Introduction to Collection

Agenda

1 **Introduction to Collection**

Objectives

At the end of this module, you will be able to:

- Understand what is Collection framework and its usage
- Understand the hierarchy of Collection Framework

Introduction to Collection

Introduction

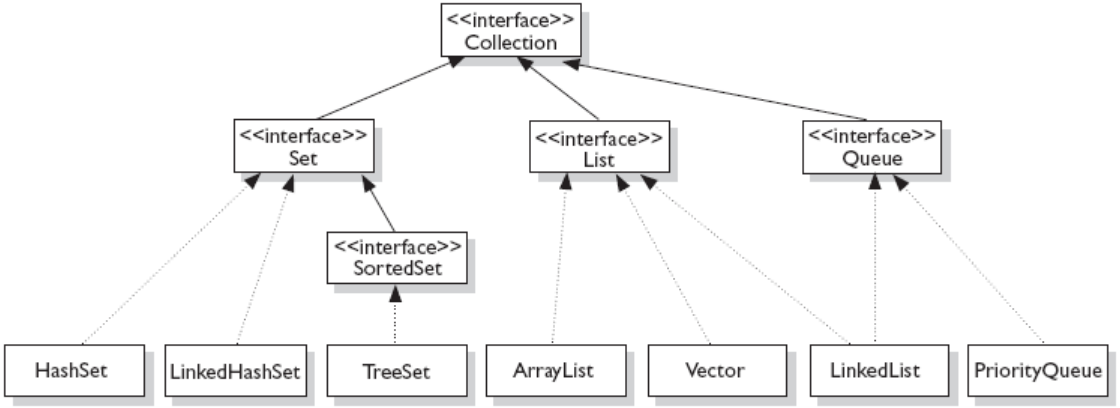
- A Collection is a group of objects
- Collections framework provide a a set of standard utility classes to manage collections
- Collections Framework consists of three parts:
  - Core Interfaces
  - Concrete Implementation
  - Algorithms such as searching and sorting

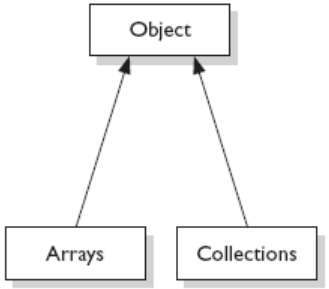
Advantages of Collections

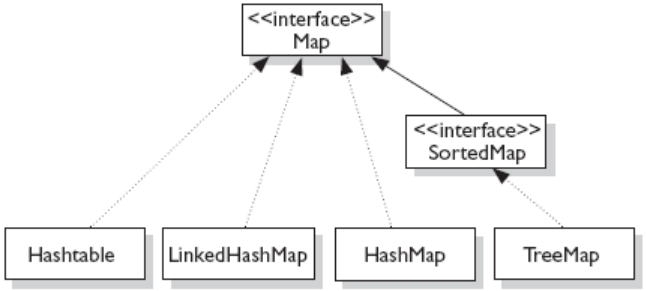
- **Reduces programming effort :** by providing useful data structures and algorithms so you don't have to write them yourself.
- **Increases performance :**by providing high-performance implementations of useful data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be easily tuned by switching implementations.
- **Provides interoperability between unrelated APIs:** by establishing a common language to pass collections back and forth.
- **Reduces the effort required to learn APIs:** by eliminating the need to learn multiple ad hoc collection APIs.
- **Reduces the effort required to design and implement APIs:** by eliminating the need to produce ad hoc collections APIs.
- **Fosters Software reuse:** by providing a standard interface for collections and algorithms to manipulate them.

Collection Hierarchy

Interfaces and their implementation



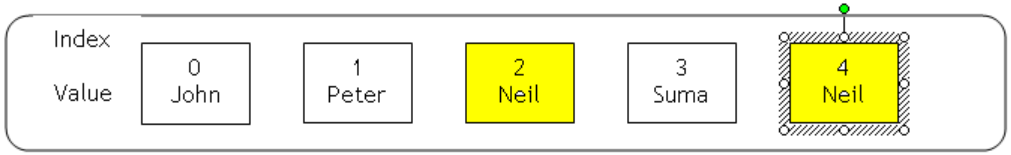




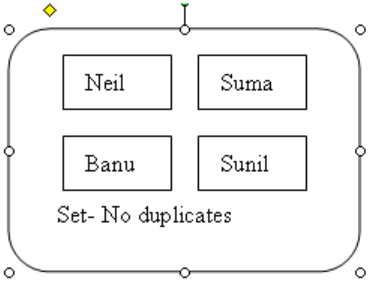
Collection Interfaces

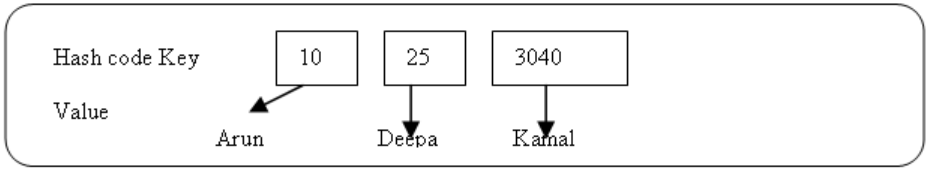
Interfaces	Description
Collection	A basic interface that defines the operations that all the classes that maintain collections of objects typically implement.
Set	Extends the Collection interface for sets that maintain unique element.
SortedSet	Augments the Set interface or Sets that maintain their elements in sorted order.
List	Collections that require position-oriented operations should be created as lists. Duplicates are allowed.
Queue	Things arranged by the order in which they are to be processed.
Map	A basic interface that defines operations that classes that represent mapping of keys to values typically implement.
SortedMap	Extends the Map interface for maps that maintain their mappings in the key order.

Illustration of List, Set and Map



List - Duplicates allowed





Hash Map - Key generated from RollNo

Collection Implementations

		Implementations				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

Collection Implementations

- Classes that implement the collection interfaces typically have names of the form <*Implementation-style*><*Interface*>. The general purpose implementations are summarized in the table above.
- **HashSet** A HashSet is an unsorted, unordered Set. It uses the hashcode of the object being inserted, so the more efficient your hashCode() implementation the better access performance you'll get. Use this class when you want a collection with no duplicates and you don't care about order when you iterate through it.
- **TreeSet** A TreeSet stores objects in a sorted sequence. It stores its elements in a tree and they are automatically arranged in a sorted order.

Collection Implementations

- **LinkedHashSet** A LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements. Use this class instead of HashSet when you care about the iteration order. When you iterate through a HashSet the order is unpredictable, while a LinkedHashSet lets you iterate through the elements in the order in which they were inserted.
- **ArrayList** Think of this as a growable array. It gives you fast iteration and fast random access. It is an ordered collection (by index), but not sorted. ArrayList now implements the new RandomAccess interface—a marker interface (meaning it has no methods) that says, "this list supports fast (generally constant time) random access." Choose this over a LinkedList when you need fast iteration but aren't as likely to be doing a lot of insertion and deletion.
- **LinkedList** A LinkedList is ordered by index position, like ArrayList, except that the elements are doubly-linked to one another.

Collection interface methods

Method	Description
`int size();`	Returns number of elements in collection.
`boolean isEmpty();`	Returns true if invoking collection is empty.
`boolean contains(Object element);`	Returns true if element is an element of invoking collection.
`boolean add(Object element);`	Adds element to invoking collection.
`boolean remove(Object element);`	Removes one instance of element from invoking collection
`Iterator iterator();`	Returns an iterator fro the invoking collection
`boolean containsAll(Collection c);`	Returns true if invoking collection contains all elements of *c*; false otherwise.
`boolean addAll(Collection c);`	Adds all elements of *c* to the invoking collection.
`boolean removeAll(Collection c);`	Removes all elements of *c* from the invoking collection
`boolean retainAll(Collection c);`	Removes all elements from the invoking collection except those in *c*.
`void clear();`	Removes all elements from the invoking collection
`Object[] toArray();`	Returns an array that contains all elements stored in the invoking collection
`Object[] toArray(Object a[]);`	Returns an array that contains only those collection elements whose type matches that of *a*.

Quiz

1. Which of the following classes is not implemented from the Collection interface?
   a. TreeSet
   b. HashTable
   c. Vector
   d. Linked List

2. Which of the following is a class?
   a. Collection
   b. Collections

3. Which of the following does not accept duplicate values?
   a. ArrayList
   b. LinkedList
   c. TreeSet
   d. Vector

Summary

- Introduction to Collection

Thank You