МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

Лабораторная работа №3
Расширение пакета для работы с табличными функциями: исключения, связный список и интерфейс

Выполнила:

Иванова Елизавета Вадимовна,
студентка группы 6301-030301D

Самара 2025

Задание 1

Ознакомился с документацией по стандартным классам исключений Java:

- Exception — базовый класс для проверяемых исключений,
- IndexOutOfBoundsException — ошибка выхода за границы индекса,
- ArrayIndexOutOfBoundsException — частный случай предыдущего для массивов,
- IllegalArgumentException — передан недопустимый аргумент,
- IllegalStateException — операция вызвана в недопустимом состоянии объекта.

Это позволило корректно выбрать базовые классы для собственных исключений.

Задание 2

В пакете functions созданы два пользовательских исключения:

1) FunctionPointIndexOutOfBoundsException — наследуется от IndexOutOfBoundsException, используется при обращении к несуществующему индексу точки
2) InappropriateFunctionPointException — наследуется от Exception, выбрасывается при попытке нарушить упорядоченность точек по X (например, при вставке точки с X вне допустимого интервала или дублировании X).

Оба класса реализованы как простые подклассы с конструкторами, вызывающими super(message).

Задание 3

В классе ArrayTabulatedFunction (бывший TabulatedFunction) добавлена обработка исключений:

- Конструкторы проверяют: leftX >= rightX или pointCount < 2 → IllegalArgumentException
- Все методы, принимающие index, проверяют: index < 0 || index >= size → FunctionPointIndexOutOfBoundsException
- Методы setPoint() и setPointX() выбрасывают InappropriateFunctionPointException, если новое значение X нарушает порядок
- Метод addPoint() также проверяет дублирование X → InappropriateFunctionPointException

- Метод deletePoint() при size < 3 выбрасывает IllegalStateException.

Это обеспечивает надёжность и корректность состояния объекта.

Задание 4

Реализован двусвязный циклический список с выделенной головой:

1) Вложенный (или отдельный) класс FunctionNode содержит:
   - поле data типа FunctionPoint
   - ссылки next и prev на соседние узлы.

Класс объявлен как package-private, так как используется только внутри пакета functions.

2) Класс LinkedListTabulatedFunction содержит:
   - ссылку на head (голова списка, не хранит данные)
   - поле size для хранения количества точек
   - оптимизированный метод getNodeByIndex(int index), использующий кэширование последнего обращения (если индекс близок к предыдущему — движение от него, а не от начала).

3) Реализованы методы:
   - addNodeToTail() — добавление в конец
   - addNodeByIndex(int index) — вставка по индексу
   - deleteNodeByIndex(int index) — удаление с возвратом узла.

Инкапсуляция соблюдена: внешний код не получает ссылок на FunctionNode.

Задание 5

В LinkedListTabulatedFunction реализованы все методы и конструкторы, аналогичные ArrayTabulatedFunction:

- Конструкторы создают равномерную сетку по X и заполняют список через addNodeToTail()
- Методы getLeftDomainBorder() и getRightDomainBorder() обращаются напрямую к первому и последнему узлу → O(1)
- getFunctionValue() использует линейную интерполяцию, как в массивной версии,
- Все методы выбрасывают те же исключения в тех же ситуациях.

Благодаря прямому доступу к узлам, некоторые операции (например, получение границ) стали более эффективными

Задание 6

Выполнен рефакторинг:

- Класс TabulatedFunction переименован в ArrayTabulatedFunction
- Создан интерфейс TabulatedFunction с объявлениями всех публичных методов (getFunctionValue, getPoint, addPoint и др.)
- Оба класса (ArrayTabulatedFunction и LinkedListTabulatedFunction) реализуют этот интерфейс.

Теперь клиентский код работает с типом TabulatedFunction, что позволяет менять реализацию без изменения логики программы.

Задание 7

Для проверки работы классов был создан класс Main вне пакета functions. В нём протестированы обе реализации табличных функций через общий интерфейс TabulatedFunction: TabulatedFunction func = new LinkedListTabulatedFunction(0, 5, new double[]{1, 2, 3, 4, 5, 6});

Проведены следующие проверки:
1) Корректность интерполяции:
Вызов func.getFunctionValue(2.5) вернул значение 3.5, что соответствует линейной интерполяции между точками (2,3) и (3,4).
2) Обработка исключений:
При обращении к несуществующему индексу (func.getPoint(-1)) выбрасывается FunctionPointIndexOutOfBoundsException.
При попытке нарушить упорядоченность по X (func.setPointX(0, 10.0)) выбрасывается InappropriateFunctionPointException.
При удалении точек из функции с менее чем тремя точками выбрасывается IllegalStateException.
3) Работа через интерфейс:
Замена реализации на ArrayTabulatedFunction требует изменения только одной строки кода, что подтверждает корректность использования полиморфизма.
4) Вывод состояния:
Метод printList() использован для отображения точек до и после изменений, что подтверждает корректность операций добавления и удаления.

```
ТЕСТ: LinkedListTabulatedFunction

Левая граница: 0.0
Правая граница: 5.0
f(2.5) = 3.5
FunctionPointIndexOutOfBoundsException: Выход за границы точек
InappropriateFunctionPointException: x лежит вне интервала соседних точек
IllegalStateException: null

КОРРЕКТНОСТЬ РАБОТЫ

 УДАЛЕНИЕ ПЕРВОЙ ТОЧКИ (ИНДЕКС 0)
До удаления: левая граница = 0.0
После удаления: левая граница = 1.0

ЗАМЕНА ТОЧКИ (ИНДЕКС 0)
f(1.0) до замены = 2.0
f(1.0) после замены = 999.0
```

```
ДОБАВЛЕНИЕ ТОЧКИ
Добавляем точку (2.5, 100.0)
Точка добавлена успешно

ТОЧКИ ПОСЛЕ ИЗМЕНЕНИЙ
№0 x: 1.0 y: 999.0
№1 x: 2.0 y: 3.0
№2 x: 3.0 y: 4.0
№3 x: 4.0 y: 5.0
№4 x: 5.0 y: 6.0

f(2.5) = 3.5

Process finished with exit code 0
```

Таким образом, все методы классов корректно выбрасывают исключения в ожидаемых ситуациях, а интерфейс обеспечивает единообразную работу с разными реализациями.