}

Lesson-8-Array based List Implementation

Day-1

- Array list User Implementation
- ArrayList API
- Sorting – Collections.sort()
  - Comparator & Collections, consist with equals
- Searching- Collections
  - BinarySearch

Day-2

- Linked List user implementation (Double LL)
- LinkedList API
- Iterator and Iterable - Interfaces

ADT – Abstract Data Type → Abstract class/ Interface

```
// You could do the abstraction using Interface / Abstract class
Interface ListADT{
   Int size();
   Void add(String x);
}
Class MyList implements ListADT
```

Array User Implementation for the String type

```
class MyStringList{
// Instance fields
String strArray[];
int size;
Final int INI_SIZE = 5;
MyStringList(){
   strArray = new String[INI_SIZE];
   size = 0;
}
MyStringList(int capacity){
   strArray = new String[capacity];
   size = 0;
}
// Do the behaviours
void add(String x){} // Add in the end of the list
}
Void add(String x, int pos){}
int size(){ return size +1;}
String remove(String x) {}
String remove(int index){}
boolean contains(String x){}
void clear(){}
boolean isEmpty(){}
}
```

When you call the Constructor, it needs to do the necessary initialization.

strArray

Null	Null	Null	Null
[0]	[1]	[2]	[3]

Size = 0

If you call ob.add("Java") // End of the list

1. Make sure the input is null or not
2. Make sure is there a room to insert, if there is room add in the end, if there is no room call resize() method to make room and then add the element in the end of the list.
3. Add the element in to the collection using size index. strArray[size] = s;
4. Then increment the size by 1.

Example

strArray

Java	Null	Null	Null
[0]	[1]	[2]	[3]

Size = 1

strArray[size] = x;

```
size++;

strArray[size++] = x;

String Get(int i) {

 If(i<0 || i>size-1) return  null;

Return strArray[i];

}
```

1. Check the index I should be in the range of 0 to size-1 return the value in that index or else return null;

```
b if(s==null) return false;

                    for(String test : strArray){

                             if(test.equals(s)) return true;

              }

                   return false;

}

 boolean search(String x){

}
```

1. If x is null return false
2. Do the Linear Search from the index 0 to size -1, Once you get the exact match return true
3. Else X is not in the list, return false

strArray

size = 3

Java	C++	HTML	FPP
0	1	2	3

Ob.find(null) → return false

Ob.find(“C++”)→ true

Ob.find(“FPP”) → false

If you want to insert in the middle or beginning.

Insert(String x, int pos) // 0 to size-1

1. You have to check the pos should be in the range of 0 to size-1, you can insert or else Return nothing.
2. Check if there is a room insert x, or else call resize()
3. To insert the element in the specific position pos, you have to make a new array and then copy the elements from the 0 to pos from the original array into your new array. Then copy the new value x in the pos. Finally copy remaining elements from the original array to your new array.
4. Increment size by 1.

strArray

Java	C++	HTML	FPP	
0	1	2	3	4

Insert("Android", 2);

1. Copy to temp array before the position value
   System.arrayCopy(strArray,0,Temp,0,pos) // // src, spos, des, dspos, number of elements

System.arraycopy(strArray,0,temp,0,pos);

temp

Java	C++			
0	1	2	3	4

2. Insert the new value in the specific position
   Temp[pos] = x;

temp

Java	C++	Android		
0	1	2	3	4

3. Copy the remaining from position to the end into temp array.

   System.arraycopy(strArray,pos,temp,pos+1, strArray.length - pos);

temp

Java	C++	Android	HTML	FPP
0	1	2	3	4

4. Assign the temp reference to the original array. Old reference is for garbage.

strArray= temp;

size++;

Remove an element boolean remove(String x)

Note: In the remove logic, elements are not deleted, just decrement the size by 1. So that you cannot access beyond the size-1.

1. Check the list is empty or not.
2. Check the argument x is null or not.
3. Search the element is in the list, once you found shift down the elements then return true.
4. After removing, decrement the size by 1
5. Return false, the element is not found.

Original list

Java	C++	Android	HTML	FPP
0	1	2	3	4

Size = 5

After Removing remove("Android ")

Java	C++	HTML	FPP	FPP(not referred) Garbage/replaced by new value
0	1	2	3	4

Size=4

For example, remove "Android" from the index 2. Copy elements from index 0 and index 1 to temp array

Java	C++			
0	1	2	3	4

Again copy the elements from original array from index 2 to length-1 into Temp array.

Java	C++	HTML	FPP	
0	1	2	3	4

Resize()

1. Make a new array with double capacity of original.
2. Copy the elements from the original array to new array using either System.arraycopy(src, start, des, start, count) or Arrays.copyOf(original, new length)
3. Finally assign new array to original strArray

Java	C++	HTML	Swing

Temp = Arrays.copyof(strArray, newlength);

temp

Java	C++	HTML	Swing	Null	Null	Null	Null

strArray = temp;

```
// Remove an element from the specific position

public String remove(int index) {
   if (index < 0 || index >= size) {
          return null;
} // Get the element to be removed
  String removedElement = array[index];
// Shift elements to the left to fill the gap
   for (int i = index; i < size - 1; i++) {
         strArray[i] = strArray [i + 1]; }
// Decrease the size
       size--;
       return removedElement;
}
```

<u>Min Sort Procedure</u>

1. Identify the minimum element from the array.
2. Then swap the minimum element with the respective position.
3. Continue the same procedure until list get started.

Original Array with five inputs

Java	C++	Android	Html	VC++

Step 1:

Android	C++	Java	Html	VC++

Step 2:

Android	C++	Java	Html	VC++

Step 3:

Android	C++	Html	Java	VC++

Step 4:

Android	C++	Html	Java	VC++

Step 5:

Android	C++	Html	Java	VC++

Swap ()

X = “Java”

Y = “Android”

Temp = x;

X=y;

Y=temp;

Collection – Is an Interface

Collections → It is Utility class to help your collection framework.

<u>Difference between Comparable and Comparator</u>

- If any class implements comparable interface then collection of that object can be sorted automatically using Collection.sort() or Arrays.sort().Object will be sort on the basis of compareTo method in that class.
- Using Comparator interface, we can write different sorting based on different attributes of objects to be sorted. You can use anonymous comparator to compare at particular line of code or other class can implement this interface to sort.
  public void sort(Collection obj,Comparator c): is used to sort the elements of List by the given comparator.

What does Comparator consist with equals means in Java?

In Java, Comparator consistent with equals means that the comparison provided by the Comparator and the equality check provided by the equals() method give consistent results.

In other words: If two objects are considered equal by the equals() method, they should also be treated as equal by the Comparator (i.e., the compare() method should return 0).

```
public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass())
                 return false;
   Employee employee = (Employee) o;
   return Objects.*equals*(fname,
employee.fname)
&& Objects.*equals*(lname, employee.lname)
&& Objects.*equals*(hireDay,
employee.hireDay);
}
```
	```
	Collections.*sort*(elist,
	Comparator.*comparing*(Employee::getFname)
	        .thenComparing(Employee::getLname)
	.thenComparing(Employee::getHireDay));
	```

About Random Access

- The RandomAccess marker interface itself does not contain any methods because its sole purpose is to act as a flag.
- It tells the Java Collections Framework and developers that the implementing class supports efficient random access.
- This design follows a principle known as the **marker interface pattern**, which is used to indicate a particular property or behavior without defining any methods.
- In ArrayList, direct access to an element at a given index is achieved through the array's ability to compute the memory address of any element directly. [Elements are stored in the consecutive memory location]. This avoids the need for traversal.

- Simple way for better understanding,

  memory_address = base_address + (i * element_size);

- Refer ArrayList API source code to know the implementation

```
// Using toArray()
String[] array1 = list2.toArray(new String[0]);
```

- **Step 1:** `new String[0]` creates an empty array of type `String[]`.
- **Step 2:** `arrayList.toArray(new String[0])` calls the `toArray` method on the `arrayList`:
  - The method checks the size of the provided array (which is 0 in this case).
  - Since the size of the provided array is not sufficient to hold all elements of the `arrayList`, the method internally creates a new array of the required size (3 in this case) and the same type (`String[]`). You can also directly pass the exact size.
  - The elements of the `arrayList` are then copied into this newly created array.
- **Step 3:** The new array is returned and assigned to `array1`.

This approach ensures type safety, convenience, and efficiency when converting a list to an array in Java.