Лабораторная работа № 3

Выполнила: Оленина Арина Игоревна
группа 6204-010302D

Оглавление

Задание 1

Я ознакомилась со следующими классами исключений, входящих в API Java:

• java.lang.Exception • java.lang.IndexOutOfBoundsException • java.lang.ArrayIndexOutOfBoundsException • java.lang.IllegalArgumentException • java.lang.IllegalStateException

Задание 2

В пакете functions создала два класса исключений:
• FunctionPointIndexOutOfBoundsException – исключение выхода за границы набора точек при обращении к ним по номеру, наследует от класса IndexOutOfBoundsException;
• InappropriateFunctionPointException – исключение, выбрасываемое при попытке добавления или изменения точки функции несоответствующим образом, наследует от класса Exception.
При создании классов исключений я использовала специализированные средства среды разработки (Exception).

Результат:

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException {

  public FunctionPointIndexOutOfBoundsException(String message) {

    super(message);

  }

}

package functions;

public class InappropriateFunctionPointException extends Exception {

    public InappropriateFunctionPointException(String message) {

        super(message);

    }

}
```

Задание 3

В разработанный ранее класс TabulatedFunction 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 обеспечло создание объекта только при корректных параметрах.
В оба конструктора (по количеству точек и значениям точек в виде массива) добавила

```
if (leftX >= rightX) { //Проверка области определения
            throw new IllegalArgumentException("Левая граница области определения
больше или равна правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек меньше двух");
        }
```

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

```
if (index < 0 || index >= pointsCount) { // Проверка на правильность индекса
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит за
границы: " + index);
        }
```

• 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 обеспечило сохранение упорядоченности точек функции.
В метод setPoint() (и аналогично в setPointX()) я добавила

```
if (index == 0) { // Проверка для первого элемента
            if (newX >= points[1].getX()) {
                throw new InappropriateFunctionPointException("Новая X для первой точки
должна быть меньше следующей точки");
            }
        }
        else if (index == pointsCount - 1) { // Проверка для последнего элемента
            if (newX <= points[pointsCount - 2].getX()) { // строго больше предыдущего
                throw new InappropriateFunctionPointException("Новая X для последней
точки должна быть больше предыдущей точки");
            }
        }
        else { // Проверка для средних элементов
```

```
        if (newX <= points[index - 1].getX() || newX >= points[index + 1].getX()) {
            throw new InappropriateFunctionPointException("Новая X для последней
точки должна быть больше предыдущей точки");
        }
    }
```

А в метод addPoint() добавила

```
for (int i = 0; i < pointsCount; i++) { //Проверка на уникальность x
        if (points[i].getX() == point.getX()) {
            throw new InappropriateFunctionPointException("Точка с такой X уже
существует");
        }
    }
```

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

```
if (pointsCount < 3) { //Если количество точек меньше трех

            throw new IllegalStateException("Невозможно удалить точку: количество
точек меньше трех");

        }
```

Задание 4

В пакете functions создала класс LinkedListTabulatedFunction, объект которого также описывает табулированную функцию. Отличие этого класса заключается в том, что для хранения набора точек в нем используется не массив, а динамическая структура – связный список.
Для полноценного описания связного списка я реализовала два класса. Ответственность первого из них – элемент списка и его связи, объекты этого класса являются объектами элементов списка, хранящими данные и ссылки на соседние элементы. Ответственность второго – список в целом и операции с ним, именно в нем реализуются методы по манипулированию списком, а его объект – это объект списка целиком.

Таким образом, в первом классе есть информационное поле и поля связи. А во втором классе хранится ссылка на объект головы списка и описаны методы для работы со списком, причем публичные методы не получают и не возвращают ссылки на объекты элементов списка, т.к. это нарушение инкапсуляции. Объекты элементов списка являются частями объекта списка и не существуют вне его.

В качестве структуры списка в настоящей работе я использовала двусвязный циклический список с выделенной головой.

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 я выполнила следующие действия.

1. Описала класс элементов списка FunctionNode, содержащий информационное поле для хранения данных типа FunctionPoint, а также поля для хранения ссылок на предыдущий и следующий элемент

```
public class LinkedListTabulatedFunction {
    private static class FunctionNode {
        FunctionPoint point;      // Данные точки
        FunctionNode prev;        // Ссылка на предыдущий элемент
        FunctionNode next;        // Ссылка на следующий элемент
        // Конструктор
        FunctionNode(FunctionPoint point) {
            this.point = point;
        }
```

Я вложила private static class внутри LinkedListTabulatedFunction. Он приватный, так как не должен быть доступен вне класса (внутренняя реализация связного списка) и статический, так как объекты FunctionNode не должны содержать ссылку на внешний объект LinkedListTabulatedFunction. Так же FunctionNode внутри класса LinkedListTabulatedFunction, так как FunctionNode не имеет смысла без этого класса и логически принадлежит ему. Реализация инкапсуляции: все поля приватные и FunctionNode приватный.

2. Описала класс LinkedListTabulatedFunction объектов списка, содержащий поле ссылки на объект головы, а также иные вспомогательные поля.

```
private FunctionNode head; //Голова списка

private int pointsCount; //Количество точек в списке
```

3. В классе LinkedListTabulatedFunction реализовала 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) начинается с 0. Метод обеспечивает оптимизацию доступа к элементам списка.

```
private FunctionNode getNodeByIndex(int index) {
    if (index < 0 || index >= pointsCount) {
        throw new FunctionPointIndexOutOfBoundsException("Индекс выходит за границы: " + index);
    }
    FunctionNode current;
    // Выбор оптимального направления обхода
    if (index < pointsCount / 2) {
        // Ищем с начала (если индекс в первой половине)
        current = head.next;
        for (int i = 0; i < index; i++) {
            current = current.next;
        }
    } else {
        // Ищем с конца (если индекс во второй половине)
        current = head.prev;
        for (int i = pointsCount - 1; i > index; i--) {
            current = current.prev;
        }
    }
    return current;
}
```

4. В классе LinkedListTabulatedFunction реализовала метод FunctionNode addNodeToTail(), добавляющий новый элемент в конец списка и возвращающий ссылку на объект этого элемента.

```
private FunctionNode addNodeToTail() {
        FunctionNode newNode = new FunctionNode(null);
        FunctionNode lastNode = head.prev;
```

```
        // Устанавливаем связи нового узла:
        newNode.prev = lastNode;  // предыдущий для нового - старый последний
        newNode.next = head;      // следующий для нового - голова
        // Обновляем связи соседних узлов:
        lastNode.next = newNode;  // следующий для старого последнего - новый
        head.prev = newNode;      // предыдущий для головы - новый
        pointsCount++;
        return newNode;
    }
```

5. В классе LinkedListTabulatedFunction реализовала метод FunctionNode addNodeByIndex(int index), добавляющий новый элемент в указанную позицию списка и возвращающий ссылку на объект этого элемента.

```
private FunctionNode addNodeByIndex(int index) {
        // Проверка корректности индекса
        if (index < 0 || index > pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит за границы: " + index);
        }
        // Если вставляем в конец, используем существующий метод
        if (index == pointsCount) {
            return addNodeToTail();
        }
        FunctionNode newNode = new FunctionNode(null);
        // Находим узел, который будет находиться после нового (по указанному индексу)
        FunctionNode nextNode = getNodeByIndex(index);
        // Узел, который будет находиться перед новым
        FunctionNode prevNode = nextNode.prev;
        // Устанавливаем связи нового узла:
        newNode.prev = prevNode;  // предыдущий для нового - узел перед nextNode
```

```
        newNode.next = nextNode;  // следующий для нового - nextNode
        // Обновляем связи соседних узлов:
        prevNode.next = newNode;  // следующий для prevNode - новый
        nextNode.prev = newNode;  // предыдущий для nextNode - новый
        pointsCount++;
        return newNode;
    }
```

6. В классе LinkedListTabulatedFunction реализовала метод FunctionNode deleteNodeByIndex(int index), удаляющий элемент списка по номеру и возвращающий ссылку на объект удаленного элемента.

```
private FunctionNode deleteNodeByIndex(int index) {
        // Проверка корректности индекса
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс выходит за границы: " + index);
        }
        // Находим узел для удаления
        FunctionNode nodeToDelete = getNodeByIndex(index);
        // Получаем соседние узлы
        FunctionNode prevNode = nodeToDelete.prev;
        FunctionNode nextNode = nodeToDelete.next;
        // Удаляем узел из списка - перебрасываем связи
        prevNode.next = nextNode;  // следующий для предыдущего - становится следующий от удаляемого
        nextNode.prev = prevNode;  // предыдущий для следующего - становится предыдущий от удаляемого
        // Очищаем связи удаленного узла
        nodeToDelete.prev = null;
        nodeToDelete.next = null;
        pointsCount--;
        return nodeToDelete;
```

```
    }
```

Задание 5

Для обеспечения второй функции
класса LinkedListTabulatedFunction реализовала в классе конструкторы и методы, аналогичные конструкторам и методам класса TabulatedFunction. Конструкторы имеют те же параметры, а методы - те же сигнатуры. Также выбрасываются те же виды исключений в тех же случаях.
Некоторые методы я оптимизировала (getFunctionValue, addPoint). Вышло следующее:

```
public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница области определения больше или равна правой");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек меньше двух");
    }
    // Инициализация списка
    head = new FunctionNode(null);
    head.next = head;
    head.prev = head;
    this.pointsCount = 0;

    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        addNodeToTail().point = new FunctionPoint(x, 0.0);
    }
}

public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница области определения больше или равна правой");
    }
    if (values.length < 2) {
        throw new IllegalArgumentException("Количество точек меньше двух");
    }

    // Инициализация списка
    head = new FunctionNode(null);
    head.next = head;
```

```
    head.prev = head;
    this.pointsCount = 0;

    double step = (rightX - leftX) / (values.length - 1);
    for (int i = 0; i < values.length; i++) {
        double x = leftX + i * step;
        addNodeToTail().point = new FunctionPoint(x, values[i]);
    }
}
public double getLeftDomainBorder() {
    if (pointsCount == 0) return Double.NaN;
    return head.next.point.getX(); // Первый значащий узел
}

public double getRightDomainBorder() {
    if (pointsCount == 0) return Double.NaN;
    return head.prev.point.getX(); // Последний значащий узел
}

public double getFunctionValue(double x) {
    if (pointsCount == 0 || x < getLeftDomainBorder() || x > getRightDomainBorder()) {
        return Double.NaN;
    }
    // Используем прямое обращение к узлам для поиска интервала
    FunctionNode current = head.next;
    while (current != head) {
        FunctionNode next = current.next;
        if (next != head && x >= current.point.getX() && x <= next.point.getX()) {
            // Линейная интерполяция
            double x1 = current.point.getX();
            double x2 = next.point.getX();
            double y1 = current.point.getY();
            double y2 = next.point.getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
        current = current.next;
    }

    return Double.NaN;
}

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

```
public FunctionPoint getPoint(int index) {
    FunctionNode node = getNodeByIndex(index);
    return new FunctionPoint(node.point); // Возвращаем копию
}

public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException {
    FunctionNode node = getNodeByIndex(index);
    double newX = point.getX();

    // Проверка порядка точек
    if (pointsCount == 1) {
        node.point = new FunctionPoint(point);
        return;
    }

    if (index == 0) {
        // Первый элемент: новая X должна быть < следующей
        if (newX >= node.next.point.getX()) {
            throw new InappropriateFunctionPointException("Новая X для первой точки
должна быть меньше следующей точки");
        }
    } else if (index == pointsCount - 1) {
        // Последний элемент: новая X должна быть > предыдущей
        if (newX <= node.prev.point.getX()) {
            throw new InappropriateFunctionPointException("Новая X для последней точки
должна быть больше предыдущей точки");
        }
    } else {
        // Средний элемент: новая X должна быть между соседними
        if (newX <= node.prev.point.getX() || newX >= node.next.point.getX()) {
            throw new InappropriateFunctionPointException("Новая X должна быть между
соседними точками");
        }
    }

    node.point = new FunctionPoint(point);
}

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

public void setPointX(int index, double x) throws InappropriateFunctionPointException {
    FunctionNode node = getNodeByIndex(index);
```

```
    double newX = x;

    // Проверка порядка точек
    if (pointsCount == 1) {
        node.point.setX(x);
        return;
    }

    if (index == 0) {
        if (newX >= node.next.point.getX()) {
            throw new InappropriateFunctionPointException("Новая X для первой точки
должна быть меньше следующей точки");
        }
    } else if (index == pointsCount - 1) {
        if (newX <= node.prev.point.getX()) {
            throw new InappropriateFunctionPointException("Новая X для последней точки
должна быть больше предыдущей точки");
        }
    } else {
        if (newX <= node.prev.point.getX() || newX >= node.next.point.getX()) {
            throw new InappropriateFunctionPointException("Новая X должна быть между
соседними точками");
        }
    }

    node.point.setX(x);
}

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

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

public void deletePoint(int index) {
    if (pointsCount < 3) {
        throw new IllegalStateException("Невозможно удалить точку: количество точек
меньше трех");
    }
    deleteNodeByIndex(index);
}

public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
```

```
    // Проверка на существование точки с таким X
    FunctionNode current = head.next;
    while (current != head) {
        if (Math.abs(current.point.getX() - point.getX()) < 1e-10) {
            throw new InappropriateFunctionPointException("Точка с такой X уже
существует");
        }
        current = current.next;
    }

    // Поиск позиции для вставки
    int insertIndex = 0;
    current = head.next;
    while (current != head && current.point.getX() < point.getX()) {
        current = current.next;
        insertIndex++;
    }
    // Используем готовый метод для вставки
    FunctionNode newNode = addNodeByIndex(insertIndex);
    newNode.point = new FunctionPoint(point);
}

public void printTabulatedFunction() {
    FunctionNode current = head.next;
    int index = 0;
    while (current != head) {
        System.out.println(index + ": x = " + current.point.getX() + ", y = " +
current.point.getY());
        current = current.next;
        index++;
    }
}
```

Задание 6

Класс TabulatedFunction переименовала в класс ArrayTabulatedFunction. Создала интерфейс TabulatedFunction, содержащий объявления общих методов классов ArrayTabulatedFunction и LinkedListTabulatedFunction.

```
package functions;

public interface TabulatedFunction {
    double getLeftDomainBorder();
    double getRightDomainBorder();
    double getFunctionValue(double x);
    int getPointsCount();
```

```
    FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException;

    void setPoint(int index, FunctionPoint point)
            throws FunctionPointIndexOutOfBoundsException,
InappropriateFunctionPointException;

    double getPointX(int index) throws FunctionPointIndexOutOfBoundsException;

    void setPointX(int index, double x)
            throws FunctionPointIndexOutOfBoundsException,
InappropriateFunctionPointException;

    double getPointY(int index) throws FunctionPointIndexOutOfBoundsException;

    void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;

    void deletePoint(int index)
            throws FunctionPointIndexOutOfBoundsException, IllegalStateException;

    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;

    void printTabulatedFunction();
}
```

Сделала так, чтобы оба класса функций реализовывали созданный интерфейс:

```
public class ArrayTabulatedFunction implements TabulatedFunction
public class LinkedListTabulatedFunction implements TabulatedFunction
```

Теперь суть работы с табулированными функциями заключена в типе интерфейса, а в классах заключена только реализация этой работы.

Задание 7

Проверила работу написанных классов.

Для этого в созданном ранее классе Main, содержащем точку входа программы, добавила проверку для случаев, в которых объект табулированной функции должен выбрасывать исключения. Ссылочную переменную для работы с объектом функции объявила типа TabulatedFunction, а при создании объекта указала реальный класс. Тогда проверка работы класса LinkedListTabulatedFunction может быть произведена путем простой замены класса, объект которого создается. Результат – код main

```
import functions.*;

public class Main {
    public static void main(String[] args) {
        // Для тестирования каждой из реализаций можно закомментировать одну из
следующих 2 строк
        TabulatedFunction cubic = new ArrayTabulatedFunction(-2.0, 2.0, 5);
        // TabulatedFunction cubic = new LinkedListTabulatedFunction(-2.0, 2.0, 5);

        testBasicFunctionality(cubic, "ArrayTabulatedFunction");
        testExceptions();
    }

    private static void testBasicFunctionality(TabulatedFunction function, String
className) {
        try {
            System.out.println("Тестирование: " + className);
            // Заполняем значениями y = x^3
            for (int i = 0; i < function.getPointsCount(); i++) {
                double x = function.getPointX(i);
                function.setPointY(i, x * x * x);
            }
            // Выводим исходные точки
            System.out.println("Исходные точки:");
            function.printTabulatedFunction();
            // Тестируем интерполяцию
            double[] testPoints = {-3.0, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 3.0};
            System.out.println("Тест интерполяции:");
            for (double x : testPoints) {
                double value = function.getFunctionValue(x);
                if (Double.isNaN(value)) {
                    System.out.println("не определено " + x);
                } else {
                    System.out.println(x + "=" + value + " ожидалось = " + x*x*x);
                }
            }
            // Меняем одну из точек
            FunctionPoint newPoint = new FunctionPoint(-1.5, -3.375);
            function.setPoint(1, newPoint);
            System.out.println("После замены точки:");
            function.printTabulatedFunction();
            // Добавляем новую точку
            FunctionPoint addedPoint = new FunctionPoint(1.7, 4.913);
            function.addPoint(addedPoint);
            System.out.println("После добавления точки:");
```

```
            function.printTabulatedFunction();
            // Удаляем точку
            function.deletePoint(3);
            System.out.println("После удаления точки:");
            function.printTabulatedFunction();

        } catch (Exception e) {
            System.out.println("Ошибка при тестировании " + className + ": " +
e.getMessage());
        }
    }

    private static void testExceptions() {
        // Тестируем все требуемые исключения
        try {
            TabulatedFunction func = new ArrayTabulatedFunction(0.0, 10.0, 3);

            // FunctionPointIndexOutOfBoundsException
            try {
                func.getPoint(10);
            } catch (FunctionPointIndexOutOfBoundsException e) {
                System.out.println("FunctionPointIndexOutOfBoundsException: " +
e.getMessage());
            }

            // InappropriateFunctionPointException - нарушение порядка
            try {
                func.setPointX(1, 10.0);
            } catch (InappropriateFunctionPointException e) {
                System.out.println("InappropriateFunctionPointException: " + e.getMessage());
            }

            // InappropriateFunctionPointException - дублирование X
            try {
                func.addPoint(new FunctionPoint(5.0, 100.0));
            } catch (InappropriateFunctionPointException e) {
                System.out.println("InappropriateFunctionPointException: " + e.getMessage());
            }

            // IllegalStateException - удаление при малом количестве
            try {
                TabulatedFunction smallFunc = new ArrayTabulatedFunction(0.0, 2.0, 2);
                smallFunc.deletePoint(0);
            } catch (IllegalStateException e) {
                System.out.println("IllegalStateException: " + e.getMessage());
```

```
            }

            // IllegalArgumentException - некорректные параметры конструктора
            try {
                new ArrayTabulatedFunction(10.0, 0.0, 5);
            } catch (IllegalArgumentException e) {
                System.out.println("IllegalArgumentException: " + e.getMessage());
            }

        } catch (Exception e) {
            System.out.println("Неожиданная ошибка: " + e.getMessage());
        }
    }
}
```

Вывод при тестировании ArrayTabulatedFunction

```
Тестирование: ArrayTabulatedFunction

Исходные точки:

x = -2.0, y = -8.0

x = -1.0, y = -1.0

x = 0.0, y = 0.0

x = 1.0, y = 1.0

x = 2.0, y = 8.0

Тест интерполяции:

не определено -3.0

-2.0=-8.0 ожидалось = -8.0

-1.5=-4.5 ожидалось = -3.375

-1.0=-1.0 ожидалось = -1.0

-0.5=-0.5 ожидалось = -0.125

0.0=0.0 ожидалось = 0.0

0.5=0.5 ожидалось = 0.125

1.0=1.0 ожидалось = 1.0

1.5=4.5 ожидалось = 3.375

2.0=8.0 ожидалось = 8.0

не определено 3.0

После замены точки:
```

x = -2.0, y = -8.0

x = -1.5, y = -3.375

x = 0.0, y = 0.0

x = 1.0, y = 1.0

x = 2.0, y = 8.0

После добавления точки:

x = -2.0, y = -8.0

x = -1.5, y = -3.375

x = 0.0, y = 0.0

x = 1.0, y = 1.0

x = 1.7, y = 4.913

x = 2.0, y = 8.0

После удаления точки:

x = -2.0, y = -8.0

x = -1.5, y = -3.375

x = 0.0, y = 0.0

x = 1.7, y = 4.913

x = 2.0, y = 8.0

FunctionPointIndexOutOfBoundsException: Индекс выходит за границы: 10

InappropriateFunctionPointException: Новая X для последней точки должна быть больше предыдущей точки

InappropriateFunctionPointException: Точка с такой X уже существует

IllegalStateException: Невозможно удалить точку: количество точек меньше трех

IllegalArgumentException: Левая граница области определения больше или равна правой
Вывод при тестировании LinkedListTabulatedFunction точно такой же