**Module 21 - Collections**

**1. Module Introduction**

Learning Objectives

* Introduction
* Hierarchy of Collection Framework
* Methods of Collection interface

Introduction

To understand this Module, you have to be familiar with:

Polymorphism, especially the upcasting and downcasting operations.

Interfaces, abstract methods and their implementations.

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

Although we can use an array to store a group of elements of the same type (either primitives or objects). The array, however, does not support so-called dynamic allocation - it has a fixed length which cannot be changed once allocated. Furthermore, array is a simple linear structure. Many applications may require more complex data structure such as linked list, stack, hash table, sets, or trees.

In Java, dynamically allocated data structures (such as ArrayList, LinkedList, Vector, Stack, HashSet, HashMap, Hashtable) are supported in a unified architecture called the Collection Framework, which mandates the common behaviors of all the classes.

A collection, as its name implied, is simply an object that holds a collection (or a group, a container) of objects. Each item in a collection is called an element. A framework, by definition, is a set of interfaces that force you to adopt some design practices. A well-designed framework can improve your productivity and provide ease of maintenance.

The collection framework provides a unified interface to store, retrieve and manipulate the elements of a collection, regardless of the underlying and actual implementation. This allows the programmers to program at the interfaces, instead of the actual implementation.

The Java Collection Framework package (java.util) contains:

* A set of interfaces
* Implementation classes, and
* Algorithms (such as sorting and searching).

Benefits of the Java Collections Framework

**The Java Collections Framework provides the following benefits:**

* **Reduces programming effort:**

By providing useful data structures and algorithms, the Collections Framework frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work. By facilitating interoperability among unrelated APIs, the Java Collections Framework frees you from writing adapter objects or conversion code to connect APIs.

* **Increases program speed and quality:**

This Collections Framework provides high-performance, high-quality implementations of useful data structures and algorithms. The various implementations of each interface are interchangeable, so programs can be easily tuned by switching collection implementations. Because you're freed from the drudgery of writing your own data structures, you'll have more time to devote to improving programs' quality and performance.

* **Allows interoperability among unrelated APIs:**

The collection interfaces are the vernacular by which APIs pass collections back and forth. If my network administration API furnishes a collection of node names and if your GUI toolkit expects a collection of column headings, our APIs will interoperate seamlessly, even though they were written independently.

* **Reduces effort to learn and to use new APIs:**

Many APIs naturally take collections on input and furnish them as output. In the past, each such API had a small sub-API devoted to manipulating its collections. There was little consistency among these ad hoc collections sub-APIs, so you had to learn each one from scratch, and it was easy to make mistakes when using them. With the advent of standard collection interfaces, the problem went away.

* **Reduces effort to design new APIs:**

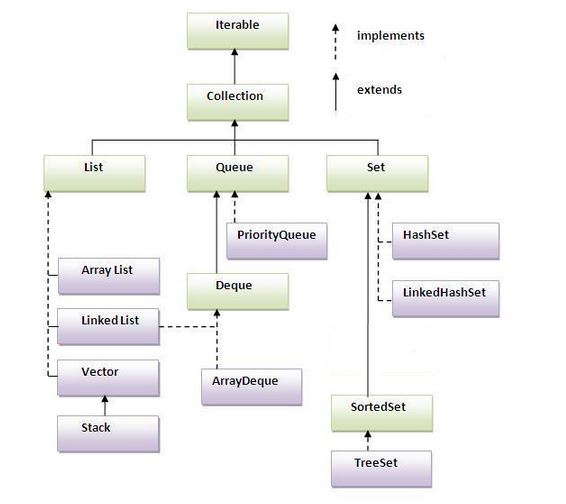
This is the flip side of the previous advantage. Designers and implementers don't have to reinvent the wheel each time they create an API that relies on collections; instead, they can use standard collection interfaces.

* **Fosters software reuse:**

New data structures that conform to the standard collection interfaces are by nature reusable. The same goes for new algorithms that operate on objects that implement these interfaces.

Hierarchy of Collection Framework

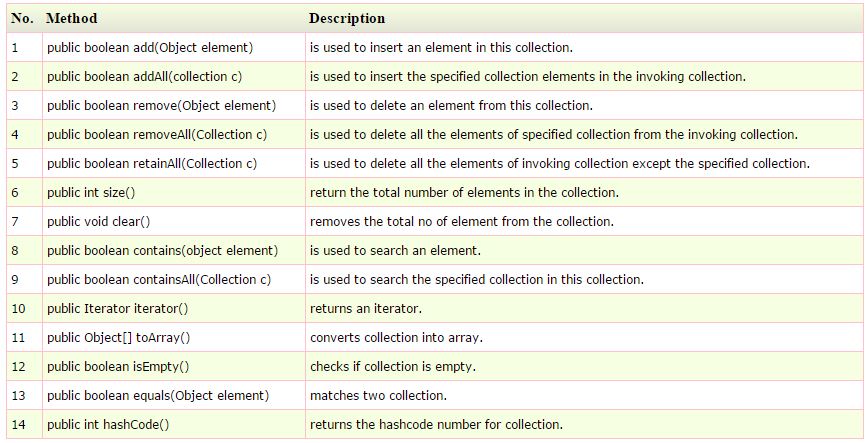
Let us see the hierarchy of collection framework.The java.util package contains all the classes and interfaces for Collection framework.

[](http://wiki2016.msitprogram.net/index.php/File:Hier.JPG)

### 2. Resources

## Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

[](http://wiki2016.msitprogram.net/index.php/File:Methods.JPG)

## Retrieving Elements from Collections

Following are the 4 ways to retrieve any elements from a collection object:

* Using for-each loop
* Using Iterator Interface
* Using ListIterator Interface
* Using Enumeration Interface

#### for-each Loop

for-each loop is like for loop which repeatedly executes a group of statements for each element of the collection.

The format is:

for(variable:collection\_object)

{

statement;

}

#### Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only. Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

* public boolean hasNext() it returns true if iterator has more elements.
* public object next() it returns the element and moves the cursor pointer to the next element.
* public void remove() it removes the last elements returned by the iterator. It is rarely used.

#### ListIterator interface

ListIterator interface provides the facility of iterating the elements in both forward and reverse direction.

There are four methods:

* public boolean hasNext() it returns true if iterator has more elements.
* public object next() it returns the element and moves the cursor pointer to the next element.
* public object previous() it returns the element and moves the cursor pointer to the previous element.
* public void remove() it removes the last elements returned by the iterator. It is rarely used.

#### Enumeration Interface

Enumeration interface provides the facility of iterating the elements in forward direction only.

There are 2 methods:

* boolean hasMoreElements() : It tests if the Enumerator has anymore elements or not
* object nextElement() : It returns the next element available in Enumeration

## Video Resource

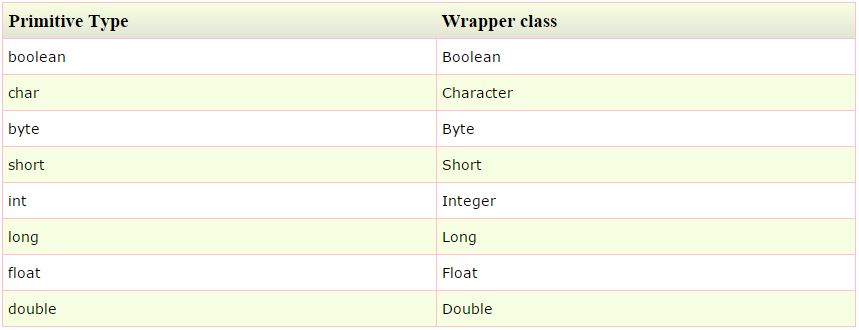
## <https://youtu.be/d3QbptJRln4>

## Wrapper class in Java

Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.

Since J2SE 5.0, autoboxing and unboxing feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.

One of the eight classes of java.lang package are known as wrapper class in java. The list of eight wrapper classes are given below:

[](http://wiki2016.msitprogram.net/index.php/File:Wrapper.JPG)

##### Note: Java Collections use Wrapper Classes only.

## ArrayList class

* Java ArrayList class uses a dynamic array for storing the elements.It extends AbstractList class and implements List interface.
* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.

In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

## Java Non-generic Vs Generic Collection

* Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.
* Java new generic collection allows you to have only one type of object in collection.Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

ArrayList al=new ArrayList();//creating old non-generic arraylist

Let's see the new generic example of creating java collection.

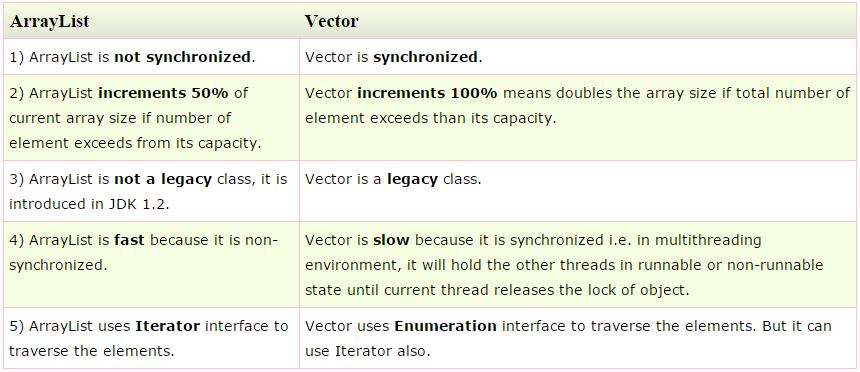
ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives compile time error.

## Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.

[](http://wiki2016.msitprogram.net/index.php/File:Array_vector.JPG)

Let's see a simple example where we are using ArrayList to store and traverse the elements.

import java.util.\*;

class TestArrayList21{

public static void main(String args[]){

List<String> al=new ArrayList<String>();//creating arraylist

al.add("Sonoo");//adding object in arraylist

al.add("Michael");

al.add("James");

al.add("Andy");

//traversing elements using Iterator

Iterator itr=al.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Let's see a simple example of java Vector class that uses Enumeration interface.

import java.util.\*;

class TestVector1{

public static void main(String args[]){

Vector<String> v=new Vector<String>();//creating vector

v.add("umesh");//method of Collection

v.addElement("irfan");//method of Vector

v.addElement("kumar");

//traversing elements using Enumeration

Enumeration e=v.elements();

while(e.hasMoreElements()){

System.out.println(e.nextElement());

}

}

}

Java Hashtable class

A Hashtable is an array of list.Each list is known as a bucket.The position of bucket is identified by calling the hashcode() method.A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.

* It contains only unique elements.
* It may have not have any null key or value.
* It is synchronized.

Example of Hashtable:

import java.util.\*;

class TestCollection16{

public static void main(String args[]){

Hashtable<Integer,String> hm=new Hashtable<Integer,String>();

hm.put(100,"Amit");

hm.put(102,"Ravi");

hm.put(101,"Vijay");

hm.put(103,"Rahul");

for(Map.Entry m:hm.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}