Java Week 9

Data Structure

**Data Structure** is a way to store and organize **data** so that it can be used efficiently.

* Some examples of Data Structures are
* arrays,
* Linked List,
* **Stacks**. ...
* **Queues**. ...
* Hash Tables. ...
* Trees. ...
* Heaps

**Data:** Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.

**Group Items:** Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

**Record:** Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

**File:** A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.

**Attribute and Entity:** An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity.

**Field:** Field is a single elementary unit of information representing the attribute of an entity.

### **Need of Data Structures**

**Processor speed:** To handle very large amout of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

**Data Search:** Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.

**Multiple requests:** If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process

Solution 1

in order to solve the above problems, data structures are used. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

Advantages of Data Structures

**Efficiency:** Efficiency of a program depends upon the choice of data structures. For example: suppose, we have some data and we need to perform the search for a particular record. In that case, if we organize our data in an array, we will have to search sequentially element by element. hence, using array may not be very efficient here. There are better data structures which can make the search process efficient like ordered array, binary search tree or hash tables.

**Reusability:** Data structures are reusable, i.e. once we have implemented a particular data structure, we can use it at any other place. Implementation of data structures can be compiled into libraries which can be used by different clients.

**Abstraction:** Data structure is specified by the ADT which provides a level of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.

**Data Structure Classification**



**Linear Data Structures:** A data structure is called linear if all of its elements are arranged in the linear order.

# String Class and its methods

**Strings**, which are widely used in **Java** programming, are a sequence of characters. In the **Java** programming language, **strings** are objects. The **Java** platform provides the **String** class to create and manipulate **strings.**

**Creating a String**

**There are two ways to create a String in Java.**

1. String literal
2. Using new keyword

**String literal**

In java, Strings can be created like this: Assigning a String literal to a String instance:

String str1 = "Welcome";

String str2 = "Welcome";

## Why do we need a Linked List?

You must be aware of the arrays which is also a linear data structure but **arrays have certain limitations such as:**

**-Size of the array is fixed**

- Array elements **need contiguous memory locations** to store their values

-  **Inserting an element in an array is performance wise expensive**

**These limitations are handled in the Linked List by providing following features:**

1. Linked list allows **dynamic memory allocation**,
2. Linked list elements **don’t need contiguous memory locations**
3. Insert and delete operations in the Linked list are not performance wise expensive because adding and deleting an element from the linked list does’t require element shifting, only the pointer of the previous and the next node requires change.

## Why do we need a Linked List?

**Linked lists** are linear data structures that hold data in individual objects called nodes.

**Linked lists** are often used because of their efficient insertion and deletion. They can be used to implement stacks, queues, and other abstract data types.

You must be aware of the arrays which is also a linear data structure but **arrays have certain limitations such as:**

1. **Size of the array is fixed**
2. Array elements **need contiguous memory locations** to store their values
3. **Inserting an element in an array is performance wise expensive** as we have to shift several elements to make a space for the new element

**Inserting an element in an array is performance wise expensive** as we have to shift several elements to make a space for the new element.

1. Linked list allows **dynamic memory allocation**, which means memory allocation is done at the run time by the compiler and we do not need to mention the size of the list during linked list declaration.

2. Linked list elements **don’t need contiguous memory locations** because elements are linked with each other using the reference part of the node that contains the address of the next node of the list.

3. Insert and delete operations in the Linked list are not performance wise expensive because adding and deleting an element from the linked list does’t require element shifting, only the pointer of the previous and the next node requires change.