Лабораторная работа по ООП №3

Студент: Тарасова Полина

Группа: 6204-010302D

Задание 1

Я изучила следующие классы исключений, входящиие в API Java: java.lang.Exception, java.lang.IndexOutOfBoundsException, java.lang.ArrayIndexOutOfBoundsException, java.lang.IllegalArgumentException, java.lang.IllegalStateException.

Задание 2

В пакете functions я создала два класса исключений:

- FunctionPointIndexOutOfBoundsException – исключение выхода за границы набора точек при обращении к ним по номеру, наследует от класса IndexOutOfBoundsException.

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends
IndexOutOfBoundsException {
    public FunctionPointIndexOutOfBoundsException() {
        super();
    }

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

- InappropriateFunctionPointException – исключение, выбрасываемое при попытке добавления или изменения точки функции несоответствующим образом, наследует от класса Exception.

```
package functions;

public class InappropriateFunctionPointException extends Exception {
    public InappropriateFunctionPointException() {
        super();
    }

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

Задание 3

В разработанный ранее класс TabulatedFunction я внесла изменения, обеспечивающие выбрасывание исключений методами класса.

- Оба конструктора класса выбрасывают исключение IllegalArgumentException, если левая граница области определения больше или равна правой, а также если предлагаемое количество точек меньше двух. Это обеспечит создание объекта только при корректных параметрах.
- 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.
- Методы setPoint() и setPointX() выбрасывают 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 обеспечит сохранение упорядоченности точек функции.
- Метод deletePoint() выбрасывает исключение IllegalStateException, если на момент удаления точки количество точек в наборе менее трех. Это обеспечит невозможность получения функции с некорректным количеством точек.

```
package functions;

public class TabulatedFunction {
    private FunctionPoint[] points;
    private int pointsCount;

    public TabulatedFunction(double leftX, double rightX, int
pointsCount) {
        if (leftX >= rightX)
            throw new IllegalArgumentException("Левая граница >=
правой");
        if (pointsCount < 2)
            throw new IllegalArgumentException("Количество точек < 2");
```

```
        this.pointsCount = pointsCount;
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(leftX + i * step, 0);
        }
    }

    public TabulatedFunction(double leftX, double rightX, double[]
values) {
        if (leftX >= rightX)
            throw new IllegalArgumentException("Левая граница >=
правой");
        if (values.length < 2)
            throw new IllegalArgumentException("Количество точек < 2");

        pointsCount = values.length;
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(leftX + i * step, values[i]);
        }
    }

    public int getPointsCount() {
        return pointsCount;
    }

    public double getLeftDomainBorder() {
        return points[0].getX();
    }

    public double getRightDomainBorder() {
        return points[pointsCount - 1].getX();
    }

    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder())
return Double.NaN;

        for (int i = 0; i < pointsCount - 1; i++) {
            double x1 = points[i].getX();
            double x2 = points[i + 1].getX();
            if (x >= x1 && x <= x2) {
                double y1 = points[i].getY();
                double y2 = points[i + 1].getY();
                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
            }
        }
        return Double.NaN;
    }

    public FunctionPoint getPoint(int index) {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс: " + index);
        return new FunctionPoint(points[index]);
    }

    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
```

```
        if ((index > 0 && point.getX() <= points[index - 1].getX())
                || (index < pointsCount - 1 && point.getX() >=
points[index + 1].getX()))
            throw new InappropriateFunctionPointException("x нарушает
порядок точек");

        points[index] = new FunctionPoint(point);
    }

    public double getPointX(int index) {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        return points[index].getX();
    }

    public void setPointX(int index, double x) throws
InappropriateFunctionPointException {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");

        if ((index > 0 && x <= points[index - 1].getX())
                || (index < pointsCount - 1 && x >= points[index +
1].getX()))
            throw new InappropriateFunctionPointException("x нарушает
порядок точек");

        points[index].setX(x);
    }

    public double getPointY(int index) {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        return points[index].getY();
    }

    public void setPointY(int index, double y) {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        points[index].setY(y);
    }

    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
        for (int i = 0; i < pointsCount; i++) {
            if (points[i].getX() == point.getX())
                throw new InappropriateFunctionPointException("Такая
точка уже есть");
        }

        FunctionPoint[] newPoints = new FunctionPoint[pointsCount + 1];
        int i = 0;
        while (i < pointsCount && points[i].getX() < point.getX()) {
            newPoints[i] = points[i];
            i++;
        }
        newPoints[i] = new FunctionPoint(point);
        System.arraycopy(points, i, newPoints, i + 1, pointsCount - i);
        points = newPoints;
        pointsCount++;
```

```
    }

    public void deletePoint(int index) {
        if (pointsCount <= 2)
            throw new IllegalStateException("Нельзя удалить — останется
меньше двух точек");
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");

        FunctionPoint[] newArr = new FunctionPoint[pointsCount - 1];
        for (int i = 0, j = 0; i < pointsCount; i++) {
            if (i != index) newArr[j++] = points[i];
        }
        points = newArr;
        pointsCount--;
    }
}
```

Задание 4

В пакете functions я создала класс LinkedListTabulatedFunction, объект которого также должен описывать табулированную функцию. Отличие этого класса в том, что для хранения набора точек в нем используется не массив, а динамическая структура – связный список.
Класс LinkedListTabulatedFunction совмещает в себе две функции: с одной стороны, он описывает связный список и работу с ним, а с другой стороны, он описывает работу с табулированной функцией и ее точками. Для реализации первой функции:

1. Я описала класс элементов списка FunctionNode, содержащий информационное поле для хранения данных типа FunctionPoint, а также поля для хранения ссылок на предыдущий и следующий элемент. Он является вложенным и приватным внутри класса LinkedListTabulatedFunction, чтобы скрыть детали реализации списка от внешнего кода. Все операции со списком выполняются через методы внешнего класса. Это обеспечивает инкапсуляцию данных и защищает структуру списка от некорректного изменения.
2. Я описала класс LinkedListTabulatedFunction объектов списка, содержащий поле ссылки на объект головы, а также иные вспомогательные поля.

3. В классе LinkedListTabulatedFunction я реализовала метод FunctionNode getNodeByIndex(int index), возвращающий ссылку на объект элемента списка по его номеру.
4. В классе LinkedListTabulatedFunction я реализовала метод FunctionNode addNodeToTail(), добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента.
5. В классе LinkedListTabulatedFunction я реализовала метод FunctionNode addNodeByIndex(int index), добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента.
6. В классе LinkedListTabulatedFunction я реализовала метод FunctionNode deleteNodeByIndex(int index), удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента.

```
package functions;

// Табулированная функция на двусвязном списке
public class LinkedListTabulatedFunction implements TabulatedFunction {
    private class FunctionNode {
        FunctionPoint point;
        FunctionNode next;
        FunctionNode prev;

        FunctionNode(FunctionPoint p) {
            this.point = new FunctionPoint(p);
        }
    }

    private FunctionNode head; // фиктивная голова списка
    private int pointsCount;
```

и

```
private void addNodeToTail(FunctionPoint p) {
    FunctionNode node = new FunctionNode(p);
    if (pointsCount == 0) {
        head.next = node;
        head.prev = node;
        node.next = head;
        node.prev = head;
    } else {
        FunctionNode last = head.prev;
        last.next = node;
        node.prev = last;
        node.next = head;
        head.prev = node;
    }
    pointsCount++;
}
```

```
private FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException("Неверный индекс: "
+ index);
    FunctionNode node = head.next;
    for (int i = 0; i < index; i++) node = node.next;
    return node;
}
```

Задание 5

Для обеспечения второй функции класса LinkedListTabulatedFunction я реализовала в классе конструкторы и методы, аналогичные конструкторам и методам класса TabulatedFunction. Конструкторы имеют те же параметры, методы имеют те же сигнатуры. Также выбрасываются те же виды исключений в тех же случаях.

```
public LinkedListTabulatedFunction(double leftX, double rightX, double[]
values) {
    if (leftX >= rightX)
        throw new IllegalArgumentException("Левая граница >= правой");
    if (values.length < 2)
        throw new IllegalArgumentException("Количество точек < 2");

    head = new FunctionNode(new FunctionPoint());
    head.next = head;
    head.prev = head;

    double step = (rightX - leftX) / (values.length - 1);
    for (int i = 0; i < values.length; i++) {
        addNodeToTail(new FunctionPoint(leftX + i * step, values[i]));
    }
}
```

и

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

@Override
public FunctionPoint getPoint(int index) {
    return new FunctionPoint(getNodeByIndex(index).point);
}

@Override
public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
    if (index < 0 || index >= pointsCount)
        throw new FunctionPointIndexOutOfBoundsException();
    if ((index > 0 && point.getX() <= getPointX(index - 1)) ||
            (index < pointsCount - 1 && point.getX() >= getPointX(index +
1)))
```

```
        throw new InappropriateFunctionPointException("x нарушает порядок
точек");

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

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

@Override
public void setPointX(int index, double x) throws
InappropriateFunctionPointException {
    setPoint(index, new FunctionPoint(x, getPointY(index)));
}

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

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

@Override
public void deletePoint(int index) {
    if (pointsCount <= 2)
        throw new IllegalStateException("Нельзя удалить — останется меньше
двух точек");

    FunctionNode node = getNodeByIndex(index);
    node.prev.next = node.next;
    node.next.prev = node.prev;
    pointsCount--;
}

@Override
public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
    FunctionNode cur = head.next;
    while (cur != head) {
        if (cur.point.getX() == point.getX())
            throw new InappropriateFunctionPointException("Такая точка уже
есть");
        if (cur.point.getX() > point.getX())
            break;
        cur = cur.next;
    }
    FunctionNode node = new FunctionNode(point);
    node.next = cur;
    node.prev = cur.prev;
    cur.prev.next = node;
    cur.prev = node;
    pointsCount++;
}

@Override
public double getLeftDomainBorder() {
    return head.next.point.getX();
}
```

```
@Override
public double getRightDomainBorder() {
    return head.prev.point.getX();
}

@Override
public double getFunctionValue(double x) {
    if (x < getLeftDomainBorder() || x > getRightDomainBorder())
        return Double.NaN;

    FunctionNode cur = head.next;
    while (cur.next != head) {
        double x1 = cur.point.getX();
        double x2 = cur.next.point.getX();
        if (x >= x1 && x <= x2) {
            double y1 = cur.point.getY();
            double y2 = cur.next.point.getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
        cur = cur.next;
    }
    return Double.NaN;
}
```

Задание 6

Я переименовала класс TabulatedFunction в класс ArrayTabulatedFunction. Создала интерфейс TabulatedFunction, содержащий объявления общих методов классов ArrayTabulatedFunction и LinkedListTabulatedFunction.
Затем я сделала так, чтобы оба класса функций реализовывали созданный интерфейс. Теперь суть работы с табулированными функциями заключена в типе интерфейса, а в классах заключена только реализация этой работы.

TabulatedFunction:

```
package functions;

public interface TabulatedFunction {
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException;
    double getPointX(int index);
    void setPointX(int index, double x) throws
InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException;
    double getLeftDomainBorder();
    double getRightDomainBorder();
```

```
    double getFunctionValue(double x);
}
```

ArrayTabulatedFunction:

```
package functions;

public class ArrayTabulatedFunction implements TabulatedFunction {
    private FunctionPoint[] points;
    private int pointsCount;

    // конструктор, если нужно создать функцию с нулями
    public ArrayTabulatedFunction(double leftX, double rightX, int
pointsCount) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница >= правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек < 2");
        }

        this.pointsCount = pointsCount;
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(leftX + i * step, 0);
        }
    }

    // конструктор с уже готовыми значениями y
    public ArrayTabulatedFunction(double leftX, double rightX, double[]
values) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница >= правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек < 2");
        }

        pointsCount = values.length;
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(leftX + i * step, values[i]);
        }
    }

    @Override
    public int getPointsCount() {
        return pointsCount;
    }

    @Override
    public double getLeftDomainBorder() {
        return points[0].getX();
    }

    @Override
    public double getRightDomainBorder() {
        return points[pointsCount - 1].getX();
    }
```

```
    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }

        for (int i = 0; i < pointsCount - 1; i++) {
            double x1 = points[i].getX();
            double x2 = points[i + 1].getX();
            if (x >= x1 && x <= x2) {
                double y1 = points[i].getY();
                double y2 = points[i + 1].getY();
                double k = (y2 - y1) / (x2 - x1);
                return y1 + k * (x - x1);
            }
        }

        return Double.NaN;
    }

    @Override
    public FunctionPoint getPoint(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс: " + index);
        }
        return new FunctionPoint(points[index]);
    }

    @Override
    public void setPoint(int index, FunctionPoint point)
            throws FunctionPointIndexOutOfBoundsException,
InappropriateFunctionPointException {

        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        }

        // проверяем, чтобы x не нарушил порядок
        if ((index > 0 && point.getX() <= points[index - 1].getX()) ||
                (index < pointsCount - 1 && point.getX() >= points[index +
1].getX())) {
            throw new InappropriateFunctionPointException("x нарушает порядок
точек");
        }

        points[index] = new FunctionPoint(point);
    }

    @Override
    public double getPointX(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        }
        return points[index].getX();
    }

    @Override
    public void setPointX(int index, double x)
            throws FunctionPointIndexOutOfBoundsException,
InappropriateFunctionPointException {
```

```
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        }

        if ((index > 0 && x <= points[index - 1].getX()) ||
                (index < pointsCount - 1 && x >= points[index + 1].getX())) {
            throw new InappropriateFunctionPointException("x нарушает порядок
точек");
        }

        points[index].setX(x);
    }

    @Override
    public double getPointY(int index) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        }
        return points[index].getY();
    }

    @Override
    public void setPointY(int index, double y) {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        }
        points[index].setY(y);
    }

    @Override
    public void deletePoint(int index) {
        if (pointsCount <= 2) {
            throw new IllegalStateException("Нельзя удалить — останется
меньше двух точек");
        }

        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс");
        }

        FunctionPoint[] newArr = new FunctionPoint[pointsCount - 1];
        for (int i = 0, j = 0; i < pointsCount; i++) {
            if (i != index) {
                newArr[j++] = points[i];
            }
        }
        points = newArr;
        pointsCount--;
    }

    @Override
    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
        // проверка на совпадение x
        for (int i = 0; i < pointsCount; i++) {
            if (points[i].getX() == point.getX()) {
                throw new InappropriateFunctionPointException("Такая точка
уже есть");
            }
```

```
        }

        // создаем новый массив на 1 больше
        FunctionPoint[] newArr = new FunctionPoint[pointsCount + 1];
        int i = 0;
        while (i < pointsCount && points[i].getX() < point.getX()) {
            newArr[i] = points[i];
            i++;
        }

        newArr[i] = new FunctionPoint(point);

        for (int j = i; j < pointsCount; j++) {
            newArr[j + 1] = points[j];
        }

        points = newArr;
        pointsCount++;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("ArrayTabulatedFunction:\n");
        for (int i = 0; i < pointsCount; i++) {
            sb.append(i).append(": (")
                    .append(points[i].getX()).append(", ")
                    .append(points[i].getY()).append(")\n");
        }
        return sb.toString();
    }
}
```

LinkedListTabulatedFunction:

```
package functions;

// Табулированная функция на двусвязном списке
public class LinkedListTabulatedFunction implements TabulatedFunction {
    private class FunctionNode {
        FunctionPoint point;
        FunctionNode next;
        FunctionNode prev;

        FunctionNode(FunctionPoint p) {
            this.point = new FunctionPoint(p);
        }
    }

    private FunctionNode head; // фиктивная голова списка
    private int pointsCount;

    public LinkedListTabulatedFunction(double leftX, double rightX, double[]
values) {
        if (leftX >= rightX)
            throw new IllegalArgumentException("Левая граница >= правой");
        if (values.length < 2)
            throw new IllegalArgumentException("Количество точек < 2");

        head = new FunctionNode(new FunctionPoint());
        head.next = head;
        head.prev = head;

        double step = (rightX - leftX) / (values.length - 1);
```

```
        for (int i = 0; i < values.length; i++) {
            addNodeToTail(new FunctionPoint(leftX + i * step, values[i]));
        }
    }

    private void addNodeToTail(FunctionPoint p) {
        FunctionNode node = new FunctionNode(p);
        if (pointsCount == 0) {
            head.next = node;
            head.prev = node;
            node.next = head;
            node.prev = head;
        } else {
            FunctionNode last = head.prev;
            last.next = node;
            node.prev = last;
            node.next = head;
            head.prev = node;
        }
        pointsCount++;
    }

    private FunctionNode getNodeByIndex(int index) {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException("Неверный
индекс: " + index);
        FunctionNode node = head.next;
        for (int i = 0; i < index; i++) node = node.next;
        return node;
    }

    @Override
    public int getPointsCount() {
        return pointsCount;
    }

    @Override
    public FunctionPoint getPoint(int index) {
        return new FunctionPoint(getNodeByIndex(index).point);
    }

    @Override
    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
        if (index < 0 || index >= pointsCount)
            throw new FunctionPointIndexOutOfBoundsException();
        if ((index > 0 && point.getX() <= getPointX(index - 1)) ||
                (index < pointsCount - 1 && point.getX() >= getPointX(index +
1)))
            throw new InappropriateFunctionPointException("x нарушает порядок
точек");

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

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

    @Override
    public void setPointX(int index, double x) throws
InappropriateFunctionPointException {
        setPoint(index, new FunctionPoint(x, getPointY(index)));
```

```
    }

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

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

    @Override
    public void deletePoint(int index) {
        if (pointsCount <= 2)
            throw new IllegalStateException("Нельзя удалить — останется
меньше двух точек");

        FunctionNode node = getNodeByIndex(index);
        node.prev.next = node.next;
        node.next.prev = node.prev;
        pointsCount--;
    }

    @Override
    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException {
        FunctionNode cur = head.next;
        while (cur != head) {
            if (cur.point.getX() == point.getX())
                throw new InappropriateFunctionPointException("Такая точка
уже есть");
            if (cur.point.getX() > point.getX())
                break;
            cur = cur.next;
        }
        FunctionNode node = new FunctionNode(point);
        node.next = cur;
        node.prev = cur.prev;
        cur.prev.next = node;
        cur.prev = node;
        pointsCount++;
    }

    @Override
    public double getLeftDomainBorder() {
        return head.next.point.getX();
    }

    @Override
    public double getRightDomainBorder() {
        return head.prev.point.getX();
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder())
            return Double.NaN;

        FunctionNode cur = head.next;
        while (cur.next != head) {
            double x1 = cur.point.getX();
            double x2 = cur.next.point.getX();
            if (x >= x1 && x <= x2) {
```

```
                double y1 = cur.point.getY();
                double y2 = cur.next.point.getY();
                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
            }
            cur = cur.next;
        }
        return Double.NaN;
    }
}
```

Задание 7

В созданном ранее классе Main, содержащем точку входа программы, я добавила проверку для случаев, в которых объект табулированной функции выбрасывает исключения. Ссылочную переменную для работы с объектом функции я объявила типа TabulatedFunction, а при создании объекта указала реальный класс.

Main:

```
import functions.*;

public class Main {
    public static void main(String[] args) {
        double[] vals = {0, 1, 4, 9, 16};

        // можно переключать реализацию: Array или LinkedList
        TabulatedFunction func = new ArrayTabulatedFunction(0, 4, vals);
        // TabulatedFunction func = new LinkedListTabulatedFunction(0, 4,
vals);

        System.out.println("Точки исходной функции:");
        print(func);

        System.out.println("f(2.5) = " + func.getFunctionValue(2.5));
        System.out.println();

        // пример: попытка установить x, которая нарушает порядок
        try {
            func.setPointX(1, 10); // у метода setPointX может быть throws
InappropriateFunctionPointException
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Поймано исключение при setPointX: " +
e.getMessage());
        }

        // добавление новой точки — этот вызов тоже надо обрабатывать
        try {
            func.addPoint(new FunctionPoint(2.2, 5));
            System.out.println("После добавления (2.2,5):");
            print(func);
        } catch (InappropriateFunctionPointException e) {
            System.out.println("Не удалось добавить точку: " +
e.getMessage());
        }
```

```
        // удаление точек — deletePoint может бросить IllegalStateException
(runtime),
        // для индекса возможен FunctionPointIndexOutOfBoundsException
(runtime subclass)
        try {
            func.deletePoint(0);
            func.deletePoint(0);
            System.out.println("После двух удалений:");
            print(func);
        } catch (IllegalStateException e) {
            System.out.println("Ошибка состояния при удалении: " +
e.getMessage());
        } catch (FunctionPointIndexOutOfBoundsException e) {
            System.out.println("Неправильный индекс при удалении: " +
e.getMessage());
        }
    }

    private static void print(TabulatedFunction f) {
        for (int i = 0; i < f.getPointsCount(); i++) {
            System.out.println(i + ": " + f.getPoint(i));
        }
        System.out.println();
    }
}
```

Пример работы программы:

```
Точки исходной функции:
0: (0.0; 0.0)
1: (1.0; 1.0)
2: (2.0; 4.0)
3: (3.0; 9.0)
4: (4.0; 16.0)

f(2.5) = 6.5

Поймано исключение при setPointX: новый x нарушает порядок
После добавления (2.2,5):
0: (0.0; 0.0)
1: (1.0; 1.0)
2: (2.0; 4.0)
3: (2.2; 5.0)
4: (3.0; 9.0)
5: (4.0; 16.0)

После двух удалений:
0: (2.0; 4.0)
1: (2.2; 5.0)
2: (3.0; 9.0)
3: (4.0; 16.0)
```