ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 3

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

по курсу Объектно-ориентированное программирование

Выполнил:
Родомакин Михаил
6203-010302D

Оглавление

Пункт №1 и №2

По первому заданию я изучил стандартные классы исключений Java, после чего внутри пакета functions были созданы длва класса исключений: «FunctionPointIndexOutOfBoundsException» - наследует от класса “IndexOutOfBoundsException” и вызывается, когда мы обращаемся к точке по неподходящему индексу.
«InappropriateFunctionPointException» - наследует от класса “Exception” и используется когда мы пытаемся обратится к неуместной для нашей структуры точке.

```
package functions;

public class FunctionPointIndexOutOfBoundsException
    extends IndexOutOfBoundsException {

}
```

Рисунок 1

```
package functions;

public class InappropriateFunctionPointException
    extends  Exception {

}
```

Рисунок 2

Пункт №3

По второму заданию требовалось внести изменения в класс TabulatedFunction, чтобы конструкторы, а также перечисленные в задании методы выбрасывали соответствующие ситуации исключения при неверных входных данных, чтобы обеспечить корректную работу кода. Были использованы такие классы исключений, как FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException, IllegalStateException и IllegalArgumentException.

```
//Конструктор по количеству элементов
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {  3 usages

        if (leftX > rightX) { //Проверка на соответствие левой и правой границы
            throw new IllegalArgumentException("Right border must be greater than left border");
        }

        if (pointsCount < 2){
            throw new IllegalArgumentException("There must be at least 2 points in tabulated function");
        }
```

Рисунок 3

```
//Конструктор по значениям
public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {  no usages

    if (leftX > rightX) { //Проверка на соответствие левой и правой границы
        throw new IllegalArgumentException("Right border must be greater than left border");
    }

    if(values == null || values.length == 0) { // Защита от null и пустого массива
        throw new IllegalArgumentException("There must be at least 2 points in tabulated function");
    }
```

Рисунок 4

```
public FunctionPoint getPoint(int index) {  1 usage  1 override
    if(index<0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }
    return new FunctionPoint(Points[index]);
}

public double getPointX(int index) {  8 usages  1 override
    if(index<0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }
    return this.Points[index].getX();
}
public double getPointY(int index) {  8 usages  1 override
    if(index<0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }
    return this.Points[index].getY();
}
```

Рисунок 5

```java
//Сеттеры
public void setPoint(int index, FunctionPoint point)  2 usages  1 override
    throws InappropriateFunctionPointException {
    if (index < 0 || index >= length || point == null)
        throw new FunctionPointIndexOutOfBoundsException(); //Проверка индекса
    //Проверка попадания в интервалы
    if (index>0 && point.getX() - EPSILON <= Points[index - 1].getX())
        throw new InappropriateFunctionPointException();
    Points[index] = new FunctionPoint(point);
}

public void setPointX(int index, double x)  2 usages  1 override
    throws InappropriateFunctionPointException {
    if(index<0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }

    //Проверка попадания в интервалы
    if ( index > 0 && x - EPSILON <= Points[index - 1].getX() ||
            (index < length - 1 && x >= Points[index + 1].getX())) {
        throw new InappropriateFunctionPointException();
    }

    Points[index].setX(x);
}

public void setPointY(int index, double y) {  2 usages  1 override
    if(index<0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }
    Points[index].setY(y);
}
```

Рисунок 6

```java
public void deletePoint(int index) {  3 usages  1 override
    if(index<0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }

    if (length < 3) {
        throw new IllegalStateException();
    }
```

Рисунок 7

```java
public void addPoint(FunctionPoint point)  2 usages  1 override
    throws InappropriateFunctionPointException {
    if (point == null) {
        throw new InappropriateFunctionPointException();
    }

    int index = 0; //Ищем место для новой точки по x
    while (index < length && point.getX() - EPSILON > Points[index].getX()) {
        index++;
    }
    if(Math.abs(point.getX() - Points[index].getX()) <= EPSILON) { //Если совпадает с другой точкой
        throw  new InappropriateFunctionPointException();
    }
```

Рисунок 8

Пункт №4

В четвертом задании требовалось создать класс LinkedListTabulatedFunction, представляющий собой двусвязный циклический цикл.

Чтобы сделать это внутри основного класса был создан вложенный класс FunctionNode с модификаторами private и static, содержащий точку (поле типа FunctionPoint) и ссылки на предыдущий и следующий элемент списка.

Модификатор private позволяет сохранить структуру списка, а static делает эту струтуру более надежной и экономит память.

Сам основной класс LinkedListTabulatedFunction содержит два поля – голову типа FunctionNode, не имеющую значения, но содержащую ссылки на следующий и предыдущий элементы, и поле длины списка.

Для дальнейшей работы со списком по заданию были добавлены методы addNodeToTail(), deleteNodeByIndex() и getNodeByIndex(int index).

Последний метод был по заданию оптимизирован: для поиска нужного нода мы сначала определяем выгодное направление поиска по списку.

```
public class LinkedListTabulatedFunction {  2 usages
    private final double EPSILON = 1e-9;  7 usages
    private static class FunctionNode {  19 usages
        FunctionPoint point;  16 usages
        FunctionNode prev;  15 usages
        FunctionNode next;  18 usages

        FunctionNode(FunctionPoint point) { //Конструктор узла по заданной точке  2 usages
            this.point = point;
        }

        public FunctionNode(){ //Конструктор для создания головы  1 usage
            this.prev = this;
            this.next = this;
        }


    }
    private FunctionNode head = new FunctionNode();  13 usages
    private int length;  16 usages
```

Рисунок 9

```
private FunctionNode getNodeByIndex(int index) {  11 usages
    if (index < 0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }

    FunctionNode currentElement;

    if (index < length / 2) { //Выбираем сторону поиска для оптимизации
        currentElement = head.next;
        for (int i = 0; i < index; i++) {
            currentElement = currentElement.next;
        }
    }
    else {
        currentElement = head.prev;
        for (int i = length - 1; i > index; i--) {
            currentElement = currentElement.prev;
        }
    }
    return currentElement;
}

FunctionNode addNodeToTail(FunctionPoint point){  2 usages
    length++;
    FunctionNode newPoint = new FunctionNode(point);
    FunctionNode tail = head.prev;

    tail.next = newPoint;
    head.prev = newPoint;

    newPoint.prev = tail;
    newPoint.next = head;

    return  newPoint;
}
```

Рисунок 10

```
FunctionNode addNodeByIndex(int index, FunctionPoint point){
    FunctionNode targetPosition = getNodeByIndex(index);
    FunctionNode newPoint = new FunctionNode(point);
    length++;

    newPoint.prev = targetPosition.prev;
    newPoint.next = targetPosition;

    targetPosition.prev.next = newPoint;
    targetPosition.prev = newPoint;

    return newPoint;
}

FunctionNode deleteNodeByIndex(int index){  1 usage
    FunctionNode deletedPoint = getNodeByIndex(index);
    length--;

    deletedPoint.prev.next = deletedPoint.next;
    deletedPoint.next.prev = deletedPoint.prev;

    return deletedPoint;
}
```

Рисунок 11

Пункт №5

В пятом задании требуется перенести весь функционал класса TabulatedFunction в новый класс LinkedListTabulatedFunction, в частности требовалось перенести конструкторы и методы.
Согласно заданию, при переходе к новому классу некоторые методы благодаря его особенностям должны были получить выигыш с точки зрения оптимизации. Это действительно так. В функции addPoint() из-за отсутствия необходимости в пересоздании всего списка работа значительно ускорилась.
Также deletePoint() в новом варианте эффективнее работает с памятью.

```
public void addPoint(FunctionPoint point)  no usages
        throws InappropriateFunctionPointException {
    if (point == null) {
        throw new  InappropriateFunctionPointException();
    }

    FunctionNode current = head.next;
    while (current != head) {
        if (Math.abs(current.point.getX() - point.getX()) < EPSILON) { //Если совпадает с другой точкой
            throw new InappropriateFunctionPointException();
        }
        current = current.next;
    }

    current = head.next;
    int newIndex = 0; //Ищем место для новой точки по x

    while (current != head && point.getX() > current.point.getX()) {
        current = current.next;
        newIndex++;
    }

    addNodeByIndex(newIndex, point);
}
```

Рисунок 12

```
public void deletePoint(int index) {  no usages
    if (index < 0 || index >= length) {
        throw new FunctionPointIndexOutOfBoundsException();
    }
    if (length < 3) {
        throw new IllegalStateException();
    }
    deleteNodeByIndex(index);
}
```

Рисунок 13

Пункт №6

Шестой пункт требует переименовать класс TabulatedFunction в ArrayTabulatedFunction для дальнейшего использования интерфейса TabulatedFunction, который содержит объявления общих методов двух классов.

```
package functions;

public interface TabulatedFunction {  6 usages  1 implementation
    double getLeftDomainBorder();  2 usages  1 implementation

    double getRightDomainBorder();  2 usages  1 implementation

    double getFunctionValue(double x);  2 usages  1 implementation

    int getPointsCount();  7 usages  1 implementation

    FunctionPoint getPoint(int index);  1 usage  1 implementation

    void setPoint(int index, FunctionPoint point)  2 usages  1 implementation
            throws InappropriateFunctionPointException;

    double getPointX(int index);  8 usages  1 implementation

    void setPointX(int index, double x)  2 usages  1 implementation
            throws InappropriateFunctionPointException;

    double getPointY(int index);  8 usages  1 implementation

    void setPointY(int index, double y);  2 usages  1 implementation

    void deletePoint(int index);  3 usages  1 implementation

    void addPoint(FunctionPoint point)  2 usages  1 implementation
            throws InappropriateFunctionPointException;
}
```

Рисунок 14

Пункт №7

Послдний пункт лабораторной работы требует провести проверку обоих классов на примере некоторой функции.
Учитывая усложнение работы была выбрана простейшая функция y=3x+3 для более очевидных результатов.
В main помимо проверок работы всех методов были также добавлены проверки работы исключений, а для проверки разных классов, в соответствии с заданием, достаточно поменять класс создаваемого в начале объекта.

```
----- Линейная функция y = 3x+3 -----
Исходная функция:
Точка 0: x=0.0 y=3.0
Точка 1: x=2.0 y=9.0
Точка 2: x=4.0 y=15.0
Точка 3: x=6.0 y=21.0
Точка 4: x=8.0 y=27.0
Точка 5: x=10.0 y=33.0

----- Добавление точки (3; 12) -----
После добавления:
Точка 0: x=0.0 y=3.0
Точка 1: x=2.0 y=9.0
Точка 2: x=3.0 y=12.0
Точка 3: x=4.0 y=15.0
Точка 4: x=6.0 y=21.0
Точка 5: x=8.0 y=27.0
Точка 6: x=10.0 y=33.0

----- Удаление точки с индексом 3 -----
После удаления:
Точка 0: x=0.0 y=3.0
Точка 1: x=2.0 y=9.0
Точка 2: x=3.0 y=12.0
Точка 3: x=6.0 y=21.0
Точка 4: x=8.0 y=27.0
Точка 5: x=10.0 y=33.0

----- Изменение точки с индексом 4 -----
После изменения:
Точка 0: x=0.0 y=3.0
Точка 1: x=2.0 y=9.0
Точка 2: x=3.0 y=12.0
Точка 3: x=6.0 y=21.0
Точка 4: x=7.0 y=24.0
Точка 5: x=10.0 y=33.0
```

Рисунок 15

```
----- Пример интерполяции -----
getFunctionValue(3,5) = 13,50 (3*3,5 + 3)
getFunctionValue(6,8) = 23,40 (3*6,8 + 3)

----- Границы функции -----
Левая граница: 0.0
Правая граница: 10.0

----- Изменение координат по отдельности -----
До изменения:
Точка 0: x=0.0 y=3.0
После изменения:
Точка 0: x= 0.5 y= 2.0

----- Функция после всех операций -----
Точка 0: x=0.5 y=2.0
Точка 1: x=2.0 y=9.0
Точка 2: x=3.0 y=12.0
Точка 3: x=6.0 y=21.0
Точка 4: x=7.0 y=24.0
Точка 5: x=10.0 y=33.0
Всего точек: 6
```

Рисунок 16

```
----- Демонстрация исключений -----
1) Тест некорректного конструктора:
=ArrayTabulatedFunction: Исключение: IllegalArgumentException
=LinkedListTabulatedFunction: Исключение: IllegalArgumentException

2) Тест выхода за границы индекса в getPoint():
Исключение: FunctionPointIndexOutOfBoundsException

3) Тест нарушения упорядоченности в setPoint:
Исключение: InappropriateFunctionPointException

4) Тест addPoint() с уже существующим X:
Исключение: InappropriateFunctionPointException

5) Тест deletePoint() при малом количестве точек:
Исключение IllegalStateException

6) Тест некорректного индекса в deletePoint():
Исключение: FunctionPointIndexOutOfBoundsException

7) Тест нарушения упорядоченности при setPointX():
Исключение: InappropriateFunctionPointException

8) Тест создания функции с 1 точкой:
Исключение: IllegalArgumentException

9) Тест некорректного индекса при getPointY:
Исключение: FunctionPointIndexOutOfBoundsException

Process finished with exit code 0
```

Рисунок 17