Chapter-20

What is Data Structure?

A *data structure* is a collection of data organized in some fashion. The structure not only stores data but also supports operations for accessing and manipulating the data. To define a data structure is essentially to define a class. The class for a data structure should use data fields to store data and provide methods to support such operations as search, insertion, and deletion. To create a data structure is therefore to create an instance from the class.

[In object-oriented thinking, a data structure, also known as a container or container object, is an object that stores other objects, referred to as data or elements.]

What is Java Collections Framework?

Java provides several more data structures that can be used to organize and manipulate data efficiently. These are commonly known as *Java Collections Framework*.

What is Collections?

The Collection interface defines the common operations for lists, vectors, stacks, queues, priority queues, and sets. Or A collection is a container that stores objects. The Java Collections Framework supports two types of containers:

- One for storing a collection of elements is simply called a *collection*.
- The other, for storing key/value pairs, is called a *map*.

- `Set`s store a group of non-duplicate elements.
- `List`s store an ordered collection of elements.
- `Stack`s store objects that are processed in a last-in, first-out fashion.
- `Queue`s store objects that are processed in a first-in, first-out fashion.
- `PriorityQueue`s store objects that are processed in the order of their priorities.

Note

All the interfaces and classes defined in the Java Collections Framework are grouped in the `java.util` package.

What is AbstractCollection?

The `AbstractCollection` class provides partial implementation for the `Collection` interface. It implements all the methods in `Collection` except the `add`, `size`, and `iterator` methods. These are implemented in appropriate concrete subclasses.

Describe add, addAll, removeAll, retainAll and clear() method:

add: The `add` method adds an element to the collection.

addAll : The `addAll` method adds all the elements in the specified collection to this collection.

remove: The `remove` method removes an element from the collection.

removeAll: The `removeAll` method removes the elements from this collection that are present in the specified collection.

retainAll: The `retainAll` method retains the elements in this collection that are also present in the specified collection.

clear: The `clear()` method simply removes all the elements from the collection.

Describe size, contains, containAll, isEmpty and toArray():

size: The `size` method returns the number of elements in the collection.

contain: The `containsAll` method checks whether the collection contains all the elements in the specified collection.

isEmpty: The `isEmpty` method returns `true` if the collection is empty.

toArray: The `toArray()` method, which returns an array representation for the collection.

When should a method throw an UnsupportedOperationException?

`UnsupportedOperationException`, a subclass of `RuntimeException`. This is a good design that you can use in your project. If a method has no meaning in the subclass. Example:

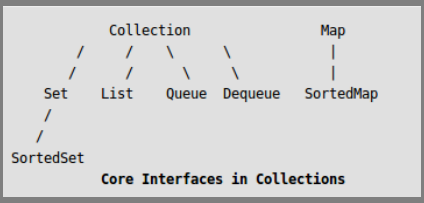
```
public void someMethod() {
throw new UnsupportedOperationException
("Method not supported");
}
```

What is Iterator?

`Iterator` is a classic design pattern for walking through a data structure without having to expose the details of how data is stored in the data structure. The `Collection` interface extends the `Iterable` interface. The `Iterable` interface defines the `iterator` method, which returns an iterator.

What is List?

The Java.util.List is a child interface of Collection. It is an ordered collection of objects in which duplicate values can be stored. *defines a collection for storing elements in a sequential order.* List Interface is implemented by ArrayList, LinkedList, Vector and Stack classes.



Core Interfaces in Collections

What are the differences between ArrayList and LinkedList classes?

ArrayList	LinkedList
It is the concrete implementations of the List interface.	It is the concrete implementations of the List interface.
It stores elements in an array.	It stores elements in a linked list.
To access randomly through an index, ArrayList is the most efficient.	To insert and delete at the beginning of the list, LinkedList is the best choice.

What is the syntax of creating List?

To create a String type [List<String> list1 = Arrays.asList("red", "green", "blue");]

To create a Inter type [List<Integer> list2 = Arrays.asList(10, 20, 30, 40, 50);]

What are the Differences between Comparable and Comparator?

`Comparable` is used to compare the objects of the class that implement `Comparable`. `Comparator` can be used to compare the objects of a class that doesn't implement `Comparable`.

Comparing elements using the `Comparable` interface is referred to as comparing using *natural order*, and comparing elements using the `Comparator` interface is referred to as comparing using *comparator*.

What are Vector and Stack classes?

`Vector` *is a subclass of* `AbstractList`. `Vector` is the same as `ArrayList`, except that it contains synchronized methods for accessing and modifying the vector. Synchronized methods can prevent data corruption when a vector is accessed and modified by two or more threads concurrently. Most of the methods in the `Vector` class listed in the UML diagram.

`Stack` *is a subclass of* `Vector` *in the Java API.* The `Stack` class extends `Vector` to provide a last-in, first-out data structure.

What is Queue?

A *queue* is a first-in, first-out data structure. Elements are appended to the end of the queue and are removed from the beginning of the queue. In a *priority queue*, elements are assigned priorities. The `Queue` interface extends `java.util.Collection` with additional insertion, extraction,

What is deque?

The name *deque* is short for "double-ended queue". The `Deque` interface extends `Queue` with additional methods for inserting and removing elements from both ends of the queue. The methods `addFirst(e)`, `removeFirst()`, `addLast(e)`, `removeLast()`, `getFirst()`, and `getLast()` are defined in the `Deque` interface.

Chapter-21

What is Set?

A set is an efficient data structure for storing and processing nonduplicate elements. We can create a set using one of its three concrete classes: **`HashSet`**, **`LinkedHashSet`**, or **`TreeSet`**. The `Set` interface extends the `Collection` interface. It does not introduce new methods or constants, but it stipulates that an instance of `Set` contains no duplicate elements.

What is HashSet?

The `HashSet` class is a concrete class that implements `Set`. You can create an empty *hash set* using its no-arg constructor or create a hash set from an existing collection. By default, the initial capacity is `16` and the load factor is `0.75`. A `HashSet` can be used to store *duplicate-free* elements.

What is LinkedHashSet?

`LinkedHashSet` extends `HashSet` with a linked-list implementation that supports an ordering of the elements in the set. The elements in a `HashSet` are not ordered, but the elements in a `LinkedHashSet` can be retrieved in the order in which they were inserted into the set.

What is TreeSet?

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Contains unique elements only like HashSet.
- Access and retrieval times are quiet fast.
- Maintains ascending order.

What is Map?

A *map* is a container object that stores a collection of key/value pairs. It enables fast retrieval, deletion, and updating of the pair through the key. A map stores the values along with the keys. The keys are like indexes. We *can create a map using one of its three concrete classes:* `HashMap`, `LinkedHashMap`, *or* `TreeMap`. A map cannot contain duplicate keys. Each key maps to one value.

Chapter-23

What is Insertion Sort?

The insertion-sort algorithm sorts a list of values by repeatedly inserting a new element into a sorted sublist until the whole list is sorted.

What is Bubble Sort?

A bubble sort sorts the array in multiple phases. Each pass successively swaps the neighboring elements if the elements are not in order. The bubble sort algorithm makes several passes through the array. On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged. The technique is called a *bubble sort* or *sinking sort*.

What is Merge Sort?

The merge sort algorithm can be described recursively as follows: The algorithm divides the array into two halves and applies a merge sort on each half recursively. After the two halves are sorted, merge them. Merge sort employs a divide-and-conquer approach to sort the array.

What is Quick Sort?

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

What is Heap Sort?

A heap sort uses a binary heap. It first adds all the elements to a heap and then removes the largest elements successively to obtain a sorted list. heap can be stored in an `ArrayList` or an array if the heap size is known in advance.
To sort an array using a heap, first create an object using the `Heap` class, add all the elements to the heap using the `add` method, and remove all the elements from the heap using the `remove` method. The elements are removed in descending order.

What is external Sort?

External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually a hard drive). External sorting typically uses a hybrid sort-merge strategy.

Chapter-24

Common Features for Lists

- Retrieve an element from the list.
- Insert a new element into the list.
- Delete an element from the list.
- Find out how many elements are in the list.
- Determine whether an element is in the list.
- Check whether the list is empty.

Chapter- 25

What is Tree?

A tree is a classic data structure with many important applications. A tree provides a hierarchical organization in which data are stored in the nodes.

Binary Search Trees

A binary search tree can be implemented using a linked structure. Lists, stacks, and queues are linear structures that consist of a sequence of elements. A binary tree is a hierarchical structure. It either is empty or consists of an element, called the root, and two distinct binary trees, called the left subtree and right subtree, either or both of which may be empty.

Chapter-29

What is Graph?

A graph is a weighted graph if each edge is assigned a weight. Weighted graphs have many practical applications.

Types of Weighted Graphs:

There are two types of weighted graphs: vertex weighted and edge weighted. In a *vertexweighted graph*, each vertex is assigned a weight. In an *edge-weighted graph*, each edge is assigned a weight. Of the two types, edge-weighted graphs have more applications.

Chepter-30

What is Thread?

A program may consist of many tasks that can run concurrently. A thread is the flow of execution, from beginning to end, of a task. A *thread* provides the mechanism for running a task.

What is run() method?

The `run()` method in a task specifies how to perform the task. This method is automatically invoked by the JVM. You should not invoke it. Invoking `run()` directly merely executes this method in the same thread; no new thread is started.

What is Semaphors?

a *semaphore* is an object that controls the access to a common resource. *Semaphores can be used to restrict the number of threads that access a shared resource.* To create a semaphore, you have to specify the number of permits with an optional fairness
policy.

What is Deadlocks?

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlock occurs when multiple threads need the same locks but obtain them in different order. A Java multithreaded program may suffer from the deadlock condition because the **synchronized** keyword causes the executing thread to block while waiting for the lock, or monitor, associated with the specified object.