Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №3

Дисциплина: «ООП»

Тема «Дополнить пакет для работы с функциями одной переменной, заданными в табличной форме, добавив классы исключений, новый класс функций и базовый интерфейс.»

Выполнил: Зорин Дмитрий Сергеевич

Группа: 6201-120303D

Самара, 2025

Задание на лабораторную работу

Задание 1

Ознакомиться (изучить документацию) со следующими классами исключений, входящих в API Java:
• java.lang.Exception • java.lang.IndexOutOfBoundsException • java.lang.ArrayIndex OutOfBoundsException • java.lang.IllegalArgumentException • java.lang.IllegalStateE xception

Я ознакомился с перечисленными классами исключений.

Задание 2

В пакете functions создать 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 настоятельно рекомендуется использовать специализированные средства среды разработки.

Я создал в пакете functions 2 класса исключений:

1. FunctionPointIndexOutOfBoundsException — наследуется от стандартного класса (Необходим, например, если индекс выходит за границы массива или списка точек).
2. InappropriateFunctionPointException — наследуется от класса Exception(Например, выбрасывает исключение при изменении точки функции неправильным образом).

Как итог, 2 класса исключений, которые используются в других классах для обработки ошибок.

Задание 3
В разработанный ранее класс TabulatedFunction внести изменения, обеспечивающие выбрасывание исключений методами класса.

Оба конструктора класса должны выбрасывать
исключение IllegalArgumentException, если левая граница области определения больше или равна правой, а также если предлагаемое количество точек меньше двух. Это обеспечит создание объекта только при корректных параметрах.

Методы getPoint(), setPoint(), getPointX(), setPointX(), getPointY(), setPointY() и d eletePoint() должны выбрасывать
исключение FunctionPointIndexOutOfBoundsException, если переданный в метод номер выходит за границы набора точек. Это обеспечит корректность обращений к точкам функции.

Методы setPoint() и setPointX() должны выбрасывать
исключение InappropriateFunctionPointException в том случае, если координата x задаваемой точки лежит вне интервала, определяемого значениями соседних точек табулированной функции. Метод addPoint() также должен выбрасывать исключение InappropriateFunctionPointException, если в наборе точек функции есть точка, абсцисса которой совпадает с абсциссой добавляемой точки. Это обеспечит сохранение упорядоченности точек функции.

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

Я внес в класс TabulatedFunction изменения, обеспечивающие выбрасывание исключений методами класса.
Конструкторы выбрасывают IllegalArgumentException, если левая граница >= правой или количество точек < 2. А методы доступа к точкам выбрасывают FunctionPointIndexOutOfBoundsException, если индекс выходит за границы. Это необходимо для безопасной работы массива точек. setPoint() и setPointX() выбрасывают InappropriateFunctionPointException, если новое значение x рушит порядок точек. Метод deletePoint() выбрасывает IllegalStateExceptio, если после удаления точек, их остается < 3.

Как итог, добавлены проверки и исключения для стабильной работы всех методов класса.

Задание 4

В пакете functions создать класс LinkedListTabulatedFunction, объект которого также должен описывать табулированную функцию. Отличие этого класса должно заключаться в том, что для хранения набора точек в нем должен использоваться не массив, а динамическая структура – связный список.

Важно понимать, что представляет собой связный список в объектно-ориентированном программировании. Действительно, в процедурных языках список обычно представляет собой набор записей (или структур) в памяти, некоторые элементы которых являются указателями на соседние элементы списка, а также набор процедур по работе со списком. В Java, во-первых, нет указателей и адресной арифметики, вместо этого будут использоваться ссылки. Во-вторых, вместо записей будут использоваться объекты. В-третьих, процедуры по работе со списком будут заменены на методы класса.

Особенность заключается в том, что для полноценного описания связного списка потребуется реализовать 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 должны присутствовать информационное поле и поля связи. А во втором классе должна храниться ссылка на объект головы списка и должны быть описаны методы для работы со списком, причем публичные методы не должны получать или возвращать ссылки на объекты элементов списка, т.к. это будет нарушением инкапсуляции. Можно сказать, что между этими классами имеется отношение композиции: объекты элементов списка являются частями объекта списка и не существуют вне его.

При написании «внешнего» класса следует учесть, что инкапсуляция работы со списком в отдельный объект позволяет несколько изменить алгоритмы по работе со списком по сравнению с обычными алгоритмами в процедурном исполнении. Так, не имеет смысла подсчитывать каждый раз длину списка

(ведь никто кроме самого объекта списка не может его изменить), вместо этого проще хранить ее как поле «внешнего» объекта. Также очевидно, что часто работа с элементами списка ведется последовательно или в соседних элементах, поэтому пробегать каждый раз от головы при доступе к элементу списка неразумно, проще хранить ссылку и номер элемента, к которому было последнее обращение, и в некоторых случаях двигаться от него.

В качестве структуры списка в настоящей работе требуется использовать двусвязный циклический список с выделенной головой. Первое означает, что объект элемента списка хранит две ссылки – на предыдущий элемент и на следующий элемент. Второе означает, что голова списка не используется для хранения данных и присутствует в списке всегда (пустой список представляет собой объект головы, ссылающийся сам на себя и как на предыдущий элемент, и как на следующий).

Класс LinkedListTabulatedFunction совмещает в себе две функции: с одной стороны, он будет описывать связный список и работу с ним, а с другой стороны, он будет описывать работу с табулированной функцией и ее точками. Для реализации первой функции требуется выполнить следующие действия.

1. Описать класс элементов списка FunctionNode, содержащий информационное поле для хранения данных типа FunctionPoint, а также поля для хранения ссылок на предыдущий и следующий элемент. Выберите и обоснуйте место описания класса и его видимость. Также выберите и обоснуйте реализацию инкапсуляции в этом классе.

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

3. В классе LinkedListTabulatedFunction реализовать 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) должна начинаться с 0. Метод должен обеспечивать оптимизацию доступа к элементам списка.

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

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

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

Я создал в пакете functions новый класс LinkedListTabulatedFunction для показа табулированной функции, где точки хранятся уже в двусвязном циклическом списке.

Было создано 2 класса FunctionNode и LinkedListTabulatedFunction. Было реализовано много методов, например, getNodeByIndex(int index) (возвращает ссылку на узел), addNodeToTail(), который добавляет элемент в конец списка, deleteNodeByIndex(int index), он удаляет элементы по индексу и другие.

Как итог, был создан класс табулированной функции, на основе двусвязного циклического списка.

Задание 5

Для обеспечения второй функции класса LinkedListTabulatedFunction реализовать в классе конструкторы и методы, аналогичные конструкторам и методам класса TabulatedFunction. Конструкторы должны иметь те же параметры, методы должны иметь те же сигнатуры. Также должны выбрасываться те же виды исключений в тех же случаях.
С одной стороны, при написании методов следует использовать уже написанные методы, отвечающие за работу со связным списком. С другой стороны, инкапсуляция в одном классе и списка, и табулированной функции позволяет в некоторых методах, относящихся к табулированной функции, значительно оптимизировать работу за счет возможности обращаться к элементам списка напрямую. Определите такие методы и оптимизируйте их.

Теперь мне необходимо было реализовать саму работу с табулированной функцией. Для этого я добавил конструкторы и методы, которые есть в классе ArrayTabulatedFunction. Например, по границам и количеству точек, по массиву Y и по массиву точек. Везде были добавлены проверки, выбрасывающие исключения: IllegalArgumentException, FunctionPointIndexOutOfBoundsException и другие. Для оптимизации некоторые методы я упростил.

Как итог, класс LinkedListTabulatedFunction полностью готов, как и ArrayTabulatedFunction. Отличие лишь в том, что в 1 случае для связного списка, во 2 для массива.

Задание 6

Класс TabulatedFunction переименовать в класс ArrayTabulatedFunction.

Создать интерфейс TabulatedFunction, содержащий объявления общих методов классов ArrayTabulatedFunction и LinkedListTabulatedFunction.

Сделать так, чтобы оба класса функций реализовывали созданный интерфейс.

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

В этом задании был реализован интерфейс TabulatedFunction, который описывает поведение всех табулированных функций. В самом интерфейсе, были объявлены методы для работы с функцией, например, getPointsCount(), void setPointY(int index, double y) и другие. Также чтоб использовать интерфейс TabulatetFunction, я использовал конструкции public class ArrayTabulatedFunction implements TabulatedFunction {} и public class LinkedListTabulatedFunction implements TabulatedFunction {}.

Как итог, был создан интерфейс для объединения всех методов.

Задание 7

Проверить работу написанных классов.

Для этого в созданном ранее классе Main, содержащем точку входа программы, добавить проверку для случаев, в которых объект табулированной функции должен выбрасывать исключения.

Ссылочную переменную для работы с объектом функции следует объявить типа TabulatedFunction, а при создании объекта следует указать реальный класс. Тогда проверка работы класса LinkedListTabulatedFunction может быть произведена путем простой замены класса, объект которого создается.

В 7 задании необходимо проверить работу программы и обработку исключений.

В классе Main я создал объект табулированной функции: TabulatedFunction func = new ArrayTabulatedFunction(0, 3, new double[]{0, 1, 4, 9}); (для массива) или TabulatedFunction func = new LinkedListTabulatedFunction(0, 3, new double[]{0, 1, 4, 9}); (для списка). Также были добавлены проверки, например, IllegalArgumentException, который выбрасывает исключение, если левая граница > правой и количество точек < 2 и другие.

Пример работы программы:

Поймано IllegalArgumentException: Левая граница должна быть < правой

Поймано IllegalArgumentException: Количество точек должно быть ≥ 2

Поймано InappropriateFunctionPointException: Точка с таким X уже существует

Поймано FunctionPointIndexOutOfBoundsException: Индекс 10 вне границ массива точек

Поймано IllegalStateException: Нельзя удалить точку, если их меньше трех

Как итог, программа полностью функционирует.