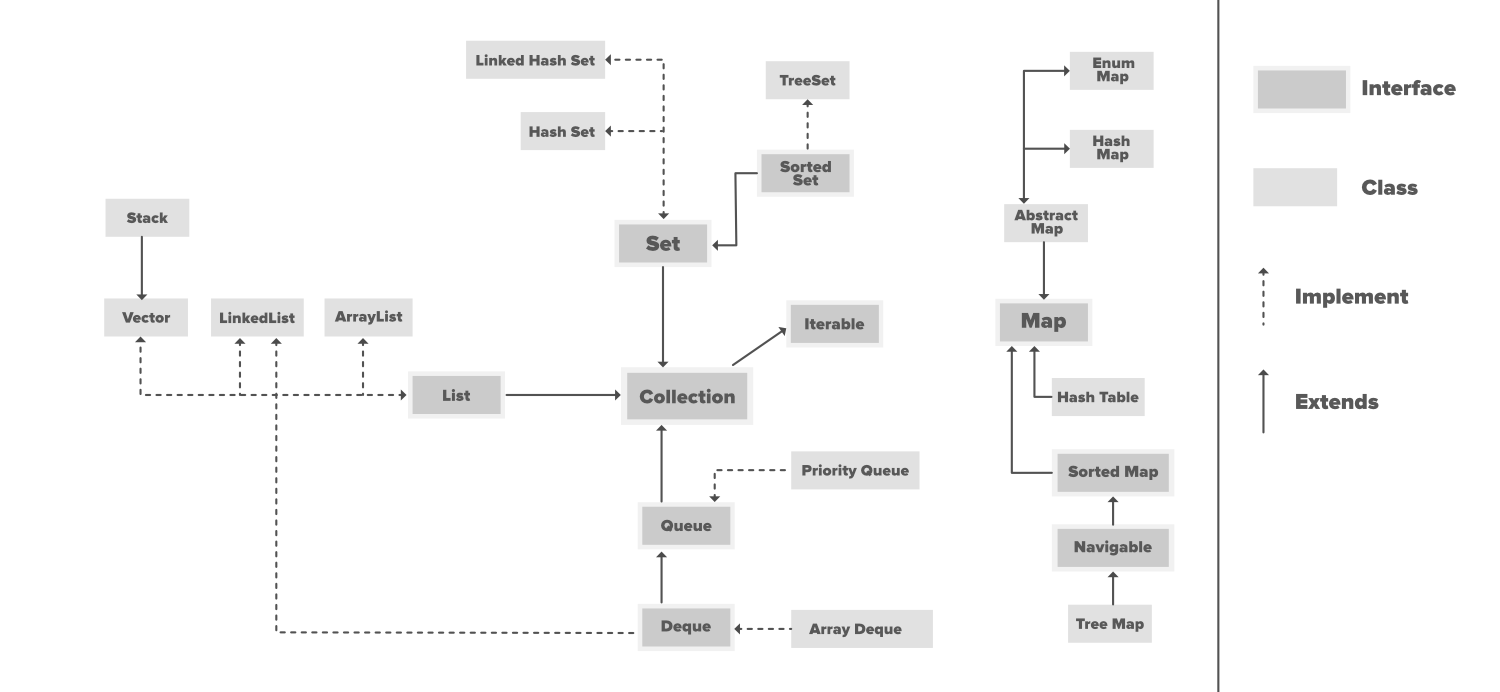
**Hierarchy of the Collection Framework :**



Any group of individual objects which are represented as a single unit is known as the collection of the objects. In Java, a separate framework named the “Collection Framework” has been defined in JDK 1.2 which holds all the collection classes and interface in it.

The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main “root” interfaces of Java collection classes.

The following are the advantages of the collection framework.

**Consistent API:** The API has a basic set of interfaces like Collection, Set, List, or Map, all the classes (ArrayList, LinkedList, Vector, etc) that implement these interfaces have some common set of methods.

**Reduces programming effort:** A programmer doesn’t have to worry about the design of the Collection but rather he can focus on its best use in his program. Therefore, the basic concept of Object-oriented programming (i.e.) abstraction has been successfully implemented.

**Increases program speed and quality:** Increases performance by providing high-performance implementations of useful data structures and algorithms because in this case, the programmer need not think of the best implementation of a specific data structure. He can simply use the best implementation to drastically boost the performance of his algorithm/program.

**What is a Framework?**

A framework is a set of **[classes](https://www.geeksforgeeks.org/classes-objects-java/)**and **[interfaces](https://www.geeksforgeeks.org/interfaces-in-java/)**which provide a ready-made architecture. In order to implement a new feature or a class, there is no need to define a framework. However, an optimal object-oriented design always includes a framework with a collection of classes such that all the classes perform the same kind of task.

Before Collection Framework(or before JDK 1.2) was introduced, the standard methods for grouping Java objects (or collections) were **[Arrays](https://www.geeksforgeeks.org/introduction-to-arrays/)**or **[Vectors](https://www.geeksforgeeks.org/java-util-vector-class-java/)**or **[Hashtables](https://www.geeksforgeeks.org/hashtable-in-java/)**. All of these collections had no common interface. Therefore, though the main aim of all the collections are same, the implementation of all these collections were defined independently and had no correlation among them. And also, its very difficult for the users to remember all the different [methods](https://www.geeksforgeeks.org/methods-in-java/), syntax and [constructors](https://www.geeksforgeeks.org/constructors-in-java/) present in every collection class.

// Accessing the first element of the

        // array, vector and hashtable

        System.out.println(arr[0]);

        System.out.println(v.elementAt(0));

        System.out.println(h.get(1));

As we can observe, none of these collections(Array, Vector or Hashtable) implements a standard member access interface, it was very difficult for programmers to write algorithms that can work for all kinds of Collections. Another drawback is that most of the ‘Vector’ methods are final, meaning we cannot extend the ’Vector’ class to implement a similar kind of Collection. Therefore, Java developers decided to come up with a common interface to deal with the above-mentioned problems and introduced the Collection Framework in JDK 1.2 post which both, legacy Vectors and Hashtables were modified to conform to the Collection Framework.

* **[Class](https://www.geeksforgeeks.org/classes-objects-java/" \t "https://www.geeksforgeeks.org/collections-in-java-2/_blank)**: A class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.
* **[Interface](https://www.geeksforgeeks.org/interfaces-in-java/)**: Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body). Interfaces specify what a class must do and not how. It is the blueprint of the class.

**List Interface in Java:**

The List interface provides a way to store the ordered collection. It is a child interface of [Collection](https://www.geeksforgeeks.org/collections-in-java-2/). It is an ordered collection of objects in which duplicate values can be stored. Since List preserves the insertion order, it allows positional access and insertion of elements.



List<Integer> l1 = **new** ArrayList<Integer>();

**ArrayList** :

**[Vector](https://www.geeksforgeeks.org/java-util-vector-class-java/" \t "https://www.geeksforgeeks.org/list-interface-java-examples/_blank)**[:](https://www.geeksforgeeks.org/java-util-vector-class-java/" \t "https://www.geeksforgeeks.org/list-interface-java-examples/_blank) Vector is a class which is implemented in the collection framework implements a growable array of objects.

Vector implements a dynamic array that means it can grow or shrink as required. Like an array,

it contains components that can be accessed using an integer index. Vectors basically fall in legacy classes but now it is fully compatible with collections. Let’s see how to create a list object using this class.

// Size of the vector

**int** n = 5;

        // Declaring the List with initial size n

        List<Integer> v = **new** Vector<Integer>(n);

        // Appending the new elements

        // at the end of the list

**for** (**int** i = 1; i <= n; i++)

            v.add(i);

        // Printing elements

        System.out.println(v);

        // Remove element at index 3

        v.remove(3);

        // Displaying the list after deletion

        System.out.println(v);

        // Printing elements one by one

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

            System.out.print(v.get(i) + " ");

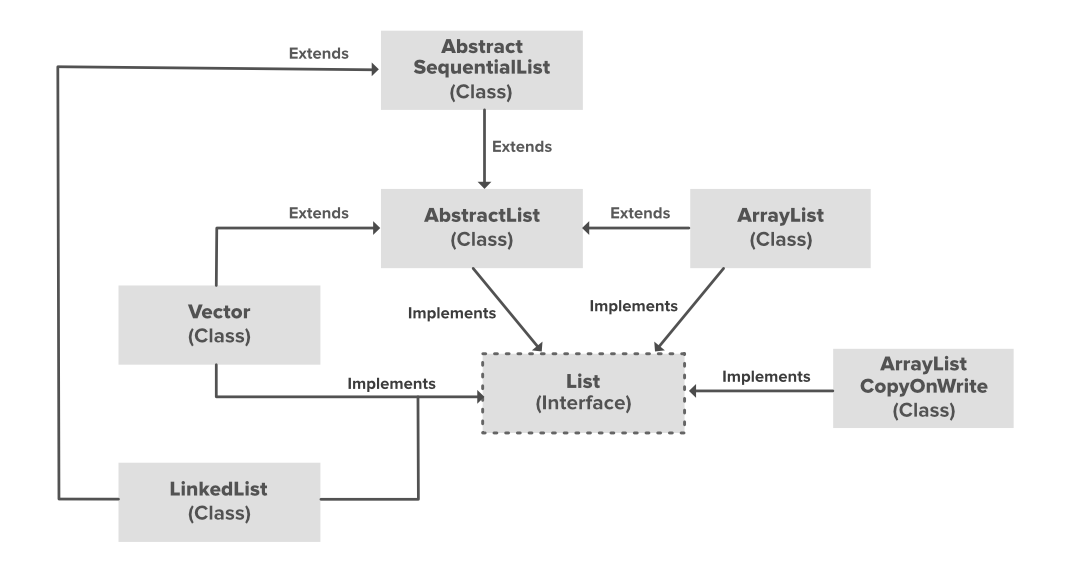
    }

**[Stack](https://www.geeksforgeeks.org/stack-class-in-java/" \t "https://www.geeksforgeeks.org/list-interface-java-examples/_blank)**[:](https://www.geeksforgeeks.org/stack-class-in-java/" \t "https://www.geeksforgeeks.org/list-interface-java-examples/_blank) Stack is a class which is implemented in the collection framework and extends the vector class models and implements the [Stack data structure](http://www.geeksforgeeks.org/stack-data-structure/). The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek. Let’s see how to create a list object using this class.

 // Declaring the List

        List<Integer> s = **new** Stack<Integer>();

**[LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/" \t "https://www.geeksforgeeks.org/list-interface-java-examples/_blank)**[:](https://www.geeksforgeeks.org/linked-list-in-java/" \t "https://www.geeksforgeeks.org/list-interface-java-examples/_blank) LinkedList is a class which is implemented in the collection framework which inherently implements the [linked list data structure](https://www.geeksforgeeks.org/data-structures/linked-list/). It is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays. Let’s see how to create a list object using this class.



**[AbstractList](https://www.geeksforgeeks.org/abstractlist-in-java-with-examples/)**, **[CopyOnWriteArrayList](https://www.geeksforgeeks.org/copyonwritearraylist-in-java/)**and the **[AbstractSequentialList](https://www.geeksforgeeks.org/abstractsequentiallist-in-java-with-examples/)**are the classes which implement the list interface. A separate functionality is implemented in each of the mentioned classes. They are:

1. **AbstractList**: This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the get() and the size() methods.
2. **CopyOnWriteArrayList**: This class implements the list interface. It is an enhanced version of [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) in which all the modifications(add, set, remove, etc.) are implemented by making a fresh copy of the list.
3. **AbstractSequentialList**: This class implements the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) and the AbstractCollection class. This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the get() and the size() methods.

### M