**Unit-4**

**Collection in Java:**

**Definition:** In Java, a collection refers to a group of objects or elements. It represents the concept of storing and manipulating multiple objects together.

**Characteristics:** Collections can have different characteristics, such as allowing duplicates, maintaining order, or being sorted. They provide methods to add, remove, iterate over, and perform various operations on the stored elements.

**Example:** An ArrayList, HashSet, LinkedList, etc., are all examples of collections in Java.

**Collection Framework in Java:**

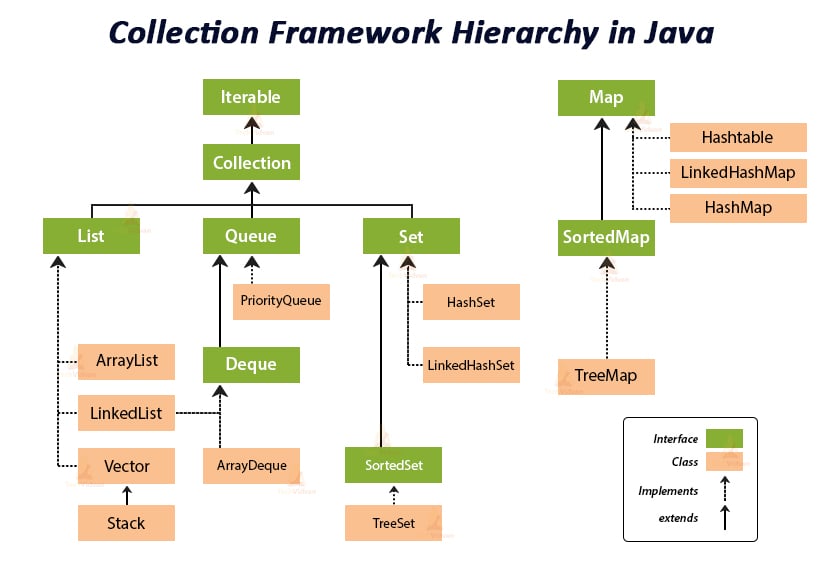
**Definition:** The Collection Framework in Java is a unified architecture that provides a set of interfaces, classes, and algorithms to represent and manipulate collections of objects.

**Purpose:** It offers reusable data structures and algorithms to handle collections efficiently. The framework aims to standardize the way collections are handled in Java applications.

**Components:** The Collection Framework includes interfaces like Collection, List, Set, Queue, Map, and their respective implementations, along with utility classes for operations like sorting and searching.

**Example Usage:** Developers can use the Collection Framework to store, retrieve, manipulate, and process data efficiently, whether it's a simple list of elements, a set of unique values, or a mapping of keys to values.

**Hierarchy of Collection Framework:**



**Interfaces:**

**1.Collection:** Root interface for all collection types.

**2.List:** Ordered collection allowing duplicate elements.

**3.Set:** Unordered collection that does not allow duplicates.

**4.Queue:** Ordered collection specifically designed for holding elements before processing (e.g., FIFO order).

**5.Map:** Represents a mapping of unique keys to values.

**Classes:**

Various implementations of interfaces like ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap, etc.

**Purpose:** The hierarchy allows developers to choose the appropriate collection type based on their requirements, providing flexibility, and ease of use.

**Inheritance:** Interfaces extend each other to provide common functionality, while classes implement these interfaces to provide concrete implementations.

**Iterator Interface:**

**Definition:**

The Iterator interface is part of the Java Collection Framework and resides in the java.util package.

It provides methods to iterate over the elements of a collection, check if there are more elements to iterate, and retrieve the next element in the iteration sequence.

It follows the Iterator design pattern, which separates the responsibility of accessing and iterating over elements from the underlying data structure.

**Methods:**

**boolean hasNext():** Returns true if the iteration has more elements. It returns false otherwise.

**E next():** Returns the next element in the iteration. If there are no more elements, it throws a NoSuchElementException.

**void remove():** Removes the last element returned by the iterator from the underlying collection. It's an optional operation and might not be supported by all collections.

Usage:

**=>**Iterators are typically used in conjunction with collections like ArrayList, LinkedList, HashSet, etc., to traverse through the elements stored in these collections.

They provide a safe and efficient way to iterate over the elements without directly manipulating the collection itself.

**=>**Iterators are preferred over traditional index-based loops (e.g., for loop with index) when working with collections, especially for collections that do not support random access.

**Example:**

import java.util.ArrayList;

import java.util.Iterator;

public class test12 {

    public static *void* main(String[] *args*) {

        ArrayList<String> list = new ArrayList<>();

        list.add("Apple");

        list.add("Banana");

        list.add("Orange");

//m-1 to print

        for (*int* i = 0; i < list.size(); i++) {

            System.out.println(list.get(i));

        }

//m-2 to print Obtaining an iterator for ArrayList

        Iterator<String> i = list.iterator();

// Iterating over the elements using hasNext() and next() methods

        while (i.hasNext()) {

            String element = i.next();

            System.out.println(element);

        }

    }

}