Лабораторная работа №3

Манякин Степан 6204-010302D

Задание 2

> В пакете functions созданы два класса исключений: FunctionPointIndexOutOfBoundsException - исключение выхода за границы набора точек при обращении к ним по номеру и InappropriateFunctionPointException - исключение, выбрасываемое при попытке добавления или изменения точки функции несоответствующим образом

```
package functions;


public class FunctionPointIndexOutOfBoundsException extends
  IndexOutOfBoundsException {
    public FunctionPointIndexOutOfBoundsException() {
        super("Индекс точки вне диапазона допустимых значений");
    }


    public FunctionPointIndexOutOfBoundsException(String message) {
        super(message);
    }
}
```

```
package functions;


public class InappropriateFunctionPointException extends RuntimeException {
    public InappropriateFunctionPointException() {
        super("Некорректная операция с точкой функции");
    }


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

}

Задание 3

Модификация класса TabulatedFunction (переименован в ArrayTabulatedFunction)

В рамках задания была реализована модификация класса, обеспечивающая корректное поведение и обработку исключений:

1. **Конструкторы класса**:

ArrayTabulatedFunction(double leftX, double rightX, int pointsCount)

ArrayTabulatedFunction(double leftX, double rightX, double[] values)

Оба конструктора:

Выбрасывают IllegalArgumentException, если leftX >= rightX или pointsCount < 2.

Создают массив points типа FunctionPoint[] с равномерным распределением точек по оси X.

Конструктор с массивом values инициализирует Y-координаты соответствующими значениями.

```
// конструктор 1: равномерное распределение точек по X
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
    if (leftX >= rightX)
        throw new IllegalArgumentException("левая граница >= правая
граница");
    if (pointsCount < 2)
        throw new IllegalArgumentException("Кол-во точек < 2");

    this.pointsCount = pointsCount;
    points = new FunctionPoint[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1); // шаг по X между
точками
    for (int i = 0; i < pointsCount; i++)
        points[i] = new FunctionPoint(leftX + i * step, 0); // создаем точки
с Y=0
}

// конструктор 2: по массиву Y-значений
public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
    if (leftX >= rightX)
        throw new IllegalArgumentException("левая граница >= правая
граница");
    if (values.length < 2)
        throw new IllegalArgumentException("Кол-во точек < 2");

    pointsCount = values.length;
```

```
    points = new FunctionPoint[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1); // шаг по X между
точками
    for (int i = 0; i < pointsCount; i++)
        points[i] = new FunctionPoint(leftX + i * step, values[i]); //
создаем точки с заданными Y
```

Методы getPoint(), setPoint(), getPointX(), setPointX(), getPointY(), setPointY() и deletePoint() выбрасывают
исключение FunctionPointIndexOutOfBoundsException, если переданный в метод номер выходит за границы набора точек. Это обеспечит корректность обращений к точкам функции.

```
private void checkIndex(int index) {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Индекс" + index
+ "выходит за границы");
    }

    // оптимизированный доступ к точке с использованием кэша
    private FunctionPoint getCachedPoint(int index) {
        checkIndex(index);
        if (index == lastAccessedIndex && lastAccessedPoint != null)
            return lastAccessedPoint; // вернуть кэшированную точку
        lastAccessedIndex = index;
        lastAccessedPoint = points[index]; // обновить кэш
        return lastAccessedPoint;
    }

    public FunctionPoint getPoint(int index) {
        return new FunctionPoint(getCachedPoint(index)); // вернуть копию
точки
    }

    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
        checkIndex(index);
        double x = point.getX();
        if ((index > 0 && x <= points[index - 1].getX()) ||
                (index < pointsCount - 1 && x >= points[index + 1].getX()))
            throw new InappropriateFunctionPointException("X вне порядка");
// проверка порядка X
        points[index] = new FunctionPoint(point); // заменить точку
        lastAccessedIndex = index; // обновить кэш
        lastAccessedPoint = points[index];
    }

    public double getPointX(int index) { return getCachedPoint(index).getX();
} // получить X точки
    public double getPointY(int index) { return getCachedPoint(index).getY();
} // получить Y точки

    public void setPointX(int index, double x) throws
InappropriateFunctionPointException {
        checkIndex(index);
        if ((index > 0 && x <= points[index - 1].getX()) ||
                (index < pointsCount - 1 && x >= points[index + 1].getX()))
            throw new InappropriateFunctionPointException("X вне порядка");
// проверка порядка X
        points[index].setX(x); // установить новое X
        lastAccessedIndex = index; // обновить кэш
```

```
        lastAccessedPoint = points[index];
    }

    public void setPointY(int index, double y) {
        checkIndex(index);
        points[index].setY(y); // установить новое Y
        lastAccessedIndex = index; // обновить кэш
        lastAccessedPoint = points[index];
    }

    // добавление новой точки
    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
        for (int i = 0; i < pointsCount; i++)
            if (Math.abs(points[i].getX() - point.getX()) < EPS)
                throw new InappropriateFunctionPointException("дубликат X");
// проверка на дублирование X

        if (pointsCount == points.length) {
            // расширение массива при необходимости
            FunctionPoint[] newPoints = new FunctionPoint[pointsCount + 1];
            System.arraycopy(points, 0, newPoints, 0, pointsCount);
            points = newPoints;
        }

        // найти позицию для вставки
        int index = 0;
        while (index < pointsCount && points[index].getX() < point.getX())
            index++;

        // сдвинуть точки вправо для вставки
        System.arraycopy(points, index, points, index + 1, pointsCount -
index);
        points[index] = new FunctionPoint(point); // вставить точку
        pointsCount++;

        lastAccessedIndex = index; // обновить кэш
        lastAccessedPoint = points[index];
    }
```

В частности, это происходит благодаря методу checkIndex и getCachedPoint. Методы getPoint(int index), getPointX(int index), getPointY(int index) вызывают getCachedPoint(index), который внутри вызывает checkIndex(index). Также важно сказать что при сравнениях чисел формата double используется машинный эпсилон

Методы setPoint() и setPointX() выбрасывает исключение InappropriateFunctionPointException в том случае, если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции. Метод addPoint() также выбрасывает исключение InappropriateFunctionPointException, если в наборе точек функции есть точка, абсцисса которой совпадает с абсциссой добавляемой точки. Это обеспечивает сохранение упорядоченности точек функции

```
    public void deletePoint(int index) {
        checkIndex(index);
        if (pointsCount <= 2)
            throw new IllegalStateException("удаление невозможно: кол-во
точек < 3"); // минимальное количество точек

        // сдвинуть оставшиеся точки влево
        System.arraycopy(points, index + 1, points, index, pointsCount -
index - 1);
        pointsCount--;

        lastAccessedIndex = -1; // сброс кэша
        lastAccessedPoint = null;
    }
}
```

Метод deletePoint() выбрасывает исключение IllegalStateException, если на момент удаления точки количество точек в наборе менее трех. Это обеспечивает невозможность получения функции с некорректным количеством точек.

Задание 4

В пакете functions создаем класс LinkedListTabulatedFunction, объект которого также должен описывать табулированную функцию. Отличие этого класса заключается в том, что для хранения набора точек в нем должен использоваться не массив, а динамическая структура – связный список.

```
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction {
    // вложенный класс узла списка
    private static class FunctionNode {
        FunctionPoint point;     // точка функции
        FunctionNode next;       // ссылка на следующий узел
        FunctionNode prev;       // ссылка на предыдущий узел
        FunctionNode(FunctionPoint p) { point = p; }
    }

    private final FunctionNode head = new FunctionNode(null); // фиктивный
узел (голова)
    private int pointsCount;                                    // текущее
количество точек
    private static final double EPS = Math.ulp(1.0);           // машинный
эпсилон для сравнения double

    // поля для кэширования последнего использованного узла
    private FunctionNode lastAccessedNode = null;
    private int lastAccessedIndex = -1;
```

private FunctionNode head; это основной элемент структуры — фиктивная «голова» двусвязного циклического списка.

Она не хранит данных функции, но служит опорной точкой для организации структуры.

```
private FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException("индекс " + index +
" вне диапазона");

    // если есть кэш и расстояние до него небольшое, идем от кэша
    if (lastAccessedNode != null && Math.abs(index - lastAccessedIndex) <=
pointsCount / 2) {
        FunctionNode node = lastAccessedNode;
        if (index > lastAccessedIndex) {
            for (int i = lastAccessedIndex; i < index; i++)
                node = node.next;
        } else if (index < lastAccessedIndex) {
            for (int i = lastAccessedIndex; i > index; i--)
                node = node.prev;
        }
        lastAccessedNode = node;
        lastAccessedIndex = index;
        return node;
    }

    // обычный обход от головы
    FunctionNode node = head.next;
    for (int i = 0; i < index; i++)
        node = node.next;

    lastAccessedNode = node; // обновляем кэш
    lastAccessedIndex = index;
    return node;
}
```

В классе LinkedListTabulatedFunction реализован приватный метод FunctionNode getNodeByIndex(int index), возвращающий узел списка по его порядковому номеру (нумерация значащих элементов начинается с 0). Метод сначала проверяет корректность индекса и выбрасывает FunctionPointIndexOutOfBoundsException, если индекс выходит за пределы. Для оптимизации доступа реализован механизм кэширования: сохраняются ссылка на последний использованный узел (lastAccessedNode) и его индекс (lastAccessedIndex). Если закэшированный узел существует и расстояние до требуемого индекса не превышает половины длины списка, поиск выполняется от кэшированного узла — двигаясь по полям next или prev. В противном случае выполняется обычный обход от головы списка (head.next). После получения узла кэш обновляется. Такой подход сокращает количество операций при последовательных и локальных обращениях к элементам списка.

```
private FunctionNode addNodeToTail() {
    FunctionNode newNode = new FunctionNode(new FunctionPoint(0, 0)); //
создаем новый узел

    if (head.next == head) { // если список пустой
        head.next = head.prev = newNode;
        newNode.next = newNode.prev = head;
    } else { // вставка в конец
        newNode.next = head;
        newNode.prev = head.prev;
        head.prev.next = newNode;
        head.prev = newNode;
```

```
    }

    pointsCount++;                  // обновляем количество точек
    lastAccessedNode = newNode;     // обновляем кэш
    lastAccessedIndex = pointsCount - 1;
    return newNode;
}
```

Метод добавляет новый элемент в конец двусвязного циклического списка и возвращает ссылку на него.

```
private FunctionNode addNodeByIndex(int index) {
    if (index < 0 || index > pointsCount)
        throw new FunctionPointIndexOutOfBoundsException("индекс" + index +
"вне допустимого диапазона для вставки");

    if (index == pointsCount)           // добавление в конец
        return addNodeToTail();

    FunctionNode currentNode = getNodeByIndex(index);  // узел, перед которым
вставляем
    FunctionNode newNode = new FunctionNode(new FunctionPoint(0, 0));

    // вставка нового узла перед currentNode
    newNode.prev = currentNode.prev;
    newNode.next = currentNode;
    currentNode.prev.next = newNode;
    currentNode.prev = newNode;

    pointsCount++;                      // обновляем количество точек
    if (lastAccessedIndex >= index)     // корректируем кэш
        lastAccessedIndex++;
    lastAccessedNode = newNode;

    return newNode;
}
```

Метод добавляет новый элемент по заданному индексу и возвращает ссылку на него. Если индекс некорректен — выбрасывает исключение. Полностью соответствует заданию.

```
private FunctionNode deleteNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException("индекс" + index +
"вне допустимого диапазона");
    if (pointsCount <= 2)
        throw new IllegalStateException("удаление невозможно: кол-во точек
меньше 3"); // минимум 2 точки оставлять нельзя

    FunctionNode nodeToDelete = getNodeByIndex(index);

    // переподключение соседних узлов
    nodeToDelete.prev.next = nodeToDelete.next;
    nodeToDelete.next.prev = nodeToDelete.prev;

    if (nodeToDelete == head.next)
        head.next = nodeToDelete.next;

    pointsCount--;                      // обновляем количество точек

    // обновление кэша
    if (lastAccessedIndex == index) {
        lastAccessedNode = null;
```

```
            lastAccessedIndex = -1;
        } else if (lastAccessedIndex > index) {
            lastAccessedIndex--;
        }

        return nodeToDelete;
    }
```

Метод удаляет элемент по номеру и возвращает ссылку на удалённый узел.
Выбрасывает нужные исключения при ошибках

Задание 5

```
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction {
    // вложенный класс узла списка
    private static class FunctionNode {
        FunctionPoint point;     // точка функции
        FunctionNode next;       // ссылка на следующий узел
        FunctionNode prev;       // ссылка на предыдущий узел
        FunctionNode(FunctionPoint p) { point = p; }
    }

    private final FunctionNode head = new FunctionNode(null); // фиктивный
узел (голова)
    private int pointsCount;                                    // текущее
количество точек
    private static final double EPS = Math.ulp(1.0);           // машинный
эпсилон для сравнения double

    // поля для кэширования последнего использованного узла
    private FunctionNode lastAccessedNode = null;
    private int lastAccessedIndex = -1;
```

В классе LinkedListTabulatedFunction реализована внутренняя структура двусвязного циклического списка с фиктивным узлом головы. Для описания элемента списка используется вложенный приватный класс FunctionNode. Этот класс содержит поле point для хранения объекта FunctionPoint и ссылки next и prev на следующий и предыдущий узел соответственно. Конструктор класса инициализирует информационное поле point. Благодаря приватной вложенности элементы списка недоступны напрямую из вне, что обеспечивает инкапсуляцию.

Поле head представляет фиктивный узел списка, который не хранит реальную точку функции. В пустом списке
ссылки head.next и head.prev ссылаются на самого себя, что позволяет сохранить циклическую структуру. Поле pointsCount хранит текущее количество значащих узлов и используется для проверки корректности индексов при доступе, добавлении и удалении точек.

Для оптимизации доступа реализовано кэширование последнего обращения к узлу через поля lastAccessedNode и lastAccessedIndex. Если следующий вызов

метода требует доступа к элементу, близкому к последнему использованному, обход списка выполняется от кэшированного узла, а не от головы, что повышает эффективность работы с длинными списками.

```
public LinkedListTabulatedFunction(double leftX, double rightX, int count) {
    if (leftX >= rightX)
        throw new IllegalArgumentException("левый >= правый");
    if (count < 2)
        throw new IllegalArgumentException("кол-во точек < 2");

    head.next = head.prev = head;  // инициализация пустого кольцевого списка
    pointsCount = 0;
    double step = (rightX - leftX) / (count - 1); // шаг по X между точками
    for (int i = 0; i < count; i++)
        addNodeToTail().point = new FunctionPoint(leftX + i * step, 0); //
создаем точки с Y=0
}

// конструктор 2: по массиву Y-значений
public LinkedListTabulatedFunction(double leftX, double rightX, double[]
values) {
    if (leftX >= rightX)
        throw new IllegalArgumentException("левый >= правый");
    if (values.length < 2)
        throw new IllegalArgumentException("кол-во точек < 2");

    head.next = head.prev = head;  // инициализация пустого кольцевого списка
    pointsCount = 0;
    double step = (rightX - leftX) / (values.length - 1); // шаг по X между
точками
    for (int i = 0; i < values.length; i++)
        addNodeToTail().point = new FunctionPoint(leftX + i * step,
values[i]); // создаем точки с заданными Y
}
```

Здесь реализованы: Полностью аналогичные конструкторы ArrayTabulatedFunction. Проверка корректности данных (xValues возрастают, values.length ≥ 2). Создание связного списка через addNodeToTail(). Исключения те же (IllegalArgumentException).

```
public int getPointsCount() {
    return pointsCount;
}

public double getPointX(int index) {
    return getNodeByIndex(index).point.getX();
}

public double getPointY(int index) {
    return getNodeByIndex(index).point.getY();
}

public void setPointY(int index, double y) {
    getNodeByIndex(index).point.setY(y);
}
```

Эти методы полностью аналогичны методам из ArrayTabulatedFunction, но работают через связный список.

```
public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
    FunctionNode node = head.next;
    while (node != head) {
        if (Math.abs(node.point.getX() - point.getX()) < EPS)
            throw new InappropriateFunctionPointException("дубликат X"); //
проверка на дублирование
        node = node.next;
    }

    FunctionNode newNode = new FunctionNode(new FunctionPoint(point));
    if (head.next == head) { // если список пустой
        head.next = head.prev = newNode;
        newNode.next = newNode.prev = head;
    } else { // вставка в правильное место по X
        node = head.next;
        while (node != head && node.point.getX() < point.getX())
            node = node.next;
        newNode.prev = node.prev;
        newNode.next = node;
        node.prev.next = newNode;
        node.prev = newNode;
    }
    pointsCount++; // обновляем количество точек
}
```

Здесь используется вставка напрямую в нужное место списка (addNodeByIndex()), без лишнего создания массива или копирования данных — это оптимизация.

```
private FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException("индекс " + index +
" вне диапазона");

    // если есть кэш и расстояние до него небольшое, идем от кэша
    if (lastAccessedNode != null && Math.abs(index - lastAccessedIndex) <=
pointsCount / 2) {
        FunctionNode node = lastAccessedNode;
        if (index > lastAccessedIndex) {
            for (int i = lastAccessedIndex; i < index; i++)
                node = node.next;
        } else if (index < lastAccessedIndex) {
            for (int i = lastAccessedIndex; i > index; i--)
                node = node.prev;
        }
        lastAccessedNode = node;
        lastAccessedIndex = index;
        return node;
    }
// обычный обход от головы
    FunctionNode node = head.next;
    for (int i = 0; i < index; i++)
        node = node.next;

    lastAccessedNode = node; // обновляем кэш
    lastAccessedIndex = index;
    return node;
}
```

В классе LinkedListTabulatedFunction реализован служебный
метод getNodeByIndex(int index),
который возвращает ссылку на элемент списка по его индексу.
Метод проверяет корректность индекса и выбрасывает исключение
FunctionPointIndexOutOfBoundsException, если индекс выходит за
допустимые границы.

Для повышения производительности реализован
механизм оптимизированного доступа:
класс хранит ссылку на последний использованный элемент
(lastAccessedNode)
и его индекс (lastAccessedIndex).
Если при следующем вызове индекс находится недалеко от предыдущего,
обход списка начинается не с головы, а от последнего элемента.
Это существенно снижает количество итераций при последовательном
доступе к элементам
и повышает эффективность работы с длинными функциями.

Таким образом, метод getNodeByIndex() обеспечивает оптимизированный
доступ
к элементам связного списка и используется во всех методах класса,
где требуется получить или изменить точку функции по индексу

```
private FunctionNode deleteNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException("индекс" + index +
"вне допустимого диапазона");
    if (pointsCount <= 2)
        throw new IllegalStateException("удаление невозможно: кол-во точек
меньше 3"); // минимум 2 точки оставлять нельзя

    FunctionNode nodeToDelete = getNodeByIndex(index);

    // переподключение соседних узлов
    nodeToDelete.prev.next = nodeToDelete.next;
    nodeToDelete.next.prev = nodeToDelete.prev;

    if (nodeToDelete == head.next)
        head.next = nodeToDelete.next;

    pointsCount--;                    // обновляем количество точек

    // обновление кэша
    if (lastAccessedIndex == index) {
        lastAccessedNode = null;
        lastAccessedIndex = -1;
    } else if (lastAccessedIndex > index) {
        lastAccessedIndex--;
    }

    return nodeToDelete;
}
```

В классе LinkedListTabulatedFunction реализован метод deleteNodeByIndex(int index), удаляющий элемент связного списка по заданному индексу.
Метод использует внутренний метод getNodeByIndex() для получения ссылки на элемент,
проверяет корректность индекса и выбрасывает исключение FunctionPointIndexOutOfBoundsException при выходе за границы.

После получения узла для удаления его связи перестраиваются:
предыдущий элемент начинает ссылаться на следующий,
а следующий — на предыдущий, тем самым удаляемый узел исключается из списка.
Затем уменьшается счётчик точек, и если список становится пустым,
голова (head) ссылается сама на себя, сохраняя циклическую структуру.

Возвращаемое значение — ссылка на удалённый узел —
позволяет использовать метод для внутренних операций класса.

Таким образом, метод обеспечивает корректное и безопасное удаление элемента списка
с сохранением целостности двусвязной структуры.

```
private FunctionNode addNodeToTail() {
    FunctionNode newNode = new FunctionNode(new FunctionPoint(0, 0)); //
создаем новый узел

    if (head.next == head) { // если список пустой
        head.next = head.prev = newNode;
        newNode.next = newNode.prev = head;
    } else { // вставка в конец
        newNode.next = head;
        newNode.prev = head.prev;
        head.prev.next = newNode;
        head.prev = newNode;
    }

    pointsCount++;                  // обновляем количество точек
    lastAccessedNode = newNode;     // обновляем кэш
    lastAccessedIndex = pointsCount - 1;
    return newNode;
}
```

Метод addNodeToTail() добавляет новый узел в конец двусвязного циклического списка. Он создаёт новый объект FunctionNode, содержащий пустую точку функции (0, 0). Если список пустой, новый узел становится первым значащим элементом, а его ссылки next и prev указывают на голову. Если список не пустой, новый узел вставляется перед головой, при этом корректируются ссылки предыдущего последнего узла и головы. После вставки увеличивается количество точек pointsCount, обновляется кэш lastAccessedNode и lastAccessedIndex. Метод возвращает ссылку на добавленный узел.

```
private FunctionNode addNodeByIndex(int index) {
    if (index < 0 || index > pointsCount)
        throw new FunctionPointIndexOutOfBoundsException("индекс" + index +
"вне допустимого диапазона для вставки");

    if (index == pointsCount)           // добавление в конец
        return addNodeToTail();

    FunctionNode currentNode = getNodeByIndex(index);  // узел, перед которым
вставляем
    FunctionNode newNode = new FunctionNode(new FunctionPoint(0, 0));

    // вставка нового узла перед currentNode
    newNode.prev = currentNode.prev;
    newNode.next = currentNode;
    currentNode.prev.next = newNode;
    currentNode.prev = newNode;

    pointsCount++;                      // обновляем количество точек
    if (lastAccessedIndex >= index)     // корректируем кэш
        lastAccessedIndex++;
    lastAccessedNode = newNode;

    return newNode;
}
```

Метод addNodeByIndex(int index) вставляет новый узел в указанную позицию списка. Сначала проверяется корректность индекса, при необходимости выбрасывается FunctionPointIndexOutOfBoundsException. Если индекс равен количеству точек, вызов перенаправляется в addNodeToTail(). Иначе метод получает узел, перед которым будет вставка, с помощью getNodeByIndex(index). Новый узел создаётся с пустой точкой функции (0, 0) и вставляется перед выбранным узлом, при этом корректируются ссылки соседних узлов. После вставки увеличивается количество точек, при необходимости корректируется кэш, и возвращается ссылка на добавленный узел. Эти методы обеспечивают безопасное и корректное добавление точек в связный список для реализации табулированной функции.

```
public double getFunctionValue(double x) {
    if (x < getLeftDomainBorder() - EPS || x > getRightDomainBorder() + EPS)
        return Double.NaN; // x вне области определения

    FunctionNode node = head.next;
    while (node.next != head) { // проход по списку
        double x1 = node.point.getX();
        double x2 = node.next.point.getX();
        if (Math.abs(x - x1) < EPS) return node.point.getY(); // совпадение с
узлом
        if (x > x1 - EPS && x < x2 + EPS) // линейная интерполяция
            return node.point.getY() + (node.next.point.getY() -
node.point.getY()) * (x - x1) / (x2 - x1);
        node = node.next;
    }
    if (Math.abs(x - head.prev.point.getX()) < EPS)
        return head.prev.point.getY(); // совпадение с последним узлом
    return Double.NaN; // значение вне диапазона
}
```

Метод позволяет корректно вычислять значения функции для любого x в пределах области определения и обеспечивает линейную интерполяцию между узлами.

```
public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
    FunctionNode node = getNodeByIndex(index);
    double x = point.getX();
    if ((node.prev != head && x <= node.prev.point.getX()) ||
            (node.next != head && x >= node.next.point.getX()))
        throw new InappropriateFunctionPointException("X вне порядка"); //
проверка порядка X
    node.point = new FunctionPoint(point); // установка новой точки
}
```

Метод setPoint(int index, FunctionPoint point) заменяет точку функции в узле с указанным индексом. Сначала проверяется корректность индекса через getNodeByIndex. Затем проверяется порядок точек по x: новый хне должен быть меньше или равен предыдущему узлу и больше или равен следующему. При нарушении выбрасывается InappropriateFunctionPointException. Если проверка пройдена, узел получает копию новой точки, обеспечивая безопасное обновление и сохранение порядка.

```
public void deletePoint(int index) {
    deleteNodeByIndex(index); // удаление точки через приватный метод
}
```

Метод deletePoint(int index) удаляет точку по индексу через deleteNodeByIndex и выбрасывает IllegalStateException, если после удаления останется меньше трёх точек.

Таким образом, класс LinkedListTabulatedFunction полностью повторяет поведение ArrayTabulatedFunction, сохраняя совместимость интерфейсов и исключений, но реализует хранение данных в виде двусвязного циклического списка с возможностью более эффективного доступа к элементам.

Задание 6

```
package functions;

public interface TabulatedFunction {
    //возвращает количество точек в функции
    int getPointsCount();

    //возвращает левую границу области определения (минимальный x)
    double getLeftDomainBorder();

    //возвращает правую границу области определения (максимальный x)
    double getRightDomainBorder();
```

```
    //вычисляет значение функции в заданной точке x
    //возвращает double.nan, если x вне области определения
    double getFunctionValue(double x);

    //возвращает точку по указанному индексу
    //выбрасывает исключение, если индекс выходит за границы
    FunctionPoint getPoint(int index) throws
FunctionPointIndexOutOfBoundsException;

    //заменяет точку по указанному индексу
    //выбрасывает исключения при недопустимом индексе или нарушении порядка
точек
    void setPoint(int index, FunctionPoint point) throws
FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;

    //возвращает координату x точки по указанному индексу
    double getPointX(int index) throws
FunctionPointIndexOutOfBoundsException;

    //устанавливает координату x точки по указанному индексу
    //проверяет сохранение упорядоченности точек по x
    void setPointX(int index, double x) throws
FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;

    //возвращает координату y точки по указанному индексу
    double getPointY(int index) throws
FunctionPointIndexOutOfBoundsException;

    //устанавливает координату y точки по указанному индексу
    void setPointY(int index, double y) throws
FunctionPointIndexOutOfBoundsException;

    //добавляет новую точку в функцию
    //проверяет отсутствие дублирования координат x и сохраняет
упорядоченность
    void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException;

    //удаляет точку по указанному индексу
    //требует наличия минимум 2 точек после удаления
    void deletePoint(int index) throws
FunctionPointIndexOutOfBoundsException, IllegalStateException;
}
```

Интерфейс TabulatedFunction содержит объявления всех общих методов, используемых в
классах ArrayTabulatedFunction и LinkedListTabulatedFunction. Он определяет работу с табулированной функцией через следующие операции: получение количества точек, границ области определения, вычисление значения функции, доступ к точкам и их координатам, добавление и удаление точек, а также изменение точек с проверкой порядка по x.

Классы ArrayTabulatedFunction и LinkedListTabulatedFunction реализуют этот интерфейс, что обеспечивает единый тип для работы с любым объектом табулированной функции. При этом реализация конкретных методов различается: первый использует массив для хранения точек, второй — двусвязный циклический список. Такой подход инкапсулирует детали

хранения данных в классах и позволяет работать с функциями через интерфейс, не завися от конкретной реализации.

Задание 7

Создаем main, который будет отображать все наши труды

```
import functions.*;

public class Main {
    public static void main(String[] args) {
        System.out.println("ТЕСТИРОВАНИЕ КЛАССОВ TABULATED FUNCTION\n");

        // Проверка конструкторов на некорректные параметры
        System.out.println("Проверка конструкторов на неверные параметры:");
        try {
            TabulatedFunction invalid1 = new ArrayTabulatedFunction(5, 2, new
double[]{1, 4, 9});
        } catch (IllegalArgumentException e) {
            System.out.println("Ловим IllegalArgumentException
ArrayTabulatedFunction (left >= right): " + e.getMessage());
        }

        try {
            TabulatedFunction invalid2 = new ArrayTabulatedFunction(0, 2, new
double[]{1});
        } catch (IllegalArgumentException e) {
            System.out.println("Ловим IllegalArgumentException
ArrayTabulatedFunction (points < 2): " + e.getMessage());
        }

        try {
            TabulatedFunction invalid3 = new LinkedListTabulatedFunction(4,
1, new double[]{0, 1, 4});
        } catch (IllegalArgumentException e) {
            System.out.println("Ловим IllegalArgumentException
LinkedListTabulatedFunction (left >= right): " + e.getMessage());
        }

        try {
            TabulatedFunction invalid4 = new LinkedListTabulatedFunction(0,
1, new double[]{5});
        } catch (IllegalArgumentException e) {
            System.out.println("Ловим IllegalArgumentException
LinkedListTabulatedFunction (points < 2): " + e.getMessage());
        }

        // тестирование ArrayTabulatedFunction
        System.out.println("\nТест: ArrayTabulatedFunction");
        testFunction(new ArrayTabulatedFunction(0, 4, new double[]{0, 1, 4,
9, 16}));

        // тестирование LinkedListTabulatedFunction
        System.out.println("\nТест: LinkedListTabulatedFunction");
        testFunction(new LinkedListTabulatedFunction(0, 4, new double[]{0, 1,
4, 9, 16}));

        System.out.println("\nВСЕ ТЕСТЫ ЗАВЕРШЕНЫ");
    }

    private static void testFunction(TabulatedFunction func) {
```

```
        System.out.println("Тип функции: " +
func.getClass().getSimpleName());
        System.out.println();

        // 1. Проверка вычисления значений функции
        System.out.println("1. Проверяем значения функции:");
        for (double x = 1; x <= 5; x += 0.5) {
            double y = func.getFunctionValue(x);
            System.out.printf("f(%.1f) = %s\n", x, Double.isNaN(y) ? "NaN" :
y);
        }

        // 2. Проверка доступа к точкам
        System.out.println("\n2. Проверяем доступ к точкам:");
        try {
            System.out.println("Количество точек: " + func.getPointsCount());
            for (int i = 0; i < func.getPointsCount(); i++) {
                FunctionPoint p = func.getPoint(i);
                System.out.printf("Точка %d: (%.2f, %.2f)\n", i, p.getX(),
p.getY());
            }
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ошибка доступа к точке: " + e.getMessage());
        }

        // 3. Проверка ошибок индекса для всех методов
        System.out.println("\n3. Проверяем ошибки индекса:");
        try {
            func.getPoint(-1);
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ловим FunctionPointIndexOutOfBoundsException
getPoint(-1): " + e.getMessage());
        }

        try {
            func.setPoint(func.getPointsCount(), new FunctionPoint(0, 0));
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ловим FunctionPointIndexOutOfBoundsException
setPoint(index==count): " + e.getMessage());
        }

        try {
            func.getPointX(func.getPointsCount());
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ловим FunctionPointIndexOutOfBoundsException
getPointX: " + e.getMessage());
        }

        try {
            func.setPointX(-1, 0);
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ловим FunctionPointIndexOutOfBoundsException
setPointX: " + e.getMessage());
        }

        try {
            func.getPointY(func.getPointsCount());
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ловим FunctionPointIndexOutOfBoundsException
getPointY: " + e.getMessage());
        }

        try {
            func.setPointY(-1, 0);
```

```
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ловим FunctionPointIndexOutOfBoundsException
setPointY: " + e.getMessage());
        }

        try {
            func.deletePoint(func.getPointsCount());
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Ловим FunctionPointIndexOutOfBoundsException
deletePoint: " + e.getMessage());
        }

        // 4. Проверка нарушений порядка X
        System.out.println("\n4. Проверяем нарушение порядка X:");
        try {
            FunctionPoint bad = new FunctionPoint(100, 0);
            func.setPoint(1, bad);
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Ловим InappropriateFunctionPointException
setPoint: " + e.getMessage());
        }

        try {
            func.setPointX(1, func.getPointX(0) - 1);
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Ловим InappropriateFunctionPointException
setPointX: " + e.getMessage());
        }

        // 5. Проверка добавления точки с дублирующим X
        System.out.println("\n5. Проверяем добавление точки с повторяющимся
X:");
        try {
            double existingX = func.getPointX(2);
            func.addPoint(new FunctionPoint(existingX, 123));
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Ловим InappropriateFunctionPointException
(дубль X): " + e.getMessage());
        }

        // 6. Проверка удаления точки при <2 точках
        System.out.println("\n6. Проверяем удаление, если останется <2
точек:");
        try {
            TabulatedFunction small = new ArrayTabulatedFunction(0, 1, new
double[]{0, 1});
            small.deletePoint(0);
        } catch (IllegalStateException e) {
            System.out.println("Ловим IllegalStateException deletePoint: " +
e.getMessage());
        }

        // 7. Дополнительно проверяем методы интерфейса
        System.out.println("\n7. Проверяем getPointX/getPointY и
setPointY:");
        for (int i = 0; i < func.getPointsCount(); i++) {
            System.out.printf("Точка %d: x=%.2f, y=%.2f\n", i,
func.getPointX(i), func.getPointY(i));
        }

        System.out.println("\nИзменяем Y у точки 2 на 100");
        func.setPointY(2, 100);
        System.out.printf("Новая точка 2: (%.2f, %.2f)\n", func.getPointX(2),
func.getPointY(2));
```

```
        // 8. Успешное изменение X точки
        System.out.println("\nИзменяем X у точки 2 на 2.2");
        func.setPointX(2, 2.2);
        System.out.printf("Новая точка 2: (%.2f, %.2f)\n", func.getPointX(2),
func.getPointY(2));

        // 9. Успешное addPoint
        System.out.println("\nДобавляем новую точку (2.5, 50)");
        func.addPoint(new FunctionPoint(2.5, 50));
        for (int i = 0; i < func.getPointsCount(); i++) {
            FunctionPoint p = func.getPoint(i);
            System.out.printf("Точка %d: (%.2f, %.2f)\n", i, p.getX(),
p.getY());
        }

        // 10. Успешное deletePoint
        System.out.println("\nУдаляем точку с индексом 2");
        func.deletePoint(2);
        for (int i = 0; i < func.getPointsCount(); i++) {
            FunctionPoint p = func.getPoint(i);
            System.out.printf("Точка %d: (%.2f, %.2f)\n", i, p.getX(),
p.getY());
        }

        // 11. Успешное setPoint
        System.out.println("\nУстанавливаем точку 1 на (1.5, 75)");
        func.setPoint(1, new FunctionPoint(1.5, 75));
        for (int i = 0; i < func.getPointsCount(); i++) {
            FunctionPoint p = func.getPoint(i);
            System.out.printf("Точка %d: (%.2f, %.2f)\n", i, p.getX(),
p.getY());
        }

        System.out.println("\nПроверка " + func.getClass().getSimpleName() +
" завершена успешно!\n");
    }
}
```

Вот его вывод:

ТЕСТИРОВАНИЕ КЛАССОВ TABULATED FUNCTION

Проверка конструкторов на неверные параметры:

Ловим IllegalArgumentException ArrayTabulatedFunction (left >= right): левая граница >= правая граница

Ловим IllegalArgumentException ArrayTabulatedFunction (points < 2): Кол-во точек < 2

Ловим IllegalArgumentException LinkedListTabulatedFunction (left >= right): левый >= правый

Ловим IllegalArgumentException LinkedListTabulatedFunction (points < 2): кол-во точек < 2

Тест: ArrayTabulatedFunction

Тип функции: ArrayTabulatedFunction

1. Проверяем значения функции:

f(1,0) = 1.0

f(1,5) = 2.5

f(2,0) = 4.0

f(2,5) = 6.5

f(3,0) = 9.0

f(3,5) = 12.5

f(4,0) = 16.0

f(4,5) = NaN

f(5,0) = NaN

2. Проверяем доступ к точкам:

Количество точек: 5

Точка 0: (0,00, 0,00)

Точка 1: (1,00, 1,00)

Точка 2: (2,00, 4,00)

Точка 3: (3,00, 9,00)

Точка 4: (4,00, 16,00)

3. Проверяем ошибки индекса:

Ловим FunctionPointIndexOutOfBoundsException getPoint(-1): Индекс-1выходит за границы

Ловим FunctionPointIndexOutOfBoundsException setPoint(index==count): Индекс5выходит за границы

Ловим FunctionPointIndexOutOfBoundsException getPointX: Индекс 5 выходит за границы

Ловим FunctionPointIndexOutOfBoundsException setPointX: Индекс -1 выходит за границы

Ловим FunctionPointIndexOutOfBoundsException getPointY: Индекс 5 выходит за границы

Ловим FunctionPointIndexOutOfBoundsException setPointY: Индекс -1 выходит за границы

Ловим FunctionPointIndexOutOfBoundsException deletePoint: Индекс 5 выходит за границы

4. Проверяем нарушение порядка X:

Ловим InappropriateFunctionPointException setPoint: X вне порядка

Ловим InappropriateFunctionPointException setPointX: X вне порядка

5. Проверяем добавление точки с повторяющимся X:

Ловим InappropriateFunctionPointException (дубль X): дубликат X

6. Проверяем удаление, если останется <2 точек:

Ловим IllegalStateException deletePoint: удаление невозможно: кол-во точек < 3

7. Проверяем getPointX/getPointY и setPointY:

Точка 0: x=0,00, y=0,00

Точка 1: x=1,00, y=1,00

Точка 2: x=2,00, y=4,00

Точка 3: x=3,00, y=9,00

Точка 4: x=4,00, y=16,00

Изменяем Y у точки 2 на 100

Новая точка 2: (2,00, 100,00)

Изменяем X у точки 2 на 2.2

Новая точка 2: (2,20, 100,00)

Добавляем новую точку (2.5, 50)

Точка 0: (0,00, 0,00)

Точка 1: (1,00, 1,00)

Точка 2: (2,20, 100,00)

Точка 3: (2,50, 50,00)

Точка 4: (3,00, 9,00)

Точка 5: (4,00, 16,00)

Удаляем точку с индексом 2

Точка 0: (0,00, 0,00)

Точка 1: (1,00, 1,00)

Точка 2: (2,50, 50,00)

Точка 3: (3,00, 9,00)

Точка 4: (4,00, 16,00)

Устанавливаем точку 1 на (1.5, 75)

Точка 0: (0,00, 0,00)

Точка 1: (1,50, 75,00)

Точка 2: (2,50, 50,00)

Точка 3: (3,00, 9,00)

Точка 4: (4,00, 16,00)

Проверка ArrayTabulatedFunction завершена успешно!

Тест: LinkedListTabulatedFunction

Тип функции: LinkedListTabulatedFunction

1. Проверяем значения функции:

f(1,0) = 1.0

f(1,5) = 2.5

f(2,0) = 4.0

f(2,5) = 6.5

f(3,0) = 9.0

f(3,5) = 12.5

f(4,0) = 16.0

f(4,5) = NaN

f(5,0) = NaN

2. Проверяем доступ к точкам:

Количество точек: 5

Точка 0: (0,00, 0,00)

Точка 1: (1,00, 1,00)

Точка 2: (2,00, 4,00)

Точка 3: (3,00, 9,00)

Точка 4: (4,00, 16,00)

3. Проверяем ошибки индекса:

Ловим FunctionPointIndexOutOfBoundsException getPoint(-1): индекс -1 вне диапазона

Ловим FunctionPointIndexOutOfBoundsException setPoint(index==count): индекс 5 вне диапазона

Ловим FunctionPointIndexOutOfBoundsException getPointX: индекс 5 вне диапазона

Ловим FunctionPointIndexOutOfBoundsException setPointX: индекс -1 вне диапазона

Ловим FunctionPointIndexOutOfBoundsException getPointY: индекс 5 вне диапазона

Ловим FunctionPointIndexOutOfBoundsException setPointY: индекс -1 вне диапазона

Ловим FunctionPointIndexOutOfBoundsException deletePoint: индекс5вне допустимого диапазона

4. Проверяем нарушение порядка X:

Ловим InappropriateFunctionPointException setPoint: X вне порядка

Ловим InappropriateFunctionPointException setPointX: X вне порядка

5. Проверяем добавление точки с повторяющимся X:

Ловим InappropriateFunctionPointException (дубль X): дубликат X

6. Проверяем удаление, если останется <2 точек:

Ловим IllegalStateException deletePoint: удаление невозможно: кол-во точек < 3

7. Проверяем getPointX/getPointY и setPointY:

Точка 0: x=0,00, y=0,00

Точка 1: x=1,00, y=1,00

Точка 2: x=2,00, y=4,00

Точка 3: x=3,00, y=9,00

Точка 4: x=4,00, y=16,00

Изменяем Y у точки 2 на 100

Новая точка 2: (2,00, 100,00)

Изменяем X у точки 2 на 2.2

Новая точка 2: (2,20, 100,00)

Добавляем новую точку (2.5, 50)

Точка 0: (0,00, 0,00)

Точка 1: (1,00, 1,00)

Точка 2: (2,20, 100,00)

Точка 3: (2,50, 50,00)

Точка 4: (3,00, 9,00)

Точка 5: (4,00, 16,00)

Удаляем точку с индексом 2

Точка 0: (0,00, 0,00)

Точка 1: (1,00, 1,00)

Точка 2: (2,50, 50,00)

Точка 3: (3,00, 9,00)

Точка 4: (4,00, 16,00)

Устанавливаем точку 1 на (1.5, 75)

Точка 0: (0,00, 0,00)

Точка 1: (1,50, 75,00)

Точка 2: (2,50, 50,00)

Точка 3: (3,00, 9,00)

Точка 4: (4,00, 16,00)

Проверка LinkedListTabulatedFunction завершена успешно!

ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ

Process finished with exit code 0

Спасибо за внимание!