EXPERIMENT 1

**Aim: - Implement the following Data structures in Java**

**I) Queues**

**II) Set**

**III) Map**

**Data Structures: -** A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently. Efficient data structures are a key to designing efficient algorithms. Java - The Set Interface. A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. Java - The Map Interface. Advertisements. The Map interface maps unique keys to values. A key is an object that you use to retrieve a value at a later date. Given a key and a value, you can store the value in a Map object. The objective is to write java programs using Map and Set interface.

The java.util.Queue is a subtype of java.util.Collection interface. It is an ordered list of objects with its use limited to inserting elements at the end of list and deleting elements from the start of list i.e. it follows FIFO principle.

Since it is an interface, we need a concrete class during its declaration. There are many ways to initialize a Queue object, most common being-

1. As a Priority Queue

2. as a Linked List

Operations on Queue:

▪ add ()-Adds an element at the tail of queue. More specifically, at the last of linked list if it is used, or according to the priority in case of priority queue implementation.

▪ Peek ()-To view the head of queue without removing it. Returns null if queue is empty.

▪ Element ()-Similar to peek (). Throws No Such Element Exception if queue is empty.

▪ Remove ()-Removes and returns the head of the queue. Throws No Such Element Exception when queue is empty.

▪ Poll ()-Removes and returns the head of the queue. Returns null if queue is empt

Map interface, which is also a part of java collection framework, doesn't inherit from Collection interface. Collection interface is a member of java.util package. Collections are a utility class in java.util package. It consists of only static methods which are used to operate on objects of type Collection.

**Advantages of Ring Topology**

1. Transmitting network is not affected by high traffic or by adding more nodes, as only the nodes having tokens can transmit data.
2. Cheap to install and expand

**Disadvantages of Ring Topology**

1. Troubleshooting is difficult in ring topology.
2. Adding or deleting the computers disturbs the network activity.
3. Failure of one computer disturbs the whole network.

**Set in Java**

• Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.

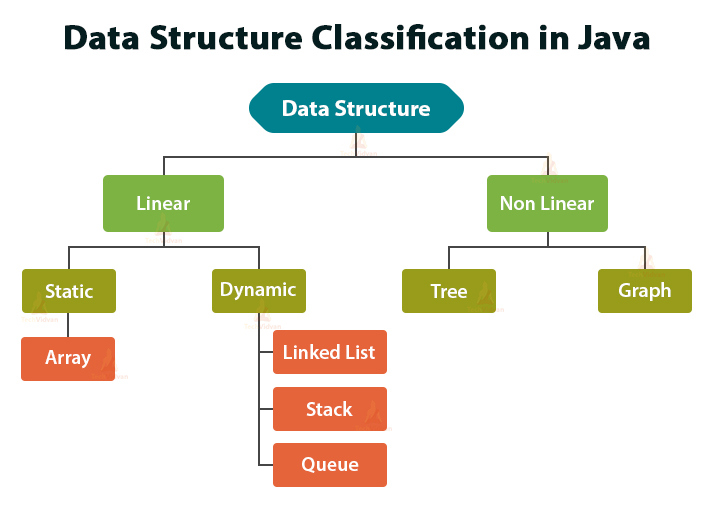
• Basically, Set is implemented by Hash Set, Linked Set or Tree Set (sorted representation).

• Set has various methods to add, remove clear, size, etc to enhance the usage of this interface

**Map in java**

The java.util. Map interface represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit different from the rest of the collection types.

A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value (Hash Map and LinkedHashMap) but some do not (Tree Map).

** **

**Implementation of Queues**

Import java.util.LinkedList;

Import java.util.Queue;

Public class QueueExample

{

Public static void main (String [] args)

{

Queue<Integer> q = new Linked List<> ();

// adds elements {0, 1, 2, 3, 4} to 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 = h.remove ();

System.out.println ("removed element-" + removedele);

System.out.println (q);

// to view the head of queue

In 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);

}

}

**Output**

Elements of queue-[0, 1, 2, 3, 4]

Removed element-0

[1, 2, 3, 4]

Head of queue-1

Size of queue-4

**Set implementation**

// Java code for adding elements in Set

Import java.util.\*;

Public class Set example

{

Public static void main (String [] args)

{

// Set demonstration using Hash Set

Set<String> hash Set = new Hash Set<String> ();

hash\_Set.add ("srinivas");

hash\_Set.add ("shoba");

hash\_Set.add ("srithan");

hash\_Set.add ("krithik");

hash\_Set.add ("shoba");

System.out.println ("Set output without the duplicates");

System.out.println (hash Set);

// Set demonstration using Tree Set

System.out.println ("Sorted Set after passing into Tree Set");

Set<String> tree Set = new Tree Set<String> (hash Set);

System.out.println (tree Set);

}

}

**Output**

Set output without the duplicates [srinivas, shoba, srithan sai, sai krithik]

Sorted Set after passing into Tree Set [sai krithik, shoba, srinivas, srithan sai]

**Implement the following Data structures in Java Hash Set**

Import java.util.\*;

Class Test

{

Public static void main (String [] args)

{

Hash Set<String> h = new Hash Set<String>();

// adding into Hash Set

h.add ("India");

h.add ("Australia");

h.add ("South Africa");

h.add ("India"); // adding duplicate elements

// printing HashSet

System.out.println (h);

System.out.println ("List contains India or not:" +

h.contains ("India"));

## // removing an item

## h.remove ("Australia");

## System.out.println ("List after removing Australia:"+h);

## // iterating over hash set items

## System.out.println ("Iterating over list :");

## Iterator<String> I = h.iterator ();

## While (i.hasNext ())

## System.out.println (i.next ());

## }

## }

## Output

[Australia, South Africa, India]

List contains India or not: true

List after removing Australia: [South Africa, India]

Iterating over list:

South Africa

India

**Implement the following Data structures in Java Map interface**

Import java.util.\*;

Class HashMapDemo

{

Public static void main (String args [])

{

Hash Map< String, Integer> hm =

New Hash Map< String, Integer> ();

hm.put ("a", new Integer (100));

hm.put ("b", new Integer (200));

hm.put ("c", new Integer (300));

hm.put ("d", new Integer (400));

// Returns Set view

Set< Map.Entry< String, Integer> > st = hm.entrySet ();

For (Map.Entry< String, Integer> me:st)

{

System.out.println (me.getKey () +":");

System.out.println (me.getValue ());

}

}

} **Output**

A: 100

B: 200

C: 300

D: 40

**Experiment No.2**

**Implement the following Data structures in Java**

**I) Linked Lists**

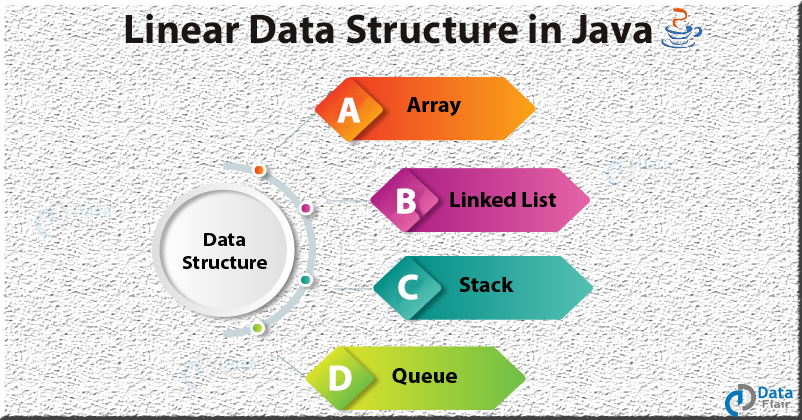
**II) Stacks**

**III) Queues**

**IV) Set**

**V) Map**

**Data Structures**. A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently. Efficient data structures are a key to designing efficient algorithms. The Java collections framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures. Although referred to as a framework, it works in a manner of a library. The JCF provides both interfaces that define various collections and classes that implement them .The objective of this program is to implement Linked list stack Queue data Structures.



Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (Array List, Vector, Linked List, Priority Queue, Hash Set, Linked Hash Set, Tree Set etc).

Map interface, which is also a part of java collection framework, doesn't inherit from Collection interface. Collection interface is a member of java.util package. Collections are a utility class in java.util package. It consists of only static methods which are used to operate on objects of type Collection Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

Typing discipline: Static, strong, safe, nominative, manifest

First appeared: May 23, 1995; 25 years ago

Stable release: Java SE 14 / March 17, 2020; 5 months ago

Designed by: James Gosling

Developer: Oracle Corporation

Filename extensions: java, class, jar

**Implementation of Linked List**

Include the package java.util.\*;

Public class Test

{

Public static void main (String args [])

{

// creating object of class linked list

Linked List<String> object = new Linked List<String> ();

// adding elements to the linked list

Object. Add ("A");

object.add ("B");

object.addLast ("C");

object.addFirst ("D");

object.add (2, "E");

**// similar to above add F and G**

System.out.println ("Linked list: " + object);

// removing elements from the linked list

Object. Remove ("B");

Object. Remove (3);

object.removeFirst ();

object.removeLast ();

System.out.println ("Linked list after deletion: " + object);

// Finding elements in the linked list

Boolean status = object. Contains ("E");

If (status)

System.out.println ("List contains the element 'E' ");

Else

System.out.println ("List doesn't contain the element 'E'");

// Number of elements in the linked list

Into size = object. Size ();

System.out.println ("Size of linked list = " + size);

// Get and set elements from linked list

Object element = object. Get (2);

System.out.println ("Element returned by get (): " + element);

Object. Set (2, "Y");

System.out.println ("Linked list after change: " + object);

}

}

**Expected Output:**

Linked list: [D, A, E, B, C, F, G]

Linked list after deletion: [A, E, F]

List contains the element 'E'

Size of linked list = 3

Element returned by get (): F

Linked list after change: [A, E, Y]

**Write the java statements for importing packages**

java.io.\*, java.util.\*.   
Class My Stack   
{   
// Pushing element on the top of the stack   
static void stack push (Stack<Integer> stack)   
{   
for(int i = 0; i < 5; i++)   
{   
stack. Push (I);  
}   
}   
// Popping element from the top of the stack   
static void stack pop(Stack<Integer> stack)   
{   
System.out.println("Pop :");

for(int i = 0; i < 5; i++)   
{   
Integer y = (Integer) stack.pop();   
System.out.println(y);   
}   
}   
  
// Displaying element on the top of the stack   
static void stack\_peek(Stack<Integer> stack)   
{   
Integer element = (Integer) stack. Peek ();   
System.out.println ("Element on stack top: " + element);  
}

// Searching element in the stack

Static void stack\_search (Stack<Integer> stack, int element)

{

Integer pos = (Integer) stack. Search (element);

If (pos == -1)

System.out.println ("Element not found");

Else

System.out.println ("Element is found at position " + pos);

}

Public static void main (String [] args)

{

Stack<Integer> stack = new Stack<Integer> ();

Stack push (stack);

Stack pop (stack);

Stack push (stack);

stack\_peek (stack);

stack\_search (stack, 2);

stack\_search (stack, 6);

}

}

**Expected Output:**

Pop:

4

3

2

1

0

Element on stack top: 4

Element is found at position 3

Element not found

**Viva questions**

1. **What is a Data Structure?**

A data structure is a way of organizing the data so that the data can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for implementation of databases, while compiler implementations usually use hash tables to look up identifiers.

**2 What are linear and non linear data Structures?**

**Linear:** A data structure is said to be linear if its elements form a sequence or a linear list. Examples: Array. Linked List, Stacks and Queues

**Non-Linear:**A data structure is said to be non-linear if traversal of nodes is nonlinear in nature. Example: Graph and Trees.

**3 What are the various operations that can be performed on different Data Structures?**

**Insertion**- Add a new data item in the given collection of data items.

**Deletion** - Delete an existing data item from the given collection of data items.

**Traversal** -Access each data item exactly once so that it can be processed.

**Searching** - Find out the location of the data item if it exists in the given collection of data items.

**Sorting** - Arranging the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

**4** [**How is an Array different from Linked List?**](https://www.geeksforgeeks.org/linked-list-vs-array/)

The size of the arrays is fixed; Linked Lists are Dynamic in size.

Inserting and deleting a new element in an array of elements is expensive, whereas both insertion and deletion can easily be done in Linked Lists.

Random access is not allowed in Linked Listed.

Extra memory space for a pointer is required with each element of the Linked list.

Arrays have better cache locality that can make a pretty big difference in performance.

**5 What is a Queue, how it is different from stack and how is it implemented?**

Queue is a linear structure which follows the order is **F**irst **in First out** (FIFO) to access elements. Mainly the following are basic operations on queue: **Enquire, Deque**, **Front, Rear**  
the difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added. Both Queues and Stacks can be implemented using Arrays and Linked Lists.