---

Линейная структура данных: Структура данных, в которой элементы данных расположены последовательно или линейно, где каждый элемент прикреплен к своему предыдущему и следующему соседним элементам, называется линейной структурой данных.

Примерами линейных структур данных являются массив, стек, очередь, связанный список.

Статическая структура данных: Статическая структура данных имеет фиксированный размер памяти.

Примером такой структуры данных является массив.

Динамическая структура данных: в динамической структуре данных размер не фиксирован. Он может произвольно обновляться во время выполнения, что можно считать эффективным с точки зрения сложности памяти (пространства) кода.

Примерами такой структуры данных являются очередь, стек и т. д.

---

Нелинейная структура данных: Структуры данных, в которых элементы данных расположены не последовательно или линейно, называются нелинейными структурами данных.

В нелинейной структуре данных мы не можем обойти все элементы только за один проход.

Примерами нелинейных структур данных являются деревья и графы.

---

Использование:

---

Массивы

Где угодно и для хранения чего угодно

нам необходимо хранить оценки всех студентов в классе, мы можем использовать массив для их хранения.

---

Связанный список

удобен когда важнее быстродействие операций вставки/удаления, которые в LinkedList выполняются за константное время.

Операции доступа по индексу производятся перебором с начала или конца (смотря что ближе) до нужного элемент

---

Стек

стеки используются для обслуживания вызовов методов (последний вызванный метод должен завершить выполнение первого) - рекурсия.

Стеки также используются для решения задач:

когда нам нужно перевернуть слово,

проверить наличие сбалансированных скобок,

в редакторах, где слово, которое вы набрали последним, удаляется первым при использовании операции отмены,

реализация функциональности возврата в веб-браузерах.

---

Очередь

представляет собой структуру данных, построенную в соответствии с очередями из нашей жизни:

очередь на остановке, где человек, стоящий в начале очереди, получает билет первым.

очередь в магазин

любая ситуация, когда ресурсы распределяются между несколькими пользователями и обслуживаются в порядке очереди.

планирование ЦП, планирование дискового пространства.

---

Деревья

Деревья кодирования Хаффмана используются в алгоритмах сжатия данных.

Деревья формируют иерархию. Они полезны в файловых структурах, где каждый файл расположен в определенном каталоге и существует определенная иерархия, связанная с файлами и каталогами.

---

Хеширование

в сбалансированном двоичном дереве поиска, если мы попытаемся выполнить поиск, вставить или удалить любой элемент, временная сложность будет O(logn).

когда наши приложения хотят выполнять те-же операции быстрее, то есть более оптимизированным способом - хеширование.

При хэшировании все вышеперечисленные операции могут быть выполнены за O(1), т.е. за постоянное время.

!!!Важно понимать, что в худшем случае временная сложность хеширования остается O(n), но в среднем временная сложность составляет O(1).

удаления дубликатов из набора элементов

частота элементов

в веб-браузерах мы можем проверять посещенные URL-адреса с помощью хеширования

Хеширование можно использовать в любой ситуации, когда требуется поиск(), вставка() и удаление() за время O(1).

---

List

List является интерфейсом, объекты типа list не могут быть созданы.

Нам всегда нужен класс, который реализует этот List, чтобы создать объект

после введения Generics в Java 1.5, можно ограничить тип объекта, который может храниться в списке.

---

практика

// Example1.java

public static void main(String[] args) {

List<Integer> list1 = new ArrayList<Integer>();

list1.add(0, 1);

list1.add(1, 2);

System.out.println(list1);

List<Integer> list2 = new ArrayList<Integer>();

list2.add(1);

list2.add(2);

list2.add(3);

list1.addAll(1, list2);

System.out.println(list1);

// Removes element from index 1

list1.remove(1);

System.out.println(list1);

System.out.println(list.get(3));

list1.set(0, 5);

System.out.println(list1);

}

Note -> код выше можно писать по мере обьяснения методов

---

Операция 1: добавление элементов в класс List с помощью метода add()

Чтобы добавить элемент в список, мы можем использовать метод add().

Этот метод перегружен для выполнения нескольких операций на основе разных параметров.

add(obj): этот метод используется для добавления элемента в конец !!! списка.

add(index, obj): этот метод используется для добавления элемента по определенному индексу !!! в список.

public static void main(String args[])

{

List<String> list = new ArrayList<>();

list.add("I");

list.add("am");

list.add("Java");

list.add(2, "study");

System.out.println(list);

}

---

Операция 2: Обновление элементов

если мы хотим изменить элемент, это можно сделать с помощью метода set().

список индексируется, на элемент, который мы хотим изменить, ссылается индекс элемента.

этот метод принимает индекс и обновленный элемент, который необходимо вставить по этому индексу.

public static void main(String args[])

{

List<String> list = new ArrayList<>();

list.add("I");

list.add("am");

list.add("Java");

list.add(2, "study");

System.out.println("Initial ArrayList " + list);

// set() method

list.set(2, "learning");

System.out.println("Updated ArrayList " + list);

}

---

Операция 3: Удаление элементов

Чтобы удалить элемент из списка, мы можем использовать метод remove().

Этот метод перегружен для выполнения нескольких операций на основе разных параметров.

remove(obj):

этот метод используется для простого удаления объекта из списка.

Если таких объектов несколько, первое вхождение объекта удаляется.

remove(index):

поскольку список индексируется, этот метод принимает целочисленное значение, которое просто удаляет элемент, присутствующий в этом конкретном индексе в списке.

После удаления элемента все элементы перемещаются влево, чтобы заполнить пространство, а индексы объектов обновляются.

public static void main(String args[])

{

List<String> list = new ArrayList<>();

list.add("I");

list.add("am");

list.add("learning")

list.add("Java");

System.out.println("Initial ArrayList " + list);

list.remove(1);

System.out.println("After the Index removal " + list);

list.remove("Java");

System.out.println("After the String removal " + list);

}

---

Итерация по списку

использование цикла for в сочетании с методом get()

public static void main(String args[])

{

List<String> list = new ArrayList<>();

list.add("I");

list.add("am");

list.add("learning")

list.add("Java");

// Using for loop for iteration

for (int i = 0; i < al.size(); i++) {

System.out.print(al.get(i) + " ");

}

}

и еще несколько интересных методов для работы со списком

size() - используется для возврата размера списка

clear() - используется для удаления всех элементов в списке. Однако ссылка на созданный список по-прежнему сохраняется.

indexOf(element) - возвращает последнее вхождение данного элемента или -1, если элемент отсутствует в списке.

isEmpty() - используется для проверки, является ли список пустым или нет. Возвращает true, если список пуст, иначе false.

contains(element) - используется для проверки, содержит ли список данный элемент или нет. Возвращает true, если список содержит элемент.

sort(Comparator) - используется для сортировки элементов списка на основе заданного компаратора.

---

Ассоциация с интерфейсом

AbstractList, CopyOnWriteArrayList и AbstractSequentialList — это классы, реализующие интерфейс List

AbstractList: этот класс используется для реализации немодифицируемого списка, для которого нужно только расширить этот класс AbstractList и реализовать только методы get() и size().

AbstractList — это абстрактный класс, поэтому ему следует назначить экземпляр его подклассов

// Example2.java

public static void main(String args[]) {

// Creating an empty AbstractList

AbstractList<String> list = new ArrayList<String>();

list.add("I");

list.add("am");

list.add("learning");

list.add("Java");

System.out.println("AbstractList:" + list);

}

CopyOnWriteArrayList: этот класс реализует интерфейс списка. Это расширенная версия ArrayList , в которой все модификации (добавление, установка, удаление и т. д.) осуществляются путем создания новой копии списка.

CopyOnWriteArrayList создает клонированную копию базового ArrayList, для каждой операции обновления в определенный момент оба будут автоматически синхронизированы, об этом заботится JVM.

нет никакого эффекта для потоков, выполняющих операцию чтения !!!

Его использование затратно, поскольку для каждой операции обновления будет создаваться клонированная копия.

CopyOnWriteArrayList — лучший выбор, если нашей частой операцией является операция чтения.

Это потокобезопасная версия ArrayList.

AbstractSequentialList: этот класс реализует интерфейс Collection и класс AbstractCollection. Этот класс используется для реализации немодифицируемого списка, для которого нужно только расширить этот класс AbstractList и реализовать только методы get() и size() .

ПРАКТИКА

1. Write a Java program to join two array lists.

// Example3.java

public static void main(String[] args) {

ArrayList<String> c1= new ArrayList<String>();

c1.add("Red");

c1.add("Green");

c1.add("Black");

c1.add("White");

c1.add("Pink");

System.out.println("List of first array: " + c1);

ArrayList<String> c2= new ArrayList<String>();

c2.add("Red");

c2.add("Green");

c2.add("Black");

c2.add("Pink");

System.out.println("List of second array: " + c2);

// Let join above two list

ArrayList<String> a = new ArrayList<String>();

a.addAll(c1);

a.addAll(c2);

System.out.println("New array: " + a);

}

2. Remove duplicates

// Example4.java

public static ArrayList<String> removeDuplicates(ArrayList<String> values) {

ArrayList<String> noDuplicates = new ArrayList<>();

for (String s : values) {

if (!noDuplicates.contains(s)) {

noDuplicates.add(s);

}

}

return noDuplicates;

}

ПАТТЕРНЫ

---

Шаблон №1

Возьмите список, измените все элементы в нем на новое значение

Распространенный сценарий:

1.вам дан список значений

2.вы должны применить определенную операцию для изменения каждого значения

3.вернуть список со всеми этими измененными значениями

Задачи:

1.у вас есть список строковых значений, и вы должны вернуть список со всеми этими строковыми значениями в нижнем регистре (или в верхнем регистре)

2.у вас есть список значений int и вы должны вернуть список, каждое значение которого умножается на 2

3.у вас есть список значений String , и вы должны вернуть список длины каждого из этих значений String .

два способа реализовать этот шаблон в Java:

1 - В этом случае создается новый список для измененных значений:

public static ArrayList<String> mapValues(ArrayList<String> oldValues) {

ArrayList<String> newValues = new ArrayList<String>();

for(String value : oldValues) {

String newValue = newValues.add(newValue):

}

return newValues;

}

2 - В этом случае значения меняются прямо в том же списке:

public static void mapValues(ArrayList<String> values) {

for(int i = 0; i < values.size(); i++) {

String newValue = newValues.set(i, newValue);

}

}

Шаблон №2

Возьмите список, удалите все элементы, которые не соответствуют определенным критериям.

Распространенный сценарий:

1.вам дан список значений

2.у вас есть критерии для этих значений

3.вернуть список только с теми значениями из исходного списка, которые соответствуют этим критериям

Задачи:

1.у вас есть список значений String и вы должны вернуть список со всеми значениями String определенной длины

2.у вас есть список значений int и вы должны вернуть список, содержащий только нечетные числа

3.у вас есть список значений String , и вы должны вернуть список этих значений без каких-либо дубликатов.

4.упражнение для списков массивов Удалить слишком короткие слова

как реализовать этот шаблон в Java:

public static ArrayList<String> filterValues(ArrayList<String> oldValues) {

ArrayList<String> newValues = new ArrayList<String>();

for(String value : oldValues) {

// here we define our condition based on value, e.g. value.length() > 10

if( ... ) {

newValues.add(value):

}

}

return newValues;

}

Шаблон №3

Возьмите список, верните одно значение

Распространенный сценарий:

1.вам дан список значений

2.вы берете все эти значения и вычисляете из них одно значение

3.вернуть это вычисленное значение

Задачи

1.у вас есть список значений String , и вы должны вернуть одну строку , которая представляет собой конкатенацию всех значений.

2.у вас есть список значений int и вы должны вернуть их сумму

3.у вас есть список значений int и вы должны вернуть их максимум или минимум

4.у вас есть список String и вы должны вернуть максимальную длину

как реализовать этот шаблон в Java:

public static int reduceValues(ArrayList<Integer> values) {

int result = 0;

for(int i = 0; i < oldValues.size(); i++) {

int value = oldValues.get(i);

// here we sum, but before each value is multiplied with its index

value = value \* i;

result = result + value;

}

return result;

}