Вкладка 1

Отчет о лабораторной работе №3

Выполнил:
Петров Евгений Станиславович
Группа: 6204-010302D

Вкладка 2

1. Цель и исходные данные

Цель работы — расширить пакет для табулированных функций одной переменной: изучить используемые исключения, добавить собственные классы ошибок, доработать интерфейс и существующую реализацию, а также создать альтернативную реализацию на связном списке.

Исходным кодом выступал проект, где функция представлена набором объектов `FunctionPoint`. Использование пакета `java.util` запрещено, поэтому все структуры данных и проверки реализованы вручную.

---

2. Ход выполнения заданий

2.1. Задание 1. Анализ стандартных исключений Java

Изучены пять базовых классов исключений:

- `Exception`
- `IndexOutOfBoundsException`
- `ArrayIndexOutOfBoundsException`
- `IllegalArgumentException`
- `IllegalStateException`

Проанализированы их назначение и типичные сценарии

использования. Особое внимание уделено:

- **ошибкам неверных аргументов (`IllegalArgumentException`);**
- **ошибкам состояния объекта (`IllegalStateException`);**
- **попыткам выхода за пределы коллекций (`IndexOutOfBoundsException` и его наследникам).**

Эти сведения использованы при проектировании собственной иерархии ошибок.

2.2. Задание 2. Проектирование пользовательских исключений

В пакет `functions` добавлены два класса:

• FunctionPointIndexOutOfBoundsException

Наследуется от `IndexOutOfBoundsException`.
Используется при обращении к точке по неправильному индексу.
Добавлен удобный конструктор с формированием понятного сообщения.

• InappropriateFunctionPointException

Наследуется от `Exception`.
Сигнализирует о нарушении правил упорядоченности точек (пересечение X-координат, разрыв монотонности).

Создание отдельных типов позволило четко разделить *ошибки доступа* и *ошибки структуры данных*.

Код классов:

```
public class FunctionPointIndexOutOfBoundsException
extends IndexOutOfBoundsException {
    public FunctionPointIndexOutOfBoundsException() {
}
    public
FunctionPointIndexOutOfBoundsException(String message)
{
        super(message);
    }
    public FunctionPointIndexOutOfBoundsException(int
index, int size) {
        super("Index " + index + " is out of bounds
for points count " + size);
    }
}

public class InappropriateFunctionPointException
extends Exception {
    public InappropriateFunctionPointException() { }
    public InappropriateFunctionPointException(String
message) {
        super(message);
    }
}
```

2.3. Задание 3. Изменения в TabulatedFunction и массивной реализации

Интерфейс `TabulatedFunction` дополнен:

- **константой `EPSILON`;**
- **уточнёнными описаниями методов;**
- **checked-исключениями при нарушении упорядоченности точек.**

В `ArrayTabulatedFunction` реализованы проверки:

- **validateBorders — проверка границ и количества точек (иначе `IllegalArgumentException`);**
- **методы получения и изменения точек (`getPoint`, `setPoint`, `getPointX`, `setPointX` и др.) используют `checkIndex` и при ошибке выбрасывают `FunctionPointIndexOutOfBoundsException`;**
- **ensureCorrectOrder проверяет соблюдение порядка X и при нарушении генерирует `InappropriateFunctionPointException`;**
- **addPoint обеспечивает уникальность абсцисс и вручную расширяет массив (без `java.util`);**
- **deletePoint запрещает уменьшение количества точек меньше двух (`IllegalStateException`).**

Пример фрагментов:

```
public interface TabulatedFunction {
    double EPSILON = 1e-9;

    FunctionPoint getPoint(int index);
```

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

    void addPoint(FunctionPoint point)
            throws
InappropriateFunctionPointException;
}

private void ensureCorrectOrder(double x, int index)
        throws InappropriateFunctionPointException {
    if (index > 0 && x <= points[index - 1].getX()) {
        throw new
InappropriateFunctionPointException("Point must keep
ascending order.");
    }
    if (index < size - 1 && x >= points[index +
1].getX()) {
        throw new
InappropriateFunctionPointException("Point must keep
ascending order.");
    }
}
```

2.4. Задание 4. Реализация LinkedListTabulatedFunction

Разработан класс `LinkedListTabulatedFunction`, использующий собственную двусвязную структуру `FunctionNode` и фиктивный узел `head`.

Реализовано:

- **конструкторы с равномерным распределением точек и проверкой корректности;**
- **оптимизированный линейный поиск узла по индексу (от головы или хвоста);**
- **методы доступа возвращают копии `FunctionPoint` для защиты инкапсуляции;**
- **линейная интерполяция в `getFunctionValue` (вне области — `NaN`);**
- **операции изменения структуры (`setPoint`, `setPointX`, `addPoint`), соблюдающие порядок X;**
- **`deletePoint` с проверкой минимального количества точек (`IllegalStateException`);**
- **все ошибки индексации — через `FunctionPointIndexOutOfBoundsException`.**

Пример:

```
private void ensureCorrectOrder(double x, FunctionNode
previous, FunctionNode next)
        throws InappropriateFunctionPointException {
    if (previous != head && x <=
previous.point.getX()) {
        throw new
```

```
InappropriateFunctionPointException("Point must keep
ascending order.");
    }
    if (next != head && x >= next.point.getX()) {
        throw new
InappropriateFunctionPointException("Point must keep
ascending order.");
    }
}

public void deletePoint(int index) {
    if (size <= 2) {
        throw new IllegalStateException("Tabulated
function must contain at least two points.");
    }
    deleteNodeByIndex(index);
}
```

---

3. Проверка и тестирование

Создан класс `Main` для ручного тестирования.

Проверка включает:

- **создание `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` на отрезке `[0; π]`;**
- **заполнение значениями `sin(x)`;**
- **операции изменения, добавления и удаления точек;**

- **метод `demonstrateExceptions`, вызывающий по очереди все типы ошибок**
  (неверный индекс, нарушение порядка, дубликат X, удаление точки при минимуме 2).

Все исключения успешно перехватываются, что подтверждает корректность системы проверок.

Пример тестового сценария:

```
private static void
demonstrateExceptions(TabulatedFunction function) {
    try {
        function.getPoint(-1);
    } catch (FunctionPointIndexOutOfBoundsException e)
{
        System.out.println("Caught expected index
error: " + e.getMessage());
    }
    try {
        function.setPointX(1, function.getPointX(0));
    } catch (InappropriateFunctionPointException e) {
        System.out.println("Caught expected ordering
error: " + e.getMessage());
    }
    try {
        function.addPoint(new
FunctionPoint(function.getPointX(1), 42.0));
    } catch (InappropriateFunctionPointException e) {
        System.out.println("Caught expected duplicate
point error: " + e.getMessage());
```

```
    }
    try {
        while (function.getPointsCount() > 2) {
            function.deletePoint(0);
        }
        function.deletePoint(0);
    } catch (IllegalStateException e) {
        System.out.println("Caught expected deletion
error: " + e.getMessage());
    }
}
```

---

4. Вывод

Работа выполнена полностью:

- **созданы собственные исключения;**
- **реализованы проверки в интерфейсе и массивной реализации;**
- **разработан связный список с ручным управлением узлами;**
- **подготовлены демонстрационные сценарии.**

Код компилируется и корректно обрабатывает все предусмотренные ошибки. Все требования задания выполнены.