Data Structures

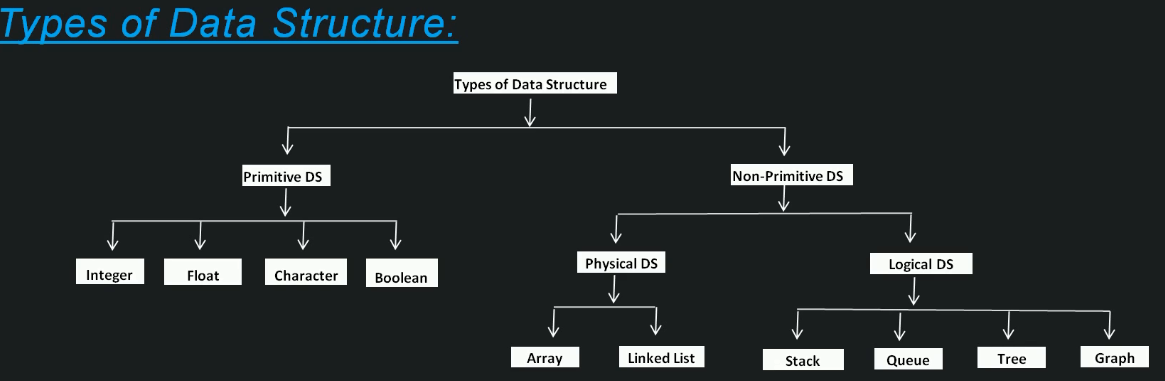
And

Algorithms

What is Data Structure?

Data Structure is way to ‘organize data’ that enables to be processed in an efficient time.

Types of Data Structures



What is Algorithm?

Algorithm is a set of rules to be followed to solve a problem

Algorithm Runtime Analysis

Array

Linked List

Stack

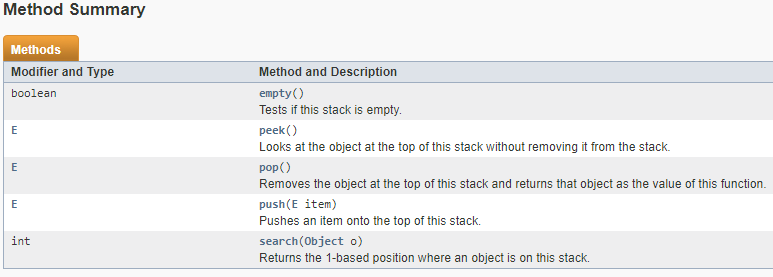
A stack is a basic data structure that can be **logically** thought as linear structure represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack.

The basic concept can be illustrated by thinking of your data set as a stack of plates or books where you can only take the top item off the stack in order to remove things from it. This structure is used all throughout programming.

The basic implementation of a stack is also called a **“Last In First Out**” structure; however there are different variations of stack implementations.

There are basically three operations that can be performed on stacks. They are:

* inserting (“pushing”) an item into a stack
* deleting (“popping”) an item from the stack
* displaying the contents of the top item of the stack (“peeking”)



The Java Stack class, java.util.Stack, is a classical stack data structure. You can push elements to the top of a Java Stack and pop them again, meaning read and remove the elements from the top of the stack.

The Java Stack class actually implements the [**Java List**](http://tutorials.jenkov.com/java-collections/list.html) interface, but you rarely use a Stack as a List - except perhaps if you need to inspect all elements currently stored on the stack

Stack stack = new Stack();

Stack stack = new Stack();

stack.push("1");

String topElement = stack.peek();

Stack stack = new Stack();

stack.push("1");

stack.push("2");

stack.push("3");

**int index = stack.search("3");** //index = 1

Stack stack = new Stack();

stack.push("123");

stack.push("456");

stack.push("789");

Iterator iterator = stack.iterator();

while(iterator.hasNext()){

Object value = iterator.next();

}

Stack stack = new Stack();

stack.push("A");

stack.push("B");

stack.push("C");

Stream stream = stack.stream();

stream.forEach((element) -> {

System.out.println(element); // print element

});

Queue

A queue is an abstract data type or a linear data structure, in which the first element is **inserted** from one end (the “**tail**”), and the **deletion** of existing element takes place from the other end (the “**head**”). A queue is a “**First In First Ou**t” structure. "First In First Out" means that elements put in the queue first will come out first, and elements put in the queue last will come out last.

An example of a queue are lines of people waiting. The first person in the line goes first, and the last person in the line goes last.

The process of adding an element to a queue is called “**enqueuing**” and the process of removing an element from a queue is called “**dequeuing**”

The Java Queue interface, java.util.Queue represents a data structure designed to have elements inserted at the end of the queue, and elements removed from the beginning of the queue. This is similar to how a queue in a supermarket works.

The Java Queue interface is a subtype of the [**Java Collection**](http://tutorials.jenkov.com/java-collections/collection.html) interface. It represents an ordered sequence of objects just like a [**Java List**](http://tutorials.jenkov.com/java-collections/list.html), but its intended use is slightly different.

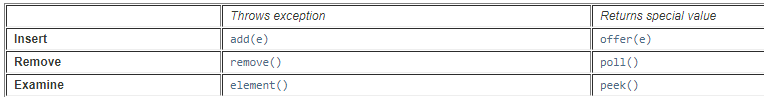
Being a Collection subtype all methods in the Collection interface are also available in the Queue interface.

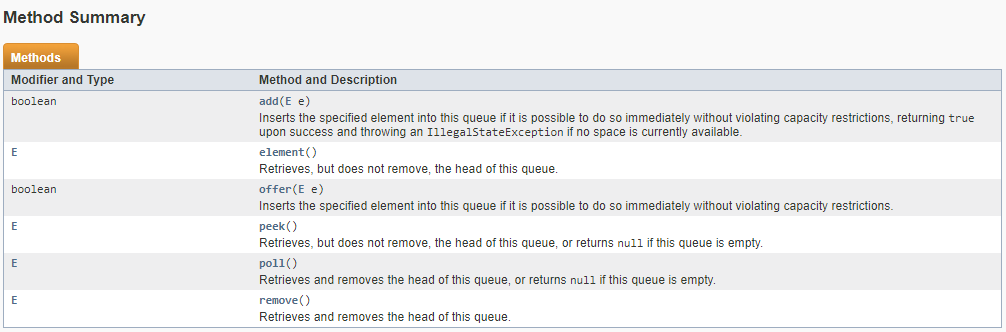
Since Queue is an interface you need to instantiate a concrete implementation of the interface in order to use it. You can choose between the following Queue implementations in the Java Collections API:

* java.util.LinkedList
* java.util.PriorityQueue

LinkedList is a pretty standard queue implementation.

PriorityQueue stores its elements internally according to their natural order (if they implement Comparable), or according to a Comparator passed to the PriorityQueue.





Queue queueA = new LinkedList();

Queue queueB = new PriorityQueue();

Tree

Graphs

Hashing

Sorting

Magic Framework

Greedy Algorithm

Divide & Conquer Algorithms

Dynamic Programming