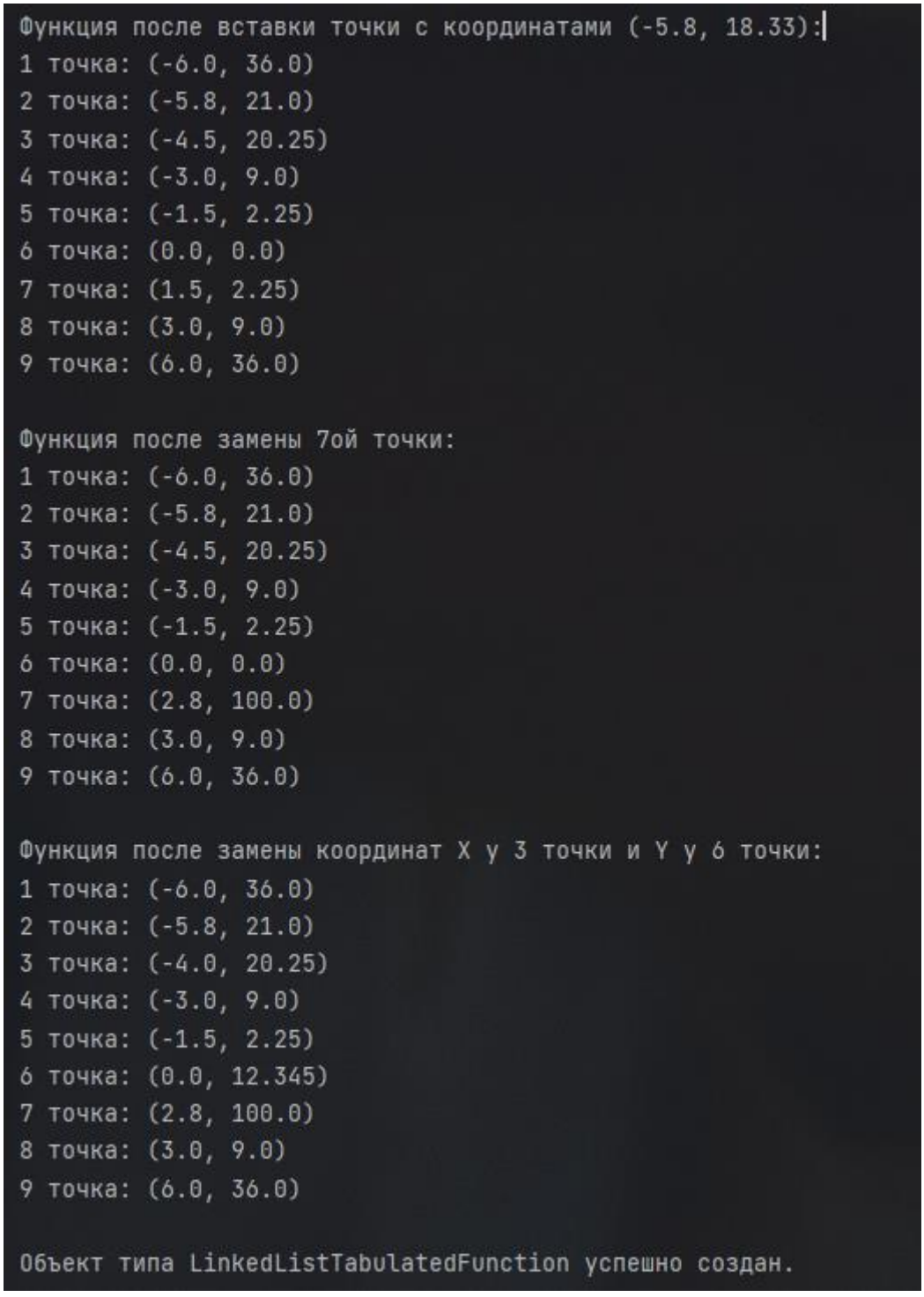

Функция после вставки точки с координатами (-5.8, 18.33):
1 точка: (-6.0, 36.0)
2 точка: (-5.8, 21.0)
3 точка: (-4.5, 20.25)
4 точка: (-3.0, 9.0)
5 точка: (-1.5, 2.25)
6 точка: (0.0, 0.0)
7 точка: (1.5, 2.25)
8 точка: (3.0, 9.0)
9 точка: (6.0, 36.0)
Функция после замены 7ой точки:
1 точка: (-6.0, 36.0)
2 точка: (-5.8, 21.0)
3 точка: (-4.5, 20.25)
4 точка: (-3.0, 9.0)
5 точка: (-1.5, 2.25)
6 точка: (0.0, 0.0)
7 точка: (2.8, 100.0)
8 точка: (3.0, 9.0)
9 точка: (6.0, 36.0)
Функция после замены координат X у 3 точки и Y у 6 точки:
1 точка: (-6.0, 36.0)
2 точка: (-5.8, 21.0)
3 точка: (-4.0, 20.25)
4 точка: (-3.0, 9.0)
5 точка: (-1.5, 2.25)
6 точка: (0.0, 12.345)
7 точка: (2.8, 100.0)
8 точка: (3.0, 9.0)
9 точка: (6.0, 36.0)
Объект типа LinkedListTabulatedFunction успешно создан.

Рис. 32

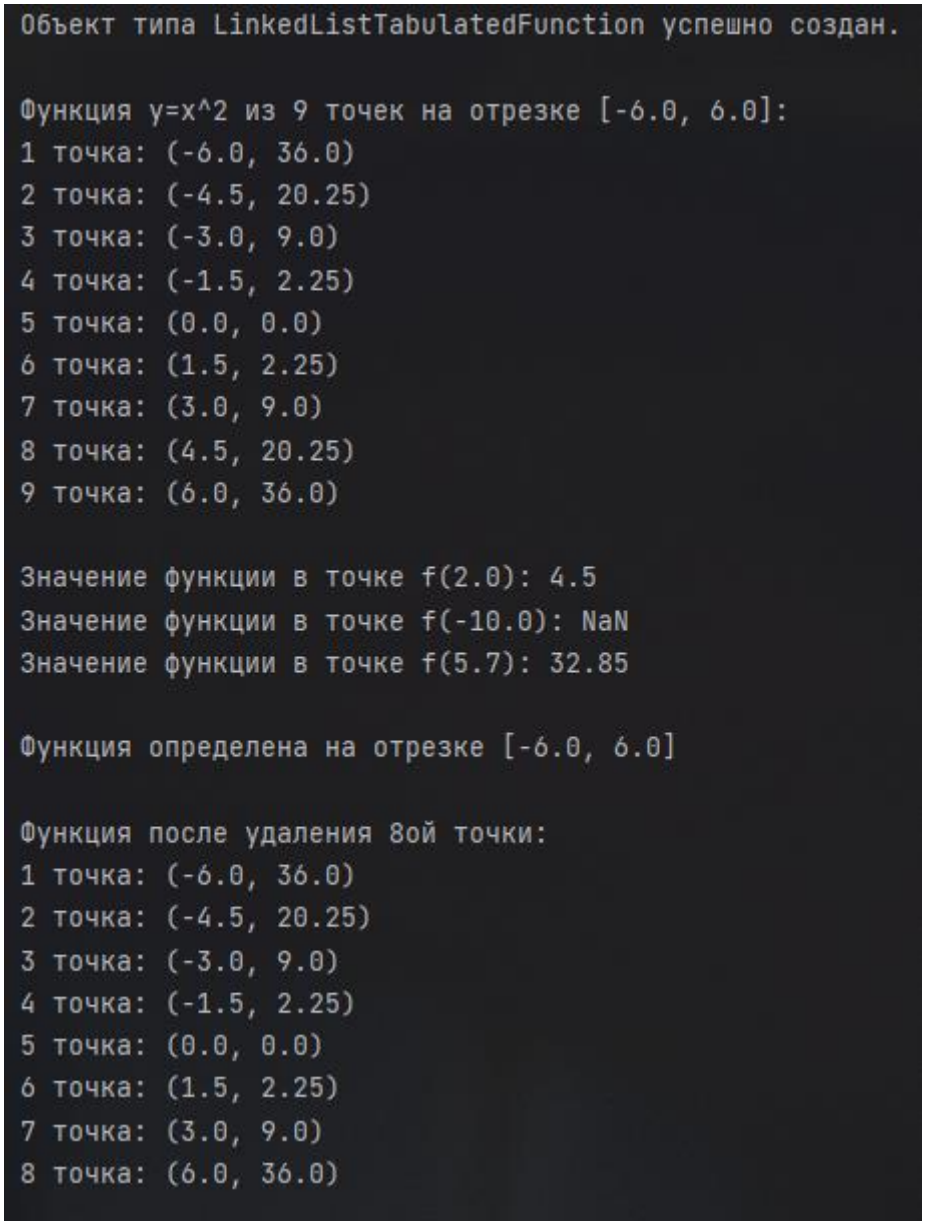

Объект типа LinkedListTabulatedFunction успешно создан.
Функция y=x^2 из 9 точек на отрезке [-6.0, 6.0]:
1 точка: (-6.0, 36.0)
2 точка: (-4.5, 20.25)
3 точка: (-3.0, 9.0)
4 точка: (-1.5, 2.25)
5 точка: (0.0, 0.0)
6 точка: (1.5, 2.25)
7 точка: (3.0, 9.0)
8 точка: (4.5, 20.25)
9 точка: (6.0, 36.0)
Значение функции в точке f(2.0): 4.5
Значение функции в точке f(-10.0): NaN
Значение функции в точке f(5.7): 32.85
Функция определена на отрезке [-6.0, 6.0]
Функция после удаления 8ой точки:
1 точка: (-6.0, 36.0)
2 точка: (-4.5, 20.25)
3 точка: (-3.0, 9.0)
4 точка: (-1.5, 2.25)
5 точка: (0.0, 0.0)
6 точка: (1.5, 2.25)
7 точка: (3.0, 9.0)
8 точка: (6.0, 36.0)

Рис. 33

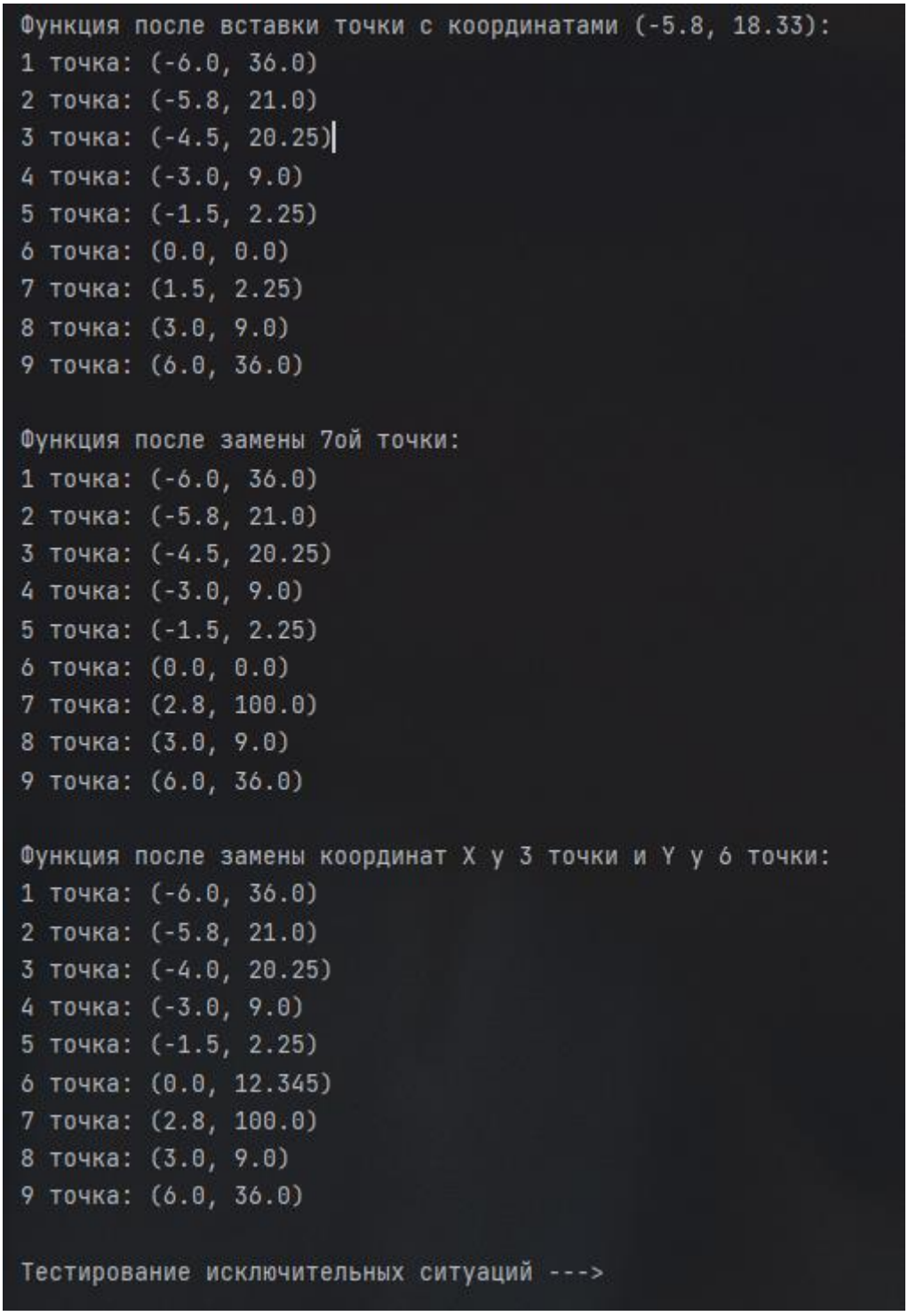

Функция после вставки точки с координатами (-5.8, 18.33):
1 точка: (-6.0, 36.0)
2 точка: (-5.8, 21.0)
3 точка: (-4.5, 20.25)
4 точка: (-3.0, 9.0)
5 точка: (-1.5, 2.25)
6 точка: (0.0, 0.0)
7 точка: (1.5, 2.25)
8 точка: (3.0, 9.0)
9 точка: (6.0, 36.0)
Функция после замены 7ой точки:
1 точка: (-6.0, 36.0)
2 точка: (-5.8, 21.0)
3 точка: (-4.5, 20.25)
4 точка: (-3.0, 9.0)
5 точка: (-1.5, 2.25)
6 точка: (0.0, 0.0)
7 точка: (2.8, 100.0)
8 точка: (3.0, 9.0)
9 точка: (6.0, 36.0)
Функция после замены координат X у 3 точки и Y у 6 точки:
1 точка: (-6.0, 36.0)
2 точка: (-5.8, 21.0)
3 точка: (-4.0, 20.25)
4 точка: (-3.0, 9.0)
5 точка: (-1.5, 2.25)
6 точка: (0.0, 12.345)
7 точка: (2.8, 100.0)
8 точка: (3.0, 9.0)
9 точка: (6.0, 36.0)
Тестирование исключительных ситуаций --->

Рис. 34

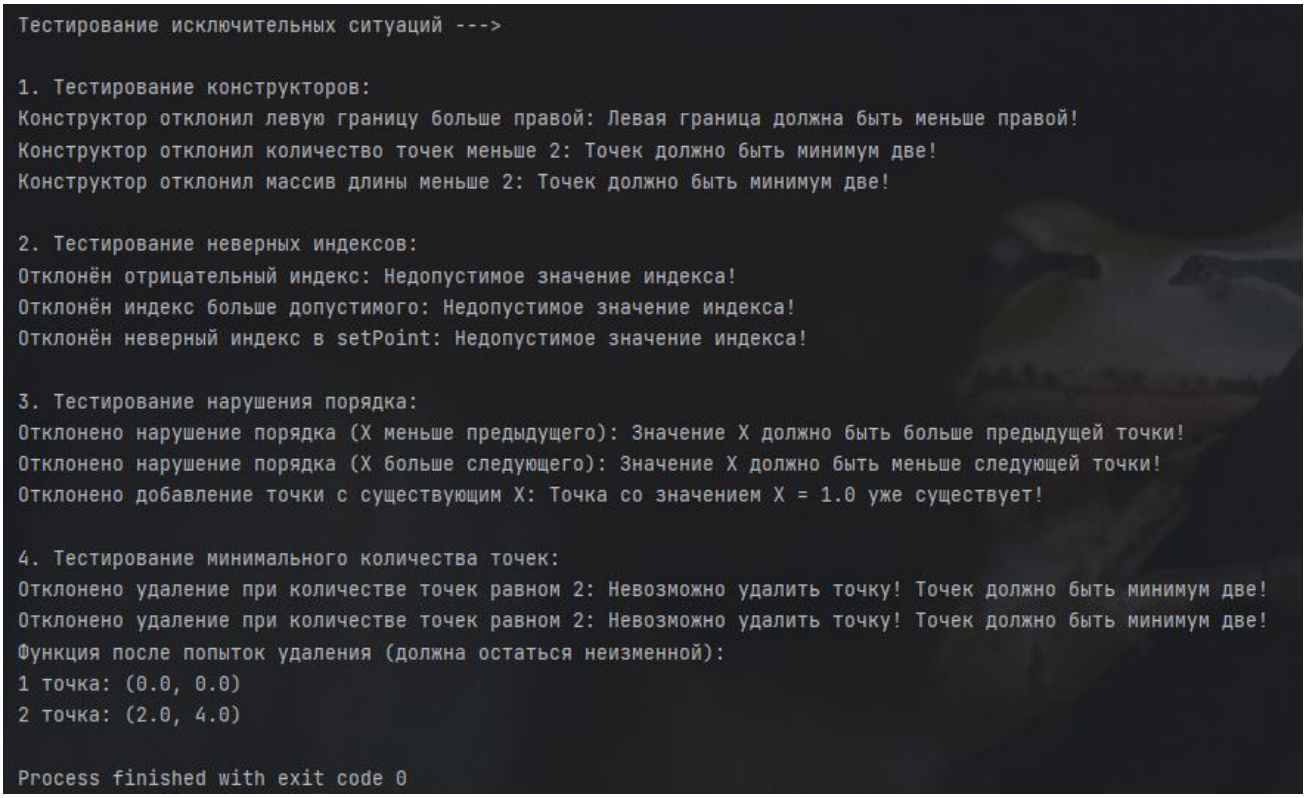

Тестирование исключительных ситуаций --->
1. Тестирование конструкторов:
Конструктор отклонил левую границу больше правой: Левая граница должна быть меньше правой!
Конструктор отклонил количество точек меньше 2: Точек должно быть минимум две!
Конструктор отклонил массив длины меньше 2: Точек должно быть минимум две!
2. Тестирование неверных индексов:
Отклонён отрицательный индекс: Недопустимое значение индекса!
Отклонён индекс больше допустимого: Недопустимое значение индекса!
Отклонён неверный индекс в setPoint: Недопустимое значение индекса!
3. Тестирование нарушения порядка:
Отклонено нарушение порядка (X меньше предыдущего): Значение X должно быть больше предыдущей точки!
Отклонено нарушение порядка (X больше следующего): Значение X должно быть меньше следующей точки!
Отклонено добавление точки с существующим X: Точка со значением X = 1.0 уже существует!
4. Тестирование минимального количества точек:
Отклонено удаление при количестве точек равном 2: Невозможно удалить точку! Точек должно быть минимум две!
Отклонено удаление при количестве точек равном 2: Невозможно удалить точку! Точек должно быть минимум две!
Функция после попыток удаления (должна остаться неизменной):
1 точка: (0.0, 0.0)
2 точка: (2.0, 4.0)
Process finished with exit code 0

Рис. 35