**Data Structures**

**Linear data Structures**

Linear data structures are sequential and ordered in a way so that there is only one first element and has only one next element, there is only one last element and has only one previous element, while all other elements have a next and a previous element.

1. **Arrays** – An array is a linear data structure representing a group of similar elements, accessed by an index. Size of an array must be provided before storing data.

* Each element in an array is of the same datatype and has the same size
* Elements of the array are store at contiguous memory locations with the first element is starting at the smallest memory location.
* Elements of the array can be randomly accessed
* The array data structure is not completely dynamic

Example:

int[] myArray = new int[5];

You can populate the array element by element using the array index:

myArray [0] = 101;

myArray [1] = 102;

You can also create and initialize an array directly using the flower brackets ({}).

int [] myArray = {10, 20, 30, 40, 50}

1. **Array-List** – Similar to an Array but an ArrayList is Resizable. Also elements can be added or removed from an ArrayList whenever you want unlike an array where you must create a new array.
   * Its best to use an ArrayList when you want to access random items frequently.
   * Also best to use an ArrayList when you only need to add or remove elements at the end of the list.

Example

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

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

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

System.out.println(cars);

}

}

Various methods in ArrayList include

* .get() – To access an element
* .set(location,”input”) – to modify an element
* .remove() – to remove an element
* .clear() – to clear all elements
* .size() – to get size of arraylist

1. **Linked Lists** – A linked list is a linear data structure with the collection of multiple nodes, where each element stores its own data and a pointer to the location of the next element. The last link in a linked list points to null, indicating the end of the chain. An element in a linked list is called a **node**. The first node is called **head**. The last is called **tail.**

* Singly linked lists are Uni-Directional
* Doubly linked lists are bi directional
* Its best to use a linked list when you only use the list by looping through it instead of accessing random items
* Its best to use a linked list when you frequently need to add or remove items from the beginning middle or end of the list.

import java.util.LinkedList;

public class Main {

public static void main(String[] args) {

LinkedList<String> cars = new LinkedList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

System.out.println(cars);

}

}

Various methods in Linked list include

* addFirst()
* addLast()
* removeFirst()
* removeLast()
* getFirst()
* getLast()

1. **Stacks** – a stack is an abstract structure, it’s a collection of objects that are inserted and removed according to the Last-in-first-out or LIFO principle. Objects can be inserted into a stack at any point of time, but only the most recently inserted object can be removed at any time.

* It is an ordered list in which insertion and deletion can be performed only at one end that is called the top
* Recursive data structure with a pointer to its top element
* Supports two most fundamental methods **push(e)** insert element e, to the top of the stack and **pop()** remove and return the top element on the stack
* Practical examples of a stack include when reversing a word, to check the correctness of parenthesis sequence, implementing back functionality in browsers.

Example:

import java.util.\*;

public class StackDemo {

static void showpush(Stack st, int a) {

st.push(new Integer(a));

System.out.println("push(" + a + ")");

System.out.println("stack: " + st);

}

static void showpop(Stack st) {

System.out.print("pop -> ");

Integer a = (Integer) st.pop();

System.out.println(a);

System.out.println("stack: " + st);

}

public static void main(String args[]) {

Stack st = new Stack();

System.out.println("stack: " + st);

showpush(st, 42);

showpush(st, 66);

showpush(st, 99);

showpop(st);

showpop(st);

showpop(st);

try {

showpop(st);

} catch (EmptyStackException e) {

System.out.println("empty stack");

}

}

}

Various methods in Stack class:

* Empty() returns true if nothing on top of stack
* Peek() returns the element on the top of the stack
* Pop() removes and returns the top element
* Push() pushes an element on the top of the stack
* Search() dtermeines whether an object exists in the stack

And many more

1. **Queues** – Queues are an abstract data structure. Unlike a stack, the queue is a collection of objects that are inserted and removed according to the first in first out or FIFO principle. That is elemnts can be inserted at any point of time, but only the element that has been in the queue the longest can be removed at any time.

* Supports two of the most fundamental methods
  + enqueue(e): which inserts element e, at the rear of the queue
  + dequeuer(): remove and return the element from the front of the queue

Queues are used in the asynchronous transfer of data between two processes, CPU scheduling, Disk Scheduling and other situations where resources are shared among multiple users and served on first in first out server basis.

**Example:**

import java.util.LinkedList;

import java.util.Queue;

public class QueueExample {

    public static void main(String[] args)

    {

        Queue<Integer> q = new LinkedList<>();

**// Adds elements {0, 1, 2, 3, 4} to the queue**

        for (int i = 0; i < 5; i++)

            q.add(i);

**// Display contents of the queue.**

        System.out.println("Elements of queue "

                           + q);

**// To remove the head of queue.**

        int removedele = q.remove();

        System.out.println("removed element-"

                           + removedele);

        System.out.println(q);

**// To view the head of queue**

        int head = q.peek();

        System.out.println("head of queue-"

                           + head);

**// Rest all methods of collection interface like size and contains**

**// can be used with this implementation.**

        int size = q.size();

        System.out.println("Size of queue-"

                           + size);

    }

}

The queue method inherits all methods present in the collections interface. Plus other methods like

* add(element) add elements to the tail of the queue
* element() this method is similar to peek
* peek() this method is used to view the head of the queue without removing it
* poll() removes and returns the head of the queue
* offer(element) used to insert an element in the queue

**Hierarchical Data Structures**

Hierarchical data structures are nonlinear data structures. These structures mainly represent data containing the hierarchical relationship between its elements for example records, trees etc.

1. **Binary tree –** Is a hierarchical tree data structure in which each node has at most two children, which are referred to as the left child and the right child. Each binary tree has the following groups of nodes:

* Root node: it is the topmost node and often referred to as the main node because all other nodes can be reached from root.
* Left Sub-Tree, which is also a binary tree
* Right Sub-Tree, which is also a binary tree
* **A binary tree can be traversed in two ways, Depth First Traversal: inorder (left-root-right), preorder (root-left-right) and postorder(left-right-root) and Breadth first traversal: level order traversal.**
* **Time complexity is O(n)**
* Binary trees are typically used in many search applications where data is constantly entering/leaving, As a workflow for composting digital images for visual effects,used in almost every high-bandwidth router for storing router tables, Also used in wireless networking and memory allocation, used in compression algorithms.

**I would go to** [Binary Tree Data Structure - GeeksforGeeks](https://www.geeksforgeeks.org/binary-tree-data-structure/) for examples of the many things you can do with a Binary Tree.

1. **Binary Heap –** A binary heap is a complete binary tree, which answers to the heap property.

* Basically a variation of the binary tree with the following properties
* **Heap** is a complete binary tree: A tree is said to complete if all its levels, except possibly the deepest are complete.
* **Follows heap property: A binary heap is either a Min-Heap or a Max-Heap.**
* **Min binary heap** for every node in a heap, nodes value is lesser than or equal to values of the children
* **Max Binary Heap** for every node in a heap, the nodes value is greater than or equal to values of the children

Popular applications for binary heap include implementing efficient priority queues, efficiently finding the smallest or larges element in an array.

Go here to see code examples of Binary heap [Binary Heap - GeeksforGeeks](https://www.geeksforgeeks.org/binary-heap/)

**Hash Tables -** Imagine you have an object you want to assign a key to it to make searching very easy. To store that key/value pair, you can use a simple array like a data structure where keys are integers and can be used directly as an index to store data values. Some cases the keys may become to large and cannot be used. In this case we would use hashing.

In hashing, the large keys are converted into smaller keys by using hash functions. The values are then stored in a data structure called a hash table. A has table is a data structure that implements a dictionary to map unique keys.

A hash table has two major components:

* **Bucket Array –** A bucket array for a hash table is an array A of size N, where each cell of A is thought of as a bucket, that is, a collection of key-value pairs. The integer N defines the capacity of the array.
* **Hash Function –** It is any function that maps each key k in our map to an integer in the range [0,n-1] where n is the capacity of the bucket array for this table.

When the object is added to the hash table and have the same hash code this is called collision, to deal with this there are techniques like chaining and open addressing.