**Chapter-20**

**Lists Stacks Queues and Priority Queues:**

**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"**);

}