МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 3

«Работа с эксепшенами и интерфейсами в наборе классов табулированной функции»

по курсу
Объектно-ориентированное программирование

Выполнил: Петрухин Роман,
студент группы 6203-010302D

Оглавление

Пункт №1

В качестве своего первого задания я создал два класса эксепшенов «FunctionPointIndexOutOfBoundsException», «InappropriateFunctionPointException». Их реализация приведена на рисунках 1, 2. Они наследуют от классов «IndexOutOfBoundsException» и «Exception» соответственно.

```
public class FunctionPointIndexOutOfBoundsException
        extends IndexOutOfBoundsException {

}
```

Рисунок 1

```
public class InappropriateFunctionPointException
        extends Exception {
}
```

Рисунок 2

Пункт №2

Следуя условию Л.Р. я добавил выбрасывание эксепешенов из функций класса «TabulatedFunction». Эксепшены, которые я выбрасывал: «FunctionPointIndexOutOfBoundsException», «InappropriateFunctionPointException», «IllegalStateException» и «IllegalArgumentException». Дополненную реализацию конструкторов и функций я привел на рисунках 3,4,5. (На скриншотах класс называется вместо «TabulatedFunction» - «ArrayTabulatedFunction», т.к. отчёт пишется после выполнения ЛР)

```java
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) 3 usages  new *
        throws IllegalArgumentException {
    if (leftX >= rightX)
        throw new IllegalArgumentException("Left border must be less than right border!");
    if (pointsCount < 2)
        throw new IllegalArgumentException("Count of points must be at least 2!");

    this.amountOfElements = pointsCount;
    this.massiveOfPoints = new FunctionPoint[pointsCount];

    double distance = (rightX - leftX) / (amountOfElements - 1);
    for(int i = 0; i < amountOfElements; i++) {
        double x = leftX + i * distance;
        this.massiveOfPoints[i] = new FunctionPoint(x, y: 0);
    }
}
```

Рисунок 3

```java
public FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException {  1 usage  new
    if (index < 0 || index >= this.amountOfElements)
        throw new FunctionPointIndexOutOfBoundsException();
    return new FunctionPoint(massiveOfPoints[index].getX(), massiveOfPoints[index].getY());
}

/**
 * Заменяет точку по указанному индексу на новую
 * @param index индекс заменяемой точки
 * @param point новая точка (не может быть null)
 */
public void setPoint(int index, FunctionPoint point) 2 usages  new *
        throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
    if (index < 0 || index >= amountOfElements || point == null)
        throw new FunctionPointIndexOutOfBoundsException();
    if ((index > 0 && point.getX() <= massiveOfPoints[index - 1].getX()) ||
            (index < amountOfElements - 1 && point.getX() >= massiveOfPoints[index + 1].getX()))
        throw new InappropriateFunctionPointException();
    massiveOfPoints[index] = new FunctionPoint(point.getX(), point.getY());
}
```

Рисунок 4

```java
public void setPointX(int index, double x) 2 usages  new *
        throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException {
    if (index < 0 || index >= amountOfElements) {
        throw new FunctionPointIndexOutOfBoundsException();
    }
    if ((index > 0 && x <= massiveOfPoints[index - 1].getX()) ||
            (index < amountOfElements - 1 && x >= massiveOfPoints[index + 1].getX())) {
        throw new InappropriateFunctionPointException();
    }
    massiveOfPoints[index] = new FunctionPoint(x, massiveOfPoints[index].getY());
}
```

Рисунок 5

Пункт №3

Во время выполнения следующего задания я вспомнил что такое «двусвязный циклический список» и как он пишется. Также были выполнены все 6 пунктов представленных в качестве задания (сами пункты представлены на рисунке 6).

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. Выберите и обоснуйте место описания класса и его видимость. Также выберите и обоснуйте реализацию инкапсуляции в этом классе.
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 (голова списка в данном случае к ним не относится) должна начинаться с 0. Метод должен обеспечивать оптимизацию доступа к элементам списка.
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.

Рисунок 6

Перед выполнением данных пунктов я создал класс «LinkedListTabulatedFunction». Внутри него я создал еще один приватный статический класс «FunctionPoint». Почему именно приватный и статический? Приватный - чтобы никто из вне не смог сломать структуру списка, статический - чтобы экономить память (не тратить её на хранение внешних ссылок на объект) и чтобы структура была более надежной (управление узлами сосредоточено в классе и они не могут «случайно» разрушить, созданную структуру) Реализация класса представлена на рисунке 7.

```
private static class FunctionNode {  26 usages  new *
    FunctionPoint point;  3 usages
    FunctionNode next;  2 usages
    FunctionNode prev;  2 usages

    /**
     * Конструктор узла с заданной точкой функции
     * @param point точка функции для хранения в узле
     */
    FunctionNode(FunctionPoint point) {  3 usages  new *
        this.point = point;
    }
```

Рисунок 7

Также для этого внутреннего класса были реализованы геттеры и сеттеры, их реализация совершенно стандартна.

Далее уже в теле класса «LinkedListTabulatedFunction» была создана «голова» списка и счетчик count, созданный для подсчета элементов в списке. Реализация представлена на рисунке 8.

```
private FunctionNode head = new FunctionNode( point: null);

{
    head.setPrev(head);
    head.setNext(head);
}

private int count;  21 usages
```

Рисунок 8

Далее была реализована функция getNodeByIndex(int index). Я попытался оптимизировать её путем использования не только головы, но и хвоста в зависимости от того промежутка в который попадает index. Если index < count / 2, то я искал индекс начиная с головы, если же нет, то начиная с хвоста. Реализация функции представлена на рисунке 9.

```
FunctionNode getNodeByIndex(int index)  12 usages  new *
        throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index >= count) {
        throw new FunctionPointIndexOutOfBoundsException();
    }

    FunctionNode current;

    if (index < count / 2) {
        // Двигаемся от головы вперед
        current = head.getNext(); // начинаем с 1ого значащего элемента
        for(int i = 0; i < index; i++) {
            current = current.getNext();
        }
    }
    else {
        // Двигаемся от хвоста назад
        current = head.getPrev(); // начинаем с последнего элемента
        for(int i = count - 1; i > index; i--) {
            current = current.getPrev();
        }
    }
```

Рисунок 9

Метод addNodeToTail(FunctionPoint point) я реализовал довольно банально, используя сеттеры класса «FunctionNode» я просто перенаправил ссылки, вставив новый узел. Реализация представлена на рисунке 10.

```
FunctionNode addNodeToTail(FunctionPoint point) {
    FunctionNode newNode = new FunctionNode(point)

    FunctionNode tail = head.getPrev(); // текущий

    // Вставляем newNode между tail и head
    newNode.setPrev(tail);
    newNode.setNext(head);
    tail.setNext(newNode);
    head.setPrev(newNode);

    count++;
    return newNode;
}
```

Рисунок 10

Реализация функций добавления и удаления по индексу выполнена примерно в том же качестве, что и добавление в конец. Единственным изменением внутри функций является использование getNodeByIndex() для нахождения элемента рядом с которым будем вставлять или то который будем удалять. Реализация представлена на рисунках 11, 12.

```
FunctionNode addNodeByIndex(int index, FunctionPoint point)
        throws FunctionPointIndexOutOfBoundsException {
    if (index < 0 || index > count) {
        throw new FunctionPointIndexOutOfBoundsException();
    }

    if (index == count) {
        return addNodeToTail(point);
    }

    FunctionNode targetNode = getNodeByIndex(index);
    FunctionNode newNode = new FunctionNode(point);

    newNode.setPrev(targetNode.getPrev());
    newNode.setNext(targetNode);
    targetNode.getPrev().setNext(newNode);
    targetNode.setPrev(newNode);

    count++;
    return newNode;
}
```

Рисунок 11

```
FunctionNode deleteNodeByIndex(int index) {  1 usage   new *
    if (index < 0 || index >= count) {
        throw new FunctionPointIndexOutOfBoundsException();
    }

    FunctionNode targetNode = getNodeByIndex(index);

    targetNode.getPrev().setNext(targetNode.getNext());
    targetNode.getNext().setPrev(targetNode.getPrev());

    count--;
    return targetNode;
}
```

Рисунок 12

Пункт №4

В следующем задании я начал добавлять функционал «TabulatedFunction» в «LinkedListTabulatedFunction». В том же задании был вопрос о том в каких функциях я получу выигрыш, благодаря созданию списка. Сначала скажу о тех где точно ничего не изменилось. Все как было так и осталось в конструкторах, геттерах-сеттерах. Значительный выигрыш в памяти я получил в функции add. Add дал прирост в скорости за счет того, что мне не нужно расширять массив, как было изначально, а достаточно создать новый узел (На больших данных списковая add даст очень сильный прирост). Также скорее всего значительный выигрыш на большом количестве данных даст еще функция delete (Списочный её вариант гораздо лучше управляет памятью). Реализация addPoint() представлена на риснуке 13.

```
public void addPoint(FunctionPoint point) {  2 usages  new *
    if (point == null) {
        throw new IllegalArgumentException("Point cannot be null");
    }

    // Проверка на дупликат
    FunctionNode current = head.getNext();
    while (current != head) {
        if (Math.abs(current.getPoint().getX() - point.getX()) < 1e-10) {
            throw new IllegalArgumentException("Point X=" + point.getX() + " is being");
        }
        current = current.getNext();
    }

    // Поиск позиции для вставки с сохранением упорядоченности
    current = head.getNext();
    int insertIndex = 0;

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

    addNodeByIndex(insertIndex, point);
}
```

Рисунок 13

Пункт №5

Переименование класса сделал при помощи рефакторинга в IDEA. Создал интерфейс «TabulatedFunction» в который вынес реализацию всех основых функций. Его реализация представлена на рисунках 14, 15, 16.

```java
package functions;

public interface TabulatedFunction {  7 usages  2 implementations  zhestok1 *

    /**
     * Возвращает левую границу области определения функции
     * @return значение левой границы области определения
     */
    double getLeftDomainBorder();  2 usages  2 implementations  new *

    /**
     * Возвращает правую границу области определения функции
     * @return значение правой границы области определения
     */
    double getRightDomainBorder();  2 usages  2 implementations  new *

    /**
     * Вычисляет значение функции в заданной точке с помощью линейной интерполяции
     * @param x координата, в которой вычисляется значение функции
     * @return значение функции в точке x или Double.NaN, если x вне области определения
     */
    double getFunctionValue(double x);  5 usages  2 implementations  new *
```

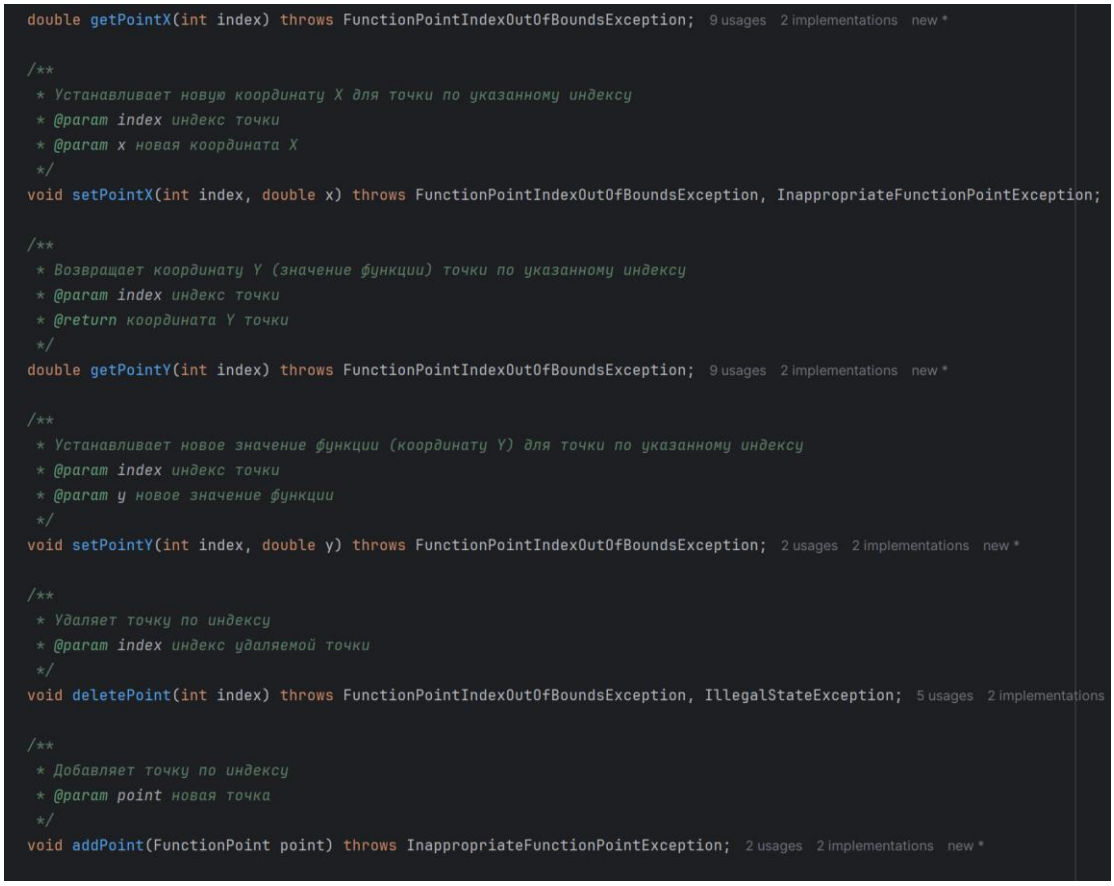
Рисунок 14

```java
    /**
     * Возвращает количество точек в табличной функции
     * @return количество точек
     */
    int getPointsCount();  8 usages  2 implementations  new *

    /**
     * Возвращает копию точки по указанному индексу
     * @param index индекс точки (от 0 до pointsCount-1)
     * @return копия объекта FunctionPoint
     */
    FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException;  1 usage  2 imp

    /**
     * Заменяет точку по указанному индексу на новую
     * @param index индекс заменяемой точки
     * @param point новая точка (не может быть null)
     */
    void setPoint(int index, FunctionPoint point)  2 usages  2 implementations  new *
            throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
```

Рисунок 15



```
double getPointX(int index) throws FunctionPointIndexOutOfBoundsException;  9 usages  2 implementations  new *

/**
 * Устанавливает новую координату X для точки по указанному индексу
 * @param index индекс точки
 * @param x новая координата X
 */
void setPointX(int index, double x) throws FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;

/**
 * Возвращает координату Y (значение функции) точки по указанному индексу
 * @param index индекс точки
 * @return координата Y точки
 */
double getPointY(int index) throws FunctionPointIndexOutOfBoundsException;  9 usages  2 implementations  new *

/**
 * Устанавливает новое значение функции (координату Y) для точки по указанному индексу
 * @param index индекс точки
 * @param y новое значение функции
 */
void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;  2 usages  2 implementations  new *

/**
 * Удаляет точку по индексу
 * @param index индекс удаляемой точки
 */
void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;  5 usages  2 implementations

/**
 * Добавляет точку по индексу
 * @param point новая точка
 */
void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;  2 usages  2 implementations  new *
```

Рисунок 16

Пункт №6

В main на прошлой ЛР я демонстрировал работоспособность своего кода на функции синуса. Но после устной встречи я полнял свою ошибку и заменил синус на обычную линейную функцию. Я добавил в качестве тестов демонстрацию работы кода на ошибках (чтобы выбрасывались эксепшены). Результат работы main представлен на рисунках 17, 18 и 19.

```
=== Линейная функция y = 2x + 1 ===
Исходная функция:
Точка 0: x=0,00, y=1,00
Точка 1: x=2,00, y=5,00
Точка 2: x=4,00, y=9,00
Точка 3: x=6,00, y=13,00
Точка 4: x=8,00, y=17,00
Точка 5: x=10,00, y=21,00

=== Добавление точки (5.5, 12.0) ===
После добавления:
Точка 0: x=0,00, y=1,00
Точка 1: x=2,00, y=5,00
Точка 2: x=4,00, y=9,00
Точка 3: x=5,50, y=12,00
Точка 4: x=6,00, y=13,00
Точка 5: x=8,00, y=17,00
Точка 6: x=10,00, y=21,00

=== Удаление точки с индексом 2 ===
После удаления:
Точка 0: x=0,00, y=1,00
Точка 1: x=2,00, y=5,00
Точка 2: x=5,50, y=12,00
Точка 3: x=6,00, y=13,00
Точка 4: x=8,00, y=17,00
Точка 5: x=10,00, y=21,00

=== Изменение точки с индексом 1 ===
После изменения:
Точка 0: x=0,00, y=1,00
Точка 1: x=3,00, y=7,00
Точка 2: x=5,50, y=12,00
Точка 3: x=6,00, y=13,00
Точка 4: x=8,00, y=17,00
Точка 5: x=10,00, y=21,00
```

Рисунок 17

```
=== Интерполяция значений ===
f(2,50) = 2*2,50 + 1 ≈ 6,0000
f(7,20) = 2*7,20 + 1 ≈ 15,4000
f(12,00) = NaN (вне области определения)

=== Границы области определения ===
Левая граница: 0,00
Правая граница: 10,00

=== Изменение координат по отдельности ===
До изменения:
Точка 0: x=0,00, y=1,00
После изменения:
Точка 0: x=0,50, y=2,00

=== Итоговое состояние функции ===
Точка 0: x=0,50, y=2,00
Точка 1: x=3,00, y=7,00
Точка 2: x=5,50, y=12,00
Точка 3: x=6,00, y=13,00
Точка 4: x=8,00, y=17,00
Точка 5: x=10,00, y=21,00
Всего точек: 6


=== ДЕМОНСТРАЦИЯ ИСКЛЮЧЕНИЙ ===

1. Тест некорректного конструктора:
✓ Поймано исключение: Left border must be less than right border!

2. Тест выхода за границы индекса:
✓ Поймано исключение: FunctionPointIndexOutOfBoundsException

3. Тест нарушения упорядоченности:
✓ Поймано исключение: InappropriateFunctionPointException
```

Рисунок 18

```
4. Тест добавления точки с существующим X:
✓ Поймано исключение: InappropriateFunctionPointException

5. Тест удаления при малом количестве точек:
✓ Поймано исключение: Cannot delete point: minimum 3 points required

6. Тест некорректного индекса при удалении:
✓ Поймано исключение: FunctionPointIndexOutOfBoundsException

7. Тест нарушения упорядоченности при setPointX:
✓ Поймано исключение: InappropriateFunctionPointException

8. Тест создания функции с 1 точкой:
✓ Поймано исключение: Count of points must be greater than 2!

9. Тест некорректного индекса при getPointY:
✓ Поймано исключение: FunctionPointIndexOutOfBoundsException


=== LinkedListTabulatedFunction ===
Функция создана через LinkedListTabulatedFunction:
Точка 0: x=0,00, y=1,00
Точка 1: x=1,00, y=3,00
Точка 2: x=2,00, y=5,00
Точка 3: x=3,00, y=7,00
Точка 4: x=4,00, y=9,00

Интерполяция для LinkedListTabulatedFunction:
f(1,50) ≈ 4,0000
f(2,80) ≈ 6,6000

=== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ ===
```

Рисунок 19