**DATA STRUCTURES IN JAVA**

-In Modern world , data and its information is an essential part

-Data is a collection of raw facts.

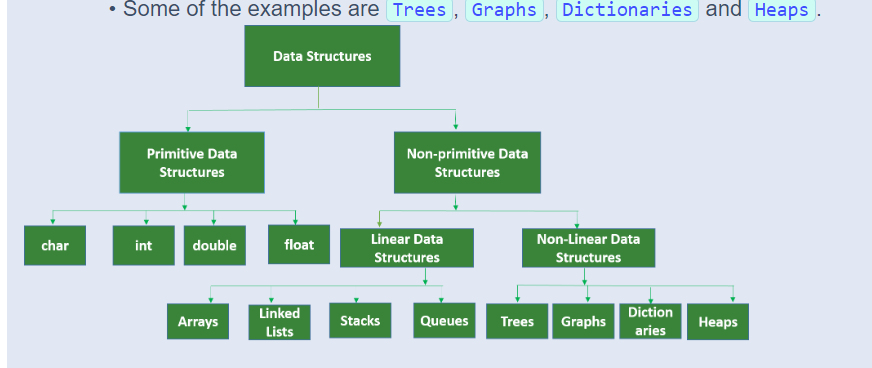
-Information is the Processed data.

-if data is not organized effectively , it is difficult to perform any task on large amount of data.

-if the data is stored in a well-organized way on storage media then it can be accessed quickly.

-A DATA STRUCTURE is a particular way of organizing a large amount of data more efficiently

**Types of Data Structures:**

****

**1)Primitive Data Sructure :** Built in Data Structures

**2)Non-primitive Data Structures:** User Defined Data Structures

**Linear Data Structure:**

-If a data structure is organizing the data in sequential order then that data structure is called a linear data structure.

**Non-Linear Data Structure:**

-If a data structure is organizing the data in random order or hierarchical order , not in sequential order .

**Homogeneous Data Structures:**

**-**In Homogeneous data structure , all the data items are of same type ex:Arrays

**Non-Homogeneous Data Structures:**

-In Non-Homogeneous data structure , all the data item may or may not be of same type ex: Structures

**Static Data Structures:**

-In static data Structure , the size and structures associated memory locations are fixed during compile time Ex: Arrays.

**Dynamic Data Structures:**

**-**It can expand or shrink depending upon the program

-This Expansion and Shrinking happen during the program runtime.

Ex:Linked Lists, Stack using Linked Lists, Queues Using Linked List, Tress , Heaps etc.

Advantages of Data Structures in Java:

-Efficient Data Organization: Data Structures provide organized ways to store and manage data for efficient access

-Better Performance : Developers can improve performance in terms of speed and memory by selecting the suitable data structure

-Code reusability : java offers a wide range od built in data structures that are simple for programmers to use

-Scalability: it allow applications to handle large volumes of data efficiently

-

**Introduction to Algorithm:**

-An Algorithm is a finite set of instructions or logic to perform a particulat task.

-Algorithm is independent of programming language

-An Algorithm is a logic to solve a given problem.

-An Algorithm is expressed as flow chart or pseudocode Algorithm

-An Algorithm can be Defined as a sequence of steps

-These sequence of steps of an algorithm can be stated in human readable English Statements

-Hence there may be more than one algorithm for a given problem.

**Properties of an Algorithm**:

1)Input: It Accept 0 or More Inputs

2)Output : it should generate at least one output

3)Definiteness: Each step of an algorithm should perform a clear defined task

4)Finiteness: An Algorithm must always terminate after a finite number of steps

5)Effectiveness: the accuracy of the output determine the effectiveness of algorithm

6)Correctness: Each Step of the Algorithm Must generate a correct output.

-An Algorithm is said to be efficient and fast , if it takes less time to execute and consumes less memory

-The Performance of an algorithm is measured on the basis of

-Time complexity

-Space Complexity

-**Time Complexity:** It is a way to represent the amount of time required by the program to run till its completion**.**

**-Space Complexity:** It is the amount of memory space that is required by the algorithm for its execution **.**

**Note: One heap and one stack is maintained during the program execution.**

**Heap:** Dynamic Memory allocation take place

**Stack:** It is a segment where automatic variables and function call is stored.

**Note:** When we are measuring the space complexity of an algorithm , we consider only the data space of the algorithm and ignore the instruction space and environmental space.

**Space Complexity:**

**-**It is The amount of memory used by the algorithm to execute and produce the result.

-Space Complexity is the number of memory cells an algorithm needs.

**-Instruction Space:** Memory Required to store the program Instructions . This Space is fixed and depends on the number of instructions in the algorithm.

-**Data Space** : Memory space Required to store Constants and variables.

-**Environment space**: Memory space that is required for storing data between functions.

Bool – 1 byte

Char – 1 byte

Int short – 2 bytes

Float long – 4 bytes

Double long – 8 bytes

Ex:

**int** sum(**int** x, **int** y) {

**int** z = x + y;

return z;

}

-The above code takes two inputs x and y of type int as formal parameters

-and variable z to store sum

-The int data type takes 2 bytes of memory

-Total space complexity **3 (number of variables) \* 2 (Size of each variable) = 6 bytes**.

**Constant space complexity:**

-If the space required of an algorithm is fixed for any input given to the algorithm .

Ex-2:

//Function to calculate a sum of **n** elements in the array

//**n** is the number of elements in the array.

**int** sum(**int** a[ ] , **int** n) {

**int** x = 0, i = 0

for (i = 0; i < n; i++) {

x= x + a[i];

}

return x;

}

-In the above code, **2 \* n** bytes of memory (size of int data type is **2**) is required by the array a[ ] and **2** bytes of memory for each variable of x, n and i.

-Hence the total space requirement for the above code would be **(2 \* n + 6)**.

**Linear Space Complexity**:

-The space complexity of the program is increasing linearly with the size of the array (input) n.

**Quadratic Space Complexity:**

**-**when the memory requirement of the algorithm increases quadratic to the given input.

**Cubic Space Complexity:**

**-**when the memory requirement of the algorithm increases cubic to the given input.

**Arrays**:

-An Array is a basic data structure used in java

-it is of fixed size

-it provide quick and easy access to elements based on index

Advantages:

-Data Organization

-Random Access

-Fixed Size

-Homogenous Elements

-Iterator

-Memory efficiency -by storing in continuous memory location

Disadvantages:

-Fixed Size

-Memory Wastage

-Insertion and deletion overhead

**2)ArrayList:**

-it is a dynamic data structure that allows for the storage and manipulation of elements

-it is a part of java collection

Advantages:

-Dynamic Size: unlike a array , ArrayList can dynamically grow and shrink in size as elements.

-Easy Element Manipulation : ArrayList offer methods to add, remove , and modify elements at any position within the list.

-Random Access : ArrayList support random Access to elements using their index

Disadvantages:

-Slower insertion And deletion

-limited Performance for search

Methods In ArrayList:

1)add(element) : Adds the element to the end of the list

2)add(int index ,element): Insert the Specified element at the Specified position in the list

3)remove(element) : Removes the first occurrence of the specified element from the list

4)remove(int index) : Removes the element at the specified position in the list

5)get(int index) : Returns the elements at the specified position in the list

6)set(int index , element) : Replace the element at the specified position in the list with the specified element

7)size() : Returns the size of th list

8)isEmpty() : Returns true if the list is empty

9)contains(element) : Returns true if the list contains the specified element

10)clear() – Removes all elements from the List

11)indexOf(element) – Returns the index of the first occurrence of the specified element in the list else -1 if the list does not contains the element.

12)toArray() : Returns an array containing all of the elements in the list

13)addAll(otherlist) :Appends all of the elements in the specified element to the end of the list

14)removeAll(otherlist) : Removes all of the elements in the specified list from the list



Linked List:

**Basic Operations**

1. **add(E e)**: Appends the specified element to the end of the list.

Ex:

linkedList.add("Element");

1. **add(int index, E element)**: Inserts the specified element at the specified position in the list.

**Ex**:

linkedList.add(1, "Element");

1. **addFirst(E e)**: Inserts the specified element at the beginning of the list.

Ex:

linkedList.addFirst("Element");

1. **addLast(E e)**: Appends the specified element to the end of the list.

Ex:

linkedList.addLast("Element");

1. **remove(Object o)**: Removes the first occurrence of the specified element from the list.

Ex:

linkedList.remove("Element");

1. **remove(int index)**: Removes the element at the specified position in the list.

Ex:

linkedList.remove(1);

1. **removeFirst()**: Removes and returns the first element from the list.

Ex:

String element = linkedList.removeFirst();

1. **removeLast()**: Removes and returns the last element from the list.

String element = linkedList.removeLast();

1. **get(int index)**: Returns the element at the specified position in the list.

Ex:

String element = linkedList.get(1);

1. **getFirst()**: Returns the first element in the list.

**Ex**:

String element = linkedList.getFirst();

1. **getLast()**: Returns the last element in the list.

Ex:

String element = linkedList.getLast();

1. **set(int index, E element)**: Replaces the element at the specified position in the list with the specified element.

Ex:

linkedList.set(1, "NewElement");

1. **size()**: Returns the number of elements in the list.

Ex:

int size = linkedList.size();

1. **isEmpty()**: Returns true if the list contains no elements.

Ex:

boolean empty = linkedList.isEmpty();

1. **contains(Object o)**: Returns true if the list contains the specified element.

Ex:

boolean contains = linkedList.contains("Element");

1. **clear()**: Removes all elements from the list.

Ex:

linkedList.clear();

1. **toArray()**: Returns an array containing all of the elements in the list.

Object[] array = linkedList.toArray();

Single Linked List:

-In a singly linked list each node contains a data element and a reference to the next node in the sequence.

-it allows for efficient insertion and deletion at the beginning or end of the list

Characteristics:

-Each node points to the next node

-traversal is only possible in one direction(forward)

Refer program singleLinkedList.java

**Double Linked List:**

**-**In a doubly linked list, each node contains a data element and two references

-one to the next node and one to the previous node

-it allows for bidirectional traversal and easier removal of nodes

Characteristics:

-each node has two pointers one to the next node and one to the previous node.

-traversal is possible in both directions (forward and backward)

-useful in scenarios where bidirectional traversal is needed.

Refer doubleLinkedList.java

**Circular Linked List:**

**-**In a circular linked list the last node points back to the first node, forming a loop

-This can be either singly or doubly circular

Characteristics:

-The last node points to the first node

-Useful in application that require a circular iteration over the list

**Operations in a Linked List:**

**1.Insertion:**

**-** insert at the Beginning

-insert at the end

-insert at specific position

**2.Deletion**

-Delete at the Beginning

-Delete at the end

-Delete at a Specific Position

**3.Traversal**

-Print List

**4.Searching**

-Search for an Element

**5.Reversing the List**

-Reverse List

**Stack:**

-Stack is a linear data Structure

-it allows LIFO Mechanism

-LIFO means last In First Out.

-Last Element will be deleted First

-The Best example for stack is Arranging the books and coins

-If there are no elements in the stack then top = -1

-Stack can be implemented in two ways

.Arrays(Static Memory)

.Linked List(dynamic Memory)

-Stack Class is a Synchronized Which is thread-safe

-If you don’t need synchronization , you might use ‘ArrayDeque’ as a alternative for better performance.

-Stack is a legacy class since it extends ‘Vector’ .

**Stack implementation using Arrays:**

-By Using Stack Array the size is fixed.

-In Arrays We need to verify two conditions

-overflow

- underflow

-The overflow occurs in push operation

-The underflow occurs in pop operation

**Stack implementation using Linked List:**

**-**-By Using Stack LinkedList the size is not fixed.

-In Linked List We need to verify two conditions

- underflow

-The underflow occurs in pop operation

**Basic Operations**:

1.Push: Adds an element to the top of the stack.

2.Pop : Removes and returns the element at the top of the stack.

3.Peek : Returns the element at the top of the stack without removing it.

4.Search: Finds the position of an element , if the element is not found it returns -1

5.Check if Empty: Checks if the stack is empty

6.Size : Returns the number of elements in the stack

**Ways to Implement a stack:**

1.Using java Built in Stack Class – Provides a basic stack implementation with built-in methods

2.Using ArrayDeque – Offers better performance for stack operations due to lack of synchronization

3.Using a Linked List – Flexible and dynamic

4.Using an Array – Efficient for fixed-size stacks

5.Sing a Linked Node – Custom implementation that uses linked nodes.

Refer programs in Eclipse.