Отчёт по лабораторной работе №3

Разработка классов для работы с табулированными функциями с обработкой исключений

По курсу
Объектно-ориентированное программирование

Выполнил: Акст Роман,
Студент группы 6203-010302D

Задание 1

Были изучены стандартные классы исключений Java:

- java.lang.Exception - базовый класс для проверяемых исключений
- java.lang.IndexOutOfBoundsException - выход за границы индекса
- java.lang.ArrayIndexOutOfBoundsException - специализированная версия для массивов
- java.lang.IllegalArgumentException - неверный аргумент
- java.lang.IllegalStateException - недопустимое состояние объекта

Задание 2

Созданы два класса исключений в пакете functions:

- **FunctionPointIndexOutOfBoundsException** - наследует от IndexOutOfBoundsException, используется при обращении к несуществующему индексу точки
- **InappropriateFunctionPointException** - наследует от Exception, используется при нарушении упорядоченности точек

На рисунках 1 и 2 представлена реализация классов исключений

```
package functions;

public class FunctionPointIndexOutOfBoundsException extends IndexOutOfBoundsException  43 usages  new *
{
    public FunctionPointIndexOutOfBoundsException()  7 usages  new *
    {
        super();
    }

    public FunctionPointIndexOutOfBoundsException(String message)  16 usages  new *
    {
        super(message);
    }
}
```

Рисунок 1

```
package functions;

public class InappropriateFunctionPointException extends Exception  33 usages  new *
{
    public InappropriateFunctionPointException()  8 usages  new *
    {
        super();
    }

    public InappropriateFunctionPointException(String message)  25 usages  new *
    {
        super(message);
    }
}
```

Рисунок 2

Задание 3

Класс TabulatedFunction был переименован в ArrayTabulatedFunction и дополнен обработкой исключений:

Конструкторы теперь выбрасывают IllegalArgumentException при:

- Левая граница ≥ правой границы
- Количество точек < 2

Методы доступа к точкам
выбрасывают FunctionPointIndexOutOfBoundsException при неверном индексе.

Методы модификации выбрасывают InappropriateFunctionPointException при нарушении упорядоченности точек.

Метод deletePoint() выбрасывает IllegalStateException при попытке удалить точку, если останется менее 3 точек.

На рисунках 3, 4, 5 представлены примеры, в которых происходит проверка с использованием исключений

```
// конструктор для границ координат и количества точек
public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount)  9 usages  new *
        throws IllegalArgumentException
{
    if (leftX >= rightX)
    {
        throw new IllegalArgumentException("Левая граница больше правой");
    }

    if (pointsCount < 2)
    {
        throw new IllegalArgumentException("Количество точек меньше двух");
    }

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

    double step = (rightX - leftX) / (pointsCount - 1);

    for(int i = 0; i < pointsCount; i++)
    {
        this.points[i] = new FunctionPoint( x: leftX + step * i,  y: 0);
    }
}
```

Рисунок 3

```java
// получение координаты X точки по индексу
public double getPointX(int index) throws FunctionPointIndexOutOfBoundsException  2 usages  new *
{
    if(index < 0 || index >= amountOfElements)
    {
        throw new FunctionPointIndexOutOfBoundsException("Индекс не входит в границы");
    }

    return this.points[index].getX();
}
```

Рисунок 4

```java
// замена значения X в точке с заданным индексом
public void setPointX(int index, double x)  no usages  new *
        throws InappropriateFunctionPointException, FunctionPointIndexOutOfBoundsException
{
    if(index < 0 || index >= this.amountOfElements)
        throw new FunctionPointIndexOutOfBoundsException("Индекс не входит в границы");

    // проверки из setPoint() адаптированные для setPointX()(для избежания избыточности кода)
    if (index == 0 && this.amountOfElements > 1 && x >= points[1].getX() - EPSILON)
    {
        throw new InappropriateFunctionPointException("Точка лежит вне интервала");
    }

    if (index == this.amountOfElements - 1 && x <= points[index - 1].getX() + EPSILON)
    {
        throw new InappropriateFunctionPointException("Точка лежит вне интервала");
    }

    if (index > 0 && index < this.amountOfElements - 1 &&
            (x <= points[index - 1].getX() + EPSILON || x >= points[index + 1].getX() - EPSILON))
    {
        throw new InappropriateFunctionPointException("Точка лежит вне интервала");
    }

    // если все проверки пройдены - меняем X
    this.points[index].setX(x);
}
```

Рисунок 5

Задание 4

Реализован класс LinkedListTabulatedFunction с использованием двусвязного циклического списка с выделенной головой.
Структура внутреннего класс FunctionNode пресдтавлен на рисунке 6

```
private static class FunctionNode  31 usages  new *
{
    FunctionPoint point;  3 usages
    FunctionNode prev;  3 usages
    FunctionNode next;  2 usages
```

Рисунок 6

Были реализованы методы работы со списком, далее они послужат вспомогательными для реализации методов, подобных методам из класса для работы с табулированной функцией с помощью массива точек

- **getNodeByIndex()** - оптимизированный доступ к элементам (поиск с начала или конца в зависимости от индекса)
- **addNodeToTail()** - добавление элемента в конец списка
- **addNodeByIndex()** - добавление элемента по индексу
- **deleteNodeByIndex()** - удаление элемента по индексу

Реализация методов представлена на рисунках 7, 8, 9

```java
private FunctionNode getNodeByIndex(int index) throws FunctionPointIndexOutOfBoundsException  16 usages  new *
{
    if(index < 0 || index >= count)
    {
        throw new FunctionPointIndexOutOfBoundsException("Индекс не входит в границы");
    }

    FunctionNode currentNode;

    if(index <= count / 2)
    {
        currentNode = head.getNext();

        for(int i = 0; i < index; i++)
        {
            currentNode = currentNode.getNext();
        }
    } else
    {
        currentNode = head.getPrev();

        for(int i = count - 1; i > index; i--)
        {
            currentNode = currentNode.getPrev();
        }
    }

    return currentNode;
}

private FunctionNode addNodeToTail(FunctionPoint point)  3 usages  new *
{
    FunctionNode insertionNode = new FunctionNode(point);

    head.getPrev().setNext(insertionNode);
    insertionNode.setPrev(head.getPrev());

    insertionNode.setNext(head);
    head.setPrev(insertionNode);

    count++;

    return insertionNode;
}
```

Рисунок 7

```java
private FunctionNode addNodeByIndex(int index, FunctionPoint point)  1 usage  new *
        throws FunctionPointIndexOutOfBoundsException
{
    if(index < 0 || index > count)
    {
        throw new FunctionPointIndexOutOfBoundsException("Индекс не входит в границы");
    }

    // если индекс точки соответствует количеству точек добавляем новую точку в конец
    if(index == count)
    {
        return addNodeToTail(point);
    }

    // создаем новый узел и связываем его с соседними точками
    FunctionNode insertionNode = new FunctionNode(point);
    FunctionNode displaceableNode = getNodeByIndex(index);

    insertionNode.setPrev(displaceableNode.prev);
    insertionNode.setNext(displaceableNode);

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

    count++;

    return insertionNode;
}
```

Рисунок 8

```java
private FunctionNode deleteNodeByIndex(int index)  1 usage  new *
        throws IllegalStateException, FunctionPointIndexOutOfBoundsException
{
    // проверка на количество точек
    if (count < 3)
    {
        throw new IllegalStateException("Нельзя удалить точку: минимальное количество точек - 3");
    }

    // проверка на индекс
    if (index < 0 || index >= count)
    {
        throw new FunctionPointIndexOutOfBoundsException("Индекс не входит в границы");
    }

    FunctionNode targetNode = getNodeByIndex(index);

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

    count--;

    return targetNode;
}
```

Рисунок 9

Задание 5

Класс LinkedListTabulatedFunction реализует те же методы, что и ArrayTabulatedFunction, но с использованием связного списка. Все методы выбрасывают соответствующие исключения в тех же случаях.

Особенность реализации - оптимизация доступа к элементам через метод getNodeByIndex(), который выбирает направление обхода в зависимости от позиции искомого элемента.

На рисунке 10 представлен пример реализации метода getFunctionValue из ArrayTabulatedFunction

```
public double getFunctionValue(double x)  5 usages  new *
{
    if(count == 0)
    {
        return Double.NaN;
    }

    // вывод в случае выхода за границы
    if(x < head.getNext().getPoint().getX() || x > head.getPrev().getPoint().getX())
    {
        return Double.NaN;
    }

    FunctionNode current = head.getNext();

    while (current.getNext() != head)
    {
        // используем две точки для проверки вхождения в промежуток
        double x1 = current.getPoint().getX();
        double x2 = current.getNext().getPoint().getX();

        if (Math.abs(x - x1) < EPSILON) return current.getPoint().getY();
        if (Math.abs(x - x2) < EPSILON) return current.getNext().getPoint().getY();

        // если необходимая точка находится между имеющимися, то ищем промежуточное значение
        if (x > x1 && x < x2)
        {
            double y1 = current.getPoint().getY();
            double y2 = current.getNext().getPoint().getY();
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }

        current = current.getNext();
    }

    return Double.NaN;
}
```

Рисунок 10

Задание 6

Создан интерфейс TabulatedFunction, содержащий объявления всех общих методов. Оба класса (ArrayTabulatedFunction и LinkedListTabulatedFunction) реализуют этот интерфейс. Также были добавлены описания того, что должен реализовать каждый метод класса, наследующего интерфейс, пример показан на рисунке 11

```
/**
 * Получает начальную границу интервала определения функции
 * @return координата X левой границы области определения
 */
double getLeftDomainBorder();  3 usages  2 implementations  new *

/**
 * Получает конечную границу интервала определения функции
 * @return координата X правой границы области определения
 */
double getRightDomainBorder();  3 usages  2 implementations  new *

/**
 * Вычисляет приближенное значение функции в указанной точке
 * Использует линейную интерполяцию между соседними точками таблицы
 * @param x координата, для которой вычисляется значение функции
 * @return значение функции в точке x или Double.NaN, если x за границами определения
 */
double getFunctionValue(double x);  5 usages  2 implementations  new *

/**
 * Возвращает общее количество точек в таблице функции
 * @return число точек табуляции
 */
int getPointsCount();  3 usages  2 implementations  new *
```

Рисунок 11

Задание 7

Создан класс Main для тестирования всех возможностей системы:

Тестирование базовой функциональности:

- Создание функций через оба конструктора
- Добавление, удаление и изменение точек
- Линейная интерполяция значений
- Работа с границами области определения

Тестирование исключений:

- Некорректные параметры конструкторов
- Выход за границы индексов
- Нарушение упорядоченности точек
- Попытка удаления при недостаточном количестве точек

Сравнение реализаций:

- Одинаковое поведение для одинаковых входных данных
- Корректность интерполяции в обеих реализациях

Пример вывода тестовой программы приведен на рисунках 12, 13(также в связи с особенностями вычислений для чисел с плавающей запятой мы видим небольшие погрешности вычислений)

```
ТЕСТИРОВАНИЕ ТАБУЛИРОВАННЫХ ФУНКЦИЙ
Квадратичная функция y = x²
Исходная функция:
  Точка 0: x=0.0, y=0.0
  Точка 1: x=1.0, y=1.0
  Точка 2: x=2.0, y=4.0
  Точка 3: x=3.0, y=9.0
  Точка 4: x=4.0, y=16.0
Всего точек: 5

Добавление точки (2.5, 6.25)
После добавления:
  Точка 0: x=0.0, y=0.0
  Точка 1: x=1.0, y=1.0
  Точка 2: x=2.0, y=4.0
  Точка 3: x=2.5, y=6.25
  Точка 4: x=3.0, y=9.0
  Точка 5: x=4.0, y=16.0
Всего точек: 6

Удаление точки с индексом 2
После удаления:
  Точка 0: x=0.0, y=0.0
  Точка 1: x=1.0, y=1.0
  Точка 2: x=2.5, y=6.25
  Точка 3: x=3.0, y=9.0
  Точка 4: x=4.0, y=16.0
Всего точек: 5

Изменение точки с индексом 1
После изменения:
  Точка 0: x=0.0, y=0.0
  Точка 1: x=1.5, y=2.25
  Точка 2: x=2.5, y=6.25
  Точка 3: x=3.0, y=9.0
  Точка 4: x=4.0, y=16.0
Всего точек: 5

Интерполяция значений
  f(0.5) = 0.75
  f(1.2) = 1.7999999999999998
  f(2.8) = 7.899999999999999
  f(3.5) = 12.5
  f(-1) = NaN (слева от области)
  f(5) = NaN (справа от области)
```

Рисунок 12

```
Границы области определения
Левая граница: 0.0
Правая граница: 4.0

ТЕСТИРОВАНИЕ ИСКЛЮЧЕНИЙ

Тест конструкторов:
ArrayTabulatedFunction:
  Поймано: Левая граница больше правой

LinkedListTabulatedFunction:
  Поймано LinkedList: Левая граница больше правой

Тест индексных ошибок:
  Поймано: Индекс не входит в границы

Тест упорядоченности:
  Поймано: X должен быть между соседними точками

Тест дублирования точек:
  Поймано: Точка с таким X уже существует

Тест удаления:
  Поймано: Нельзя удалить точку: минимальное количество точек - 3

СРАВНЕНИЕ РЕАЛИЗАЦИЙ
ArrayTabulatedFunction:
  Точка 0: x=0.0, y=0.0
  Точка 1: x=1.0, y=1.0
  Точка 2: x=2.0, y=4.0
  Точка 3: x=3.0, y=9.0
  Точка 4: x=4.0, y=16.0
Всего точек: 5

LinkedListTabulatedFunction:
  Точка 0: x=0.0, y=0.0
  Точка 1: x=1.0, y=1.0
  Точка 2: x=2.0, y=4.0
  Точка 3: x=3.0, y=9.0
  Точка 4: x=4.0, y=16.0
Всего точек: 5

Сравнение интерполяции:
  f(0.5): array=0.5, list=0.5
  f(1.5): array=2.5, list=2.5
  f(2.5): array=6.5, list=6.5
  f(3.5): array=12.5, list=12.5
```

Рисунок 13