Отчет по лабораторной работе № 3

Выполнил: Латахин Егор Дмитриевич - 6201-120303D

Специальность: Фотоника и оптоинформатика

Задание 1: Изучение стандартных исключений

Были изучены следующие классы исключений:

Исключение	Назначение
Exception	Базовый класс для всех исключений, которые могут быть восстановлены (checked exceptions).
IndexOutOfBoundsException	Исключение времени выполнения (unchecked), выбрасываемое при обращении к индексу, который находится за пределами допустимого диапазона
ArrayIndexOutOfBoundsException	Подкласс IndexOutOfBoundsException для массивов.
IllegalArgumentException	Исключение выбрасываемое, чтобы указать, что метод получил недопустимый аргумент.
IllegalStateException	Исключение времени выполнения, сигнализирующее о том, что объект находится в ненадлежащем (“незаконном”) состоянии для операции.

Задание 2: Создание пользовательских исключений

В пакете functions были созданы два новых класса исключений:

1. **FunctionPointIndexOutOfBoundsException**: Для ошибок, связанных с выходом индекса точки за границы набора.
    - Наследует от IndexOutOfBoundsException.

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends
IndexOutOfBoundsException {

    public FunctionPointIndexOutOfBoundsException() {
        super();
    }

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

2. **InappropriateFunctionPointException**: Для ошибок, связанных с нарушением порядка или уникальности абсцисс при добавлении/изменении точек.

- Наследует от Exception (является проверяемым исключением, требующим обработки try-catch или объявления throws).

```java
package functions;

public class InappropriateFunctionPointException extends Exception {

    public InappropriateFunctionPointException() {
        super();
    }

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

Задание 3: Внесение изменений в класс ArrayTabulatedFunction

Класс TabulatedFunction был переименован в ArrayTabulatedFunction. В него были добавлены проверки и механизмы выбрасывания исключений согласно требованиям:

1. **Конструкторы:** Добавлена проверка на некорректные границы (leftX >= rightX) и недостаточное количество точек (count < 2 или values.length < 2), выбрасывая IllegalArgumentException.
2. **Методы доступа по индексу (getPoint, setPoint, getPointX, setPointX, getPointY, setPointY, deletePoint):** Перед выполнением операции вызывается приватный метод checkIndex(int index), который выбрасывает FunctionPointIndexOutOfBoundsException, если индекс выходит за границы [0, pointsCount - 1].
3. **Методы изменения X (setPoint, setPointX):** Проверяется, что новая координата X находится строго между соседними точками. Если условие нарушено (новая X ≤ X предыдущей или ≥ X следующей), выбрасывается InappropriateFunctionPointException.
4. **Метод addPoint:** Проверяется, что x добавляемой точки не совпадает с x существующей (с учетом эпсилона). При совпадении выбрасывается InappropriateFunctionPointException.
5. **Метод deletePoint:** Если количество точек ≤ 2, выбрасывается IllegalStateException, поскольку функция должна иметь не менее двух точек.

// Фрагмент ArrayTabulatedFunction.java с примерами исключений

// В конструкторе

```
    public ArrayTabulatedFunction(double leftX, double rightX, int count) {
        if(count<2){
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
```

// В методах по индексу

```
private void checkIndex(int index) throws FunctionPointIndexOutOfBoundsException {
        if (index < 0 || index >= pointsCount) {
            throw new FunctionPointIndexOutOfBoundsException("Индекс " + index + " вне диапазона [0, " +
(pointsCount - 1) + "]");
        }
    }
```

// В setPointX

```
@Override public void setPointX(int i, double x) throws FunctionPointIndexOutOfBoundsException,
InappropriateFunctionPointException {
        checkIndex(i);
        if(i>0&&x<=points[i-1].getX()){
            throw new InappropriateFunctionPointException("x должен быть не больше x следующей точки");
        }
        if(i<pointsCount-1&&x>=points[i+1].getX()) {
            throw new InappropriateFunctionPointException("x должен быть не меньше x следующей точки");
        }
        points[i].setX(x);
    }
```

// В deletePoint

```
@Override public void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException,
IllegalStateException {
        if (pointsCount<=2) {
            throw new IllegalStateException("Нельзя удалить точку: минимальное количество точек - 2");
        }
        checkIndex(index);

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

Задание 4: Реализация класса LinkedListTabulatedFunction (Связный список)

Был реализован класс LinkedListTabulatedFunction на основе двусвязного циклического списка с выделенной головой.

Описание и обоснование структуры

1. **Класс FunctionNode:**

```
    private static class FunctionNode {
        FunctionPoint data;
        FunctionNode prev;
        FunctionNode next;

        FunctionNode() {
            this.data = null;
            this.prev = this;
            this.next = this;
```

```
        }

        FunctionNode(FunctionPoint data) {
            this.data = data;
        }
    }
```

- **Место описания и видимость:** Описан как **приватный статический вложенный класс** внутри LinkedListTabulatedFunction.
- **Обоснование:** Элементы списка (FunctionNode) являются внутренней деталью реализации LinkedListTabulatedFunction и не должны быть доступны извне (инкапсуляция). Статический вложенный класс не имеет неявной ссылки на внешний объект, что более логично и эффективно, поскольку узлы не зависят от конкретного экземпляра списка, за исключением их связи.
- **Инкапсуляция:** Поля (data, prev, next) имеют **пакетную или дефолтную** видимость, поскольку к ним обращается только внешний класс LinkedListTabulatedFunction, который находится в том же пакете или может обращаться к ним напрямую как внешний класс.

2. **Класс LinkedListTabulatedFunction:**
    - Поля: head (ссылка на голову), pointsCount (количество точек).

Методы работы со списком

Реализованы основные методы для манипулирования двусвязным циклическим списком.

- **FunctionNode getNodeByIndex(int index):**
    - Реализована оптимизация доступа:
        - выбирается ближайший путь: от головы (head.next) или от хвоста (head.prev).
- **FunctionNode addNodeToTail():** Добавляет новый узел “перед головой” (в конец).
- **FunctionNode addNodeByIndex(int index):** Добавляет новый узел в index.
- **FunctionNode deleteNodeByIndex(int index):** Удаляет узел по индексу, перестраивая связи.

Задание 5: Реализация методов табулированной функции в LinkedListTabulatedFunction

В LinkedListTabulatedFunction реализованы все методы интерфейса TabulatedFunction с соблюдением требований по исключениям, аналогично ArrayTabulatedFunction.

Оптимизация

Методы, которые выигрывают от прямого доступа к структуре списка (вместо использования getNodeByIndex с его накладными расходами):

- **getFunctionValue(double x):** Вместо многократного вызова getNodeByIndex для поиска интервала, обход списка осуществляется напрямую от head.next, что является оптимальным для поиска по X.
- **addPoint(FunctionPoint point):** Поиск позиции для вставки по X осуществляется прямым обходом списка, пока не будет найдена позиция вставки (current.data.getX() > newX).

Фрагмент LinkedListTabulatedFunction.java (обход списка)

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

        // Оптимизированный поиск - проходим по списку напрямую
        FunctionNode current = head.next;

        // Ищем точку или интервал
        while (current.next != head) {
            if (Math.abs(x - current.data.getX()) < EPSILON) {
                return current.data.getY();
            }
            if (x < current.next.data.getX()) {
                break;
            }
            current = current.next;
        }

        // Проверка последней точки
        if (Math.abs(x - current.data.getX()) < EPSILON) {
            return current.data.getY();
        }

        // Линейная интерполяция
        double x1 = current.data.getX(), y1 = current.data.getY();
        double x2 = current.next.data.getX(), y2 = current.next.data.getY();

        return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
    }
```

Задание 6: Создание интерфейса TabulatedFunction

Был создан интерфейс TabulatedFunction, содержащий объявления всех общих методов:

```
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;
    double getPointY(int index) throws FunctionPointIndexOutOfBoundsException;

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

    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
    void deletePoint(int index) throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
}
```

Оба класса, **ArrayTabulatedFunction** (переименованный TabulatedFunction) и **LinkedListTabulatedFunction**, были модифицированы для реализации этого интерфейса:

public class ArrayTabulatedFunction implements TabulatedFunction { ... }

public class LinkedListTabulatedFunction implements TabulatedFunction { ... }

Задание 7: Проверка работы классов

Проверка работы реализована в классе Main. Тестирование производилось путем создания объекта типа TabulatedFunction с фактической реализацией ArrayTabulatedFunction и LinkedListTabulatedFunction поочередно. Проверены все случаи выбрасывания исключений.

Результаты консольного вывода

Тестирование ArrayTabulatedFunction

=== Тестирование ArrayTabulatedFunction ===

--- Тест 1: Некорректные параметры конструктора

OK: Левая граница должна быть меньше правой

OK: Количество точек должно быть не менее 2

Функция создана: x=[0,4], y=[0.0, 1.0, 4.0, 9.0, 16.0]

--- Тест 2: Выход за границы индекса

OK getPoint(-1): Индекс -1 вне диапазона [0, 4]

OK getPoint(10): Индекс 10 вне диапазона [0, 4]

--- Тест 3: Некорректное изменение X

OK setPointX(2, 0.5): x должен быть не больше x следующей точки

OK setPointX(2, 3.5): x должен быть не меньше x следующей точки

--- Тест 4: Добавление точки с существующим X

OK addPoint(2.0, 100): Точка с таким x уже существует

--- Тест 5: Удаление при минимальном количестве точек

Осталось точек: 2

OK deletePoint(0): Нельзя удалить точку: минимальное количество точек - 2

--- Тест 6: Нормальная работа

Границы: [0.0, 4.0]

f(1.5) = 2.5

f(2.0) = 4.0

Добавлена точка (1.5, 2.25)

Количество точек: 6

f(1.5) = 2.25

=== Тестирование LinkedListTabulatedFunction ===

--- Тест 1: Некорректные параметры конструктора

OK: Левая граница должна быть меньше правой

OK: Количество точек должно быть не менее 2

Функция создана: x=[0,4], y=[0.0, 1.0, 4.0, 9.0, 16.0]

--- Тест 2: Выход за границы индекса

OK getPoint(-1): Индекс -1 вне диапазона [0, 4]

OK getPoint(10): Индекс 10 вне диапазона [0, 4]

--- Тест 3: Некорректное изменение X

OK setPointX(2, 0.5): x должен быть больше x предыдущей точки

OK setPointX(2, 3.5): x должен быть меньше x следующей точки

--- Тест 4: Добавление точки с существующим X

OK addPoint(2.0, 100): Точка с таким x уже существует

--- Тест 5: Удаление при минимальном количестве точек

Осталось точек: 2

OK deletePoint(0): Нельзя удалить точку: минимальное количество точек - 2

--- Тест 6: Нормальная работа

Границы: [0.0, 4.0]

f(1.5) = 2.5

f(2.0) = 4.0

Добавлена точка (1.5, 2.25)

Количество точек: 6

f(1.5) = 2.25