# Collections in Java

1. [Java Collection Framework](http://www.javatpoint.com/collections-in-java)
2. [Hierarchy of Collection Framework](http://www.javatpoint.com/collections-in-java#collectionhierarchy)
3. [Collection interface](http://www.javatpoint.com/collections-in-java#collectionmethods)
4. [Iterator interface](http://www.javatpoint.com/collections-in-java#collectioniterator)

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

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

#### What is Collection in java

Collection represents a single unit of objects i.e. a group.

#### What is framework in java

* provides readymade architecture.
* represents set of classes and interface.
* is optional.

#### What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

* **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
* **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
* **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

### Methods of Collection interface

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

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(Object element) | is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection c) | is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | is used to delete an element from this collection. |
| 4 | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection from the invoking collection. |
| 5 | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection except the specified collection. |
| 6 | public int size() | return the total number of elements in the collection. |
| 7 | public void clear() | removes the total no of element from the collection. |
| 8 | public boolean contains(Object element) | is used to search an element. |
| 9 | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator() | returns an iterator. |
| 11 | public Object[] toArray() | converts collection into array. |
| 12 | public boolean isEmpty() | checks if collection is empty. |
| 13 | public boolean equals(Object element) | matches two collection. |
| 14 | public int hashCode() | returns the hashcode number for collection. |

### 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:

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

Generics:-

It would be nice if we could write a single sort method that could sort the elements in an Integer array, a String array or an array of any type that supports ordering.

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods or, with a single class declaration, a set of related types, respectively.

Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Using Java Generic concept, we might write a generic method for sorting an array of objects, then invoke the generic method with Integer arrays, Double arrays, String arrays and so on, to sort the array elements.

Generic Methods:

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of the arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods:

* All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type ( < E > in the next example).
* Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
* The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
* A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char)

Example:

Following example illustrates how we can print array of different type using a single Generic method:

public class GenericMethodTest

{

// generic method printArray

public static < E > void printArray( E[] inputArray )

{

// Display array elements

for ( E element : inputArray ){

System.out.printf( "%s ", element );

}

System.out.println();

}

public static void main( String args[] )

{

// Create arrays of Integer, Double and Character

Integer[] intArray = { 1, 2, 3, 4, 5 };

Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };

Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

System.out.println( "Array integerArray contains:" );

printArray( intArray ); // pass an Integer array

System.out.println( "\nArray doubleArray contains:" );

printArray( doubleArray ); // pass a Double array

System.out.println( "\nArray characterArray contains:" );

printArray( charArray ); // pass a Character array

}

}

This would produce the following result:

Array integerArray contains:1 2 3 4 5 6

Array doubleArray contains:1.1 2.2 3.3 4.4

Array characterArray contains:H E L L O