Collections are like containers that groups multiple items in a single unit. For example; a jar of chocolates, list of names etc. Collections are used almost in every programming language and when Java arrived, it also came with few Collection classes; Vector, Stack, Hashtable, Array. Java 1.2 provided Collections Framework that is architecture to represent and manipulate Collections in java in a standard way. Java Collections Framework consists of following parts:

• **Interfaces:** Java Collections Framework interfaces provides the abstract data type to represent collection. java.util.Collection is the root interface of Collections Framework. It is on the top of Collections framework hierarchy. It contains some important methods such as size(), iterator(), add(), remove(), clear() that every Collection class must implement. Some other important interfaces are java.util.List, java.util.Set, java.util.Queue and java.util.Map. Map is the only interface that doesn’t inherits from Collection interface but it’s part of Collections framework. All the collections framework interfaces are present in java.util package.

• **Implementation Classes:** Collections in Java provides core implementation classes for collections. We can use them to create different types of collections in java program. Some important collection classes are ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet.These classes solve most of our programming needs but if we need some special collection class, we can extend them to create our custom collection class. Java 1.5 came up with thread-safe collection classes that allowed to modify Collections while iterating over it, some of them are CopyOnWriteArrayList, ConcurrentHashMap, CopyOnWriteArraySet. These classes are in java.util.concurrent package. All the collection classes are present in java.util and java.util.concurrent package.

**• Algorithms**: Algorithms are useful methods to provide some common functionalities, for example searching, sorting and shuffling. Below class diagram shows Collections Framework hierarchy. For simplicity I have included only commonly used interfaces and classes

Benefits of Java Collections Framework Java Collections framework have following benefits:

• Reduced Development Effort – It comes with almost all common types of collections and useful methods to iterate and manipulate the data. So we can concentrate more on business logic rather than designing our collection APIs.

• Increased Quality – Using core collection classes that are well tested increases our program quality rather than using any home developed data structure.

• Reusability and Interoperability

• Reduce effort – to learn any new API if we use core collection API classes. o Java Collections Interfaces Java collection interfaces are the foundation of the Java Collections Framework. Note that all the core collection interfaces are generic; for example public interface Collection. The syntax is for Generics and when we declare Collection, we should use it to specify the type of Object it can contain. It helps in reducing run-time errors by type-checking the Objects at compile-time. To keep the number of core collection interfaces manageable, the Java platform doesn’t provide separate interfaces for each variant of each collection type. If an unsupported operation is invoked, a collection implementation throws an UnsupportedOperationException.

**Collection Interface** This is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn’t provide any direct implementations of this interface. The interface has methods to tell you how many elements are in the collection (size, isEmpty), to check whether a given object is in the collection (contains), to add and remove an element from the collection (add, remove), and to provide an iterator over the collection (iterator)

Collection interface also provides bulk operations methods that work on entire collection – containsAll, addAll, removeAll, retainAll, clear. The toArray methods are provided as a bridge between collections and older APIs that expect arrays on input.

**Iterator Interface:** Iterator interface provides methods to iterate over any Collection. We can get iterator instance from a Collection using iterator method. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection during the iteration. Iterators in collection classes implement Iterator Design Pattern.

Set Interface: Set is a collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards. The Java platform contains three general-purpose Set implementations: HashSet, TreeSet, and LinkedHashSet. Set interface doesn’t allow random-access to an element in the Collection. You can use iterator or foreach loop to traverse the elements of a Set.

List Interface: List is an ordered collection and can contain duplicate elements. You can access any element from it’s index. List is more like array with dynamic length. List is one of the most used Collection type. ArrayListand LinkedList are implementation classes of List interface. List interface provides useful methods to add an element at specific index, remove/replace element based on index and to get a sub-list using index. Collections class provide some useful algorithm for List – sort, shuffle, reverse, binarySearch etc.

Queue Interface: Queue is a collection used to hold multiple elements prior to processing. Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations. Queues typically, but do not necessarily, order elements in a FIFO (first-in, first-out) manner. Among the exceptions are priority queues, which order elements according to a supplied comparator or the elements’ natural ordering. Whatever the ordering used, the head of the queue is the element that would be removed by a call to remove or poll. In a FIFO queue, all new elements are inserted at the tail of the queue.

Dequeue Interface

A linear collection that supports element insertion and removal at both ends. The name deque is short for “double ended queue” and is usually pronounced “deck”. Most Deque implementations place no fixed limits on the number of elements they may contain, but this interface supports capacity-restricted deques as well as those with no fixed size limit. This interface defines methods to access the elements at both ends of the deque. Methods are provided to insert, remove, and examine the element.

Map Interface: Java Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value. The Java platform contains three general-purpose Map implementations: HashMap, TreeMap, and LinkedHashMap. The basic operations of Map are put, get, containsKey, containsValue, size, and isEmpty.

ListIterator Interface :An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator’s current position in the list. A ListIterator has no current element; its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next(). SortedSet Interface: SortedSet is a Set that maintains its elements in ascending order. Several additional operations are provided to take advantage of the ordering. Sorted sets are used for naturally ordered sets, such as word lists and membership rolls.

SortedMap Interface: Map that maintains its mappings in ascending key order. This is the Map analog of SortedSet. Sorted maps are used for naturally ordered collections of key/value pairs, such as dictionaries and telephone directories.

Java Collections Classes

Java Collections framework comes with many implementation classes for the interfaces. Most common implementations are ArrayList, HashMap and HashSet. Java 1.5 included Concurrent implementations; for example ConcurrentHashMap and CopyOnWriteArrayList. Usually Collection classes are not thread-safe and their iterator is fail-fast. In this section, we will learn about commonly used collection classes.

HashSet Class: