

## **Лабораторная работа №3**

Выполнил: Магера Никита Алексеевич

Студент группы 6203-010302D

## Ход выполнения

### Задание 1

В рамках первого задания требуется ознакомиться (изучить документацию) со следующими классами исключений, входящих в API Java:

- `java.lang.Exception`
- `java.lang.IndexOutOfBoundsException`
- `java.lang.ArrayIndexOutOfBoundsException`
- `java.lang.IllegalArgumentException`
- `java.lang.IllegalStateException` (рис.1)

#### **java.lang.Exception**

Базовый класс для всех проверяемых исключений (checked exceptions), которые программа должна обрабатывать или объявлять.

#### **java.lang.IndexOutOfBoundsException**

Указывает, что индекс (например, для массива, списка или строки) вышел за допустимые границы (слишком велик или отрицателен).

#### **java.lang.ArrayIndexOutOfBoundsException**

Конкретный случай `IndexOutOfBoundsException` для массивов — индекс выходит за пределы длины массива.

#### **java.lang.IllegalArgumentException**

Сигнализирует, что методу передан некорректный или недопустимый аргумент.

#### **java.lang.IllegalStateException**

Указывает, что метод вызван в недопустимый момент или состояние объекта не позволяет выполнить операцию.

Рис. 1

### Задание 2

Во втором задании создаем два класса исключений:

- 1) `FunctionPointIndexOutOfBoundsException` – исключение выхода за границы набора точек при обращении к ним по номеру, наследует от класса `IndexOutOfBoundsException`;
- 2) `InappropriateFunctionPointException` – исключение, выбрасываемое при попытке добавления или изменения точки функции несуществующим образом, наследует от класса `Exception`. (Рис.2-3)

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException { no usages new *
    }

    public FunctionPointIndexOutOfBoundsException() { no usages new * }
}

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

Рис.2

```
package functions;

public class InappropriateFunctionPointException extends Exception { no usages new *
    ⚡ Rename usages
    public InappropriateFunctionPointException() { new * }
}

    public InappropriateFunctionPointException(String message) { no usages new * }
        super(message);
    }
}
```

Рис.3

### Задание 3

В третьем задании в пакете в разработанном ранее классе TabulatedFunction внесем изменения, обеспечивающие выбрасывание исключений методами класса.

Какие изменения надо внести:

- 1) Конструкторы класса TabulatedFunction должны выбрасывать 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.(Рис.4)

2)Методы `getPoint()`, `setPoint()`, `getPointX()`, `setPointX()`, `getPointY()`, `setPointY()` и `deletePoint()` должны выбрасывать исключение `FunctionPointIndexOutOfBoundsException` - если переданный в метод номер выходит за границы набора точек. Это обеспечит корректность обращений к точкам функции.(Рис.5)

3)Методы `setPoint()` и `setPointX()` должны выбрасывать исключение `InappropriateFunctionPointException` в том случае, если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции.(Рис.6)

4)Метод `addPoint()` также должен выбрасывать исключение `InappropriateFunctionPointException`, если в наборе точек функции есть точка, абсцисса которой совпадает с абсциссой добавляемой точки. Это обеспечит сохранение упорядоченности точек функции.(Рис.7)

5)Метод `deletePoint()` должен выбрасывать исключение `IllegalStateException`, если на момент удаления точки количество точек в наборе менее трех. Это обеспечит невозможность получения функции с некорректным количеством точек.(Рис.8)

```
public TabulatedFunction(double leftX, double rightX, int pointsCount) { no usages new *
    // добавляем исключения по 3 заданию
    if (leftX>= rightX)
        throw new IllegalArgumentException(
            "Левая граница является правой в данном случае."
        );
    if (pointsCount<2)
        throw new IllegalArgumentException(
            "Количество точек должно быть >= 2."
        );
    this.points = new FunctionPoint[pointsCount]; // создаем объект для массива точек
    this.pointsCount = pointsCount; // инициализируем количество точек
    double step = (rightX - leftX) / (pointsCount - 1); // считаем шаг между точками
    for (int i = 0; i < pointsCount; i++) {
        double x_c = leftX + i * step;// считаем координату x
        points[i] = new FunctionPoint(x_c, y: 0); // записываем наши точки в объект вида массива
    }
}
// второй конструктор по заданию
public TabulatedFunction(double leftX, double rightX, double[] values) { 1 usage new *
    // аналогичные проверки исключений
    if (leftX >= rightX) {
        throw new IllegalArgumentException(
            "Левая граница является правой в данном случае."
        );
    }
    if (values == null) {
        throw new IllegalArgumentException("Массив не может быть равен null");
    }
    if (values.length < 2) {
        throw new IllegalArgumentException(
            "Количество точек должно быть >= 2"
        );
    }
    this.points = new FunctionPoint[values.length]; // создаем объект для массива точек
    this.pointsCount = values.length; // инициализируем количество точек
    double step = (rightX - leftX) / (values.length - 1); // считаем шаг между точками
    for (int i = 0; i < values.length; i++) {
        double x_c = leftX + i * step; // считаем координату x
        points[i] = new FunctionPoint(x_c, values[i]); // записываем наши точки в объект вида массива
    }
}
```

Рис.4

```

// метод для вброса исключения FunctionPointIndexOutOfBoundsException(идея была взята с сайта metanit.com)
private void checkIndex(int index) { 7 usages new *
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException(
            "Значение" + index + " находится вне заданного диапазона [0, " + (pointsCount - 1) + "]"
        );
    }
}

// метод для получения копии точки
public FunctionPoint getPoint(int index) { no usages new *
    checkIndex(index);
    return new FunctionPoint(points[index]);
}

// измененный методы с исключениями
// метод для получения значения абсциссы
public double getPointX(int index) { 1 usage new *
    checkIndex(index);
    return points[index].getX();
}

// метод для получения значения ординаты
public double getPointY(int index) { 1 usage new *
    checkIndex(index);
    return points[index].getY();
}

// метод для изменения значения ординаты точки с указанным номером
public void setPointY(int index, double y) { 1 usage new *
    checkIndex(index);
    points[index].setY(y);
}

```

Рис.5

```

// метод для замены точки на табулированную
public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException { no usages new *
    checkIndex(index);
    // проверка на null
    if (point == null) {
        throw new IllegalArgumentException("Точка не может быть null");
    }
    // проверяем, что новый x между соседними точками
    if (index > 0 && point.getX() <= points[index - 1].getX()) {
        throw new InappropriateFunctionPointException(
            "значение x " + " должен быть > " + points[index - 1].getX()
        );
    }
    if (index < pointsCount - 1 && point.getX() >= points[index + 1].getX()) {
        throw new InappropriateFunctionPointException( "значение x " + " должен быть < " + points[index - 1].getX()
    }

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

// метод для замены абсциссы точки с указанным номером | с добавленными исключениями
public void setPointX(int index, double x) throws InappropriateFunctionPointException { 1 usage new * 1 related problem
    checkIndex(index);

    // проверка, что x между соседними точками
    if (index > 0 && x <= points[index - 1].getX()) {
        throw new InappropriateFunctionPointException(
            "значение x " + " должен быть > " + points[index - 1].getX()
        );
    }
    if (index < pointsCount - 1 && x >= points[index + 1].getX()) {
        throw new InappropriateFunctionPointException(
            "значение x " + " должен быть < " + points[index - 1].getX()
        );
    }

    points[index].setX(x);
}

```

Рис.6

```

    }

    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException { 1 usage new * 1 related problem
        // проверка на null
        if (point == null) {
            throw new IllegalArgumentException("Точка не может быть null");
        }

        // проверяем, что такой x еще не существует
        for (int i = 0; i < pointsCount; i++) {
            if (Math.abs(points[i].getX() - point.getX()) < 1e-10) {
                throw new InappropriateFunctionPointException(
                    "Точка с X=" + point.getX() + " уже существует"
                );
            }
        }

        // проверяем, заполнен ли массив
        if (pointsCount >= points.length) {
            // увеличиваем массив в 2 раза
            FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
            System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, pointsCount);
            points = newPoints;
        }

        // ищем позицию для вставки x
        int pos = 0;
        while (pos < pointsCount && points[pos].getX() < point.getX()) {
            pos++;
        }

        // сдвигаем точки вправо, чтобы освободить место для новой точки
        System.arraycopy(points, pos, points, destPos: pos + 1, length: pointsCount - pos);
        // вставляем новую точку
        points[pos] = new FunctionPoint(point);
        pointsCount++;
    }
}

```

Рис.7

```

public void deletePoint(int index) { 1 usage new *
    checkIndex(index);

    // проверка, что останется минимум 2 точки
    if (pointsCount <= 2) {
        throw new IllegalStateException( "Ошибка! Должно оставаться минимум 2 точки" );
    }

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

```

Рис.8

#### Задание 4

В 4 задании ознакомился с понятием двусвязный циклический список с выделенной головой. Нужно было создать класс LinkedListTabulatedFunction, который совмещает в себе две функции: первая, будет описывать связный список и работу с ним, а вторая, будет описывать работу с табулированной функцией и ее точками.

Для реализации данного класса следует сделать:

- 1) Описать класс элементов списка FunctionNode, содержащий информационное поле для хранения данных типа FunctionPoint, а также поля для хранения ссылок на предыдущий и следующий элемент.
- 2) Описать класс LinkedListTabulatedFunction объектов списка, содержащий поле ссылки на объект головы, а также иные вспомогательные поля.
- 3) В классе LinkedListTabulatedFunction реализовать метод FunctionNode getNodeByIndex(int index), возвращающий ссылку на объект элемента списка по его номеру. Нумерация значащих элементов должна начинаться с 0.
- 4) В классе LinkedListTabulatedFunction реализовать метод FunctionNode addNodeToTail(), добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента.
- 5) В классе LinkedListTabulatedFunction реализовать метод FunctionNode addNodeByIndex(int index), добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента.
- 6) В классе LinkedListTabulatedFunction реализовать метод FunctionNode deleteNodeByIndex(int index), удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента.

### **Задание 5**

В 5 задании реализовал в этом же классе LinkedListTabulatedFunction конструкторы и методы, аналогичные конструкторам и методам класса TabulatedFunction. Конструкторы должны иметь те же параметры, методы должны иметь те же сигнатуры. Также должны выбрасываться те же виды исключений в тех же случаях.

```

93 }
94     public double getLeftDomainBorder() { 4 usages new *
95         if (pointsCount == 0) return Double.NaN;
96         return head.getNext().getPoint().getX(); // Первая точка
97     }
98     public double getRightDomainBorder() { 2 usages new *
99         if (pointsCount == 0) return Double.NaN;
100        return head.getPrev().getPoint().getX(); // Последняя точка
101    }
102    public double getFunctionValue(double x) { 4 usages new *
103        final double EPSILON = 1e-10;
104
105        if (pointsCount == 0) return Double.NaN;
106
107        double leftX = getLeftDomainBorder();
108        double rightX = getRightDomainBorder();
109
110        // проверка границ (с учетом погрешности)
111        if (x < leftX - EPSILON || x > rightX + EPSILON) {
112            return Double.NaN;
113        }
114        // поиск точного совпадения x
115        FunctionNode current = head.getNext();
116        while (current != head) {
117            if (Math.abs(current.getPoint().getX() - x) < EPSILON) {
118                return current.getPoint().getY();
119            }
120            current = current.getNext();
121        }
122
123        // Поиск интервала для линейной интерполяции
124        FunctionNode node1 = head.getNext();
125        FunctionNode node2 = node1.getNext();
126
127        while (node2 != head) {
128            if (x >= node1.getPoint().getX() - EPSILON &&
129                x <= node2.getPoint().getX() + EPSILON) {
130
131                double x1 = node1.getPoint().getX();
132                double y1 = node1.getPoint().getY();
133                double x2 = node2.getPoint().getX();
134                double y2 = node2.getPoint().getY();
135
136                // Линейная интерполяция: y = y1 + (y2-y1)*(x-x1)/(x2-x1)
137                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
138            }
139            node1 = node2;
140            node2 = node2.getNext();

```

Рис.9

## Задание 6

В предпоследнем пункте надо было переименовать класс TabulatedFunction в класс ArrayTabulatedFunction.

Также надо было создать интерфейс TabulatedFunction, содержащий объявления общих методов классов ArrayTabulatedFunction и LinkedListTabulatedFunction, чтобы оба класса функций реализовывали созданный интерфейс.(Рис.10)

```
package functions;

public interface TabulatedFunction { no usages new *
    // методы области определения
    double getLeftDomainBorder(); no usages new *
    double getRightDomainBorder(); no usages new *
    double getFunctionValue(double x); no usages new *

    // методы работы с точками
    int getPointsCount(); no usages new *
    FunctionPoint getPoint(int index); no usages new *
    double getPointX(int index); no usages new *
    double getPointY(int index); no usages new *
    void setPointY(int index, double y); no usages new *
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; no usages new *
    void setPointX(int index, double x) throws InappropriateFunctionPointException; no usages new *
    void deletePoint(int index); no usages new *
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; no usages new *
}
```

Рис.10

### Задание 7

В последнем пункте проверяем работу написанных классов. в созданном ранее классе Main, содержащем точку входа программы, добавим проверку для случаев, в которых объект табулированной функции должен выбрасывать исключения.(Рис.11-14)

```
f(x) = 2x + 1 на [-2.0, 4.0]
Точек: 7
Точки после создания функции:
Точка 0:(-2.0,-3.0)
Точка 1:(-1.0,-1.0)
Точка 2:(0.0,1.0)
Точка 3:(1.0,3.0)
Точка 4:(2.0,5.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
Тестирование интерполяции (промежуточные значения)
Интерполяция между точками (-1,-1) и (0,1):
f(-0.5) = 0.0 (ожидается: 0.0)
f(0.5) = 2.0 (ожидается: 2.0)
точка левее границы:
f(-3.0) = NaN (слева)
точка правее границы:
f(5.0) = NaN (справа)

Тест самих значений
f(-4.0) = NaN
f(-2.0) = -3.0
f(0.0) = 1.0
f(2.0) = 5.0
f(4.0) = 9.0
f(6.0) = NaN
Точки после setPointY(3, 10):
Точка 0:(-2.0,-3.0)
Точка 1:(-1.0,-1.0)
Точка 2:(0.0,1.0)
Точка 3:(1.0,10.0)
Точка 4:(2.0,5.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
setPointY выведено из отсчета
```

```
Точки после setPointX(1, -0.7):
Точка 0:(-2.0,-3.0)
Точка 1:(-0.7,-1.0)
Точка 2:(0.0,1.0)
Точка 3:(1.0,10.0)
Точка 4:(2.0,5.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
addPoint выполнен успешно
Точки после addPoint(2.5, 6):
Точка 0:(-2.0,-3.0)
Точка 1:(-0.7,-1.0)
Точка 2:(0.0,1.0)
Точка 3:(1.0,10.0)
Точка 4:(2.0,5.0)
Точка 5:(2.5,6.0)
Точка 6:(3.0,7.0)
Точка 7:(4.0,9.0)
deletePoint выполнен успешно
Точки после deletePoint(2):
Точка 0:(-2.0,-3.0)
Точка 1:(-0.7,-1.0)
Точка 2:(1.0,10.0)
Точка 3:(2.0,5.0)
Точка 4:(2.5,6.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
Финальный результат:
Всего точек:7
f(2.5)=6.0
```

```
----- Тестирование LinkedListTabulatedFunction -----
LinkedListTabulatedFunction создан успешно
Исходное количество точек: 7
Исходные точки:
Точка 0:(-2.0,-3.0)
Точка 1:(-1.0,-1.0)
Точка 2:(0.0,1.0)
Точка 3:(1.0,3.0)
Точка 4:(2.0,5.0)
Точка 5:(3.0,7.0)
Точка 6:(4.0,9.0)
удаление точки с индексом 0
Удаляем первую точку (индекс 0): (-2.0, -3.0)
После удаления:
Количество точек: 6
Первая точка теперь: (-1.0, -1.0)
Все точки после удаления первой:
Точка 0:(-1.0,-1.0)
Точка 1:(0.0,1.0)
Точка 2:(1.0,3.0)
Точка 3:(2.0,5.0)
Точка 4:(3.0,7.0)
Точка 5:(4.0,9.0)
Добавление и удаление
После добавления точки (-3, -5) в начало:
ВСЕ точки после добавления:
Точка 0:(-1.0,-1.0)
Точка 1:(0.0,1.0)
Точка 2:(1.0,3.0)
Точка 3:(2.0,5.0)
Точка 4:(3.0,7.0)
Точка 5:(4.0,9.0)
После повторного удаления первой точки:
```

```
Точка 1:(0.0,1.0)
Точка 2:(1.0,3.0)
Точка 3:(2.0,5.0)
Точка 4:(3.0,7.0)
Точка 5:(4.0,9.0)
Добавление и удаление
После добавления точки (-3, -5) в начало:
ВСЕ точки после добавления:
Точка 0:(-1.0,-1.0)
Точка 1:(0.0,1.0)
Точка 2:(1.0,3.0)
Точка 3:(2.0,5.0)
Точка 4:(3.0,7.0)
Точка 5:(4.0,9.0)
После повторного удаления первой точки:
все точки после удаления:
Точка 0:(0.0,1.0)
Точка 1:(1.0,3.0)
Точка 2:(2.0,5.0)
Точка 3:(3.0,7.0)
Точка 4:(4.0,9.0)
Удаление последней точки
Удаляем последнюю точку (индекс 4): (4.0, 9.0)
После удаления последней точки, новая последняя точка: (3.0, 7.0)
Изменение значений точек
Меняем значение второй точки (индекс 1)
Y изменен с 3.0 на 30.0
Интерполяция в LinkedList
Интерполяция f(0.5) = 15.5
Поймано FunctionPointIndexOutOfBoundsException: значение точки вне границ: 100
проверка исключений
поймано IllegalArgumentException (границы): Левая граница является правой в данном случае.
Поймано IllegalArgumentException (точек < 2): Количество точек должно быть >= 2
```

Рис. 11-14