CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics
Overview

Introduction: Collections framework in Java

Collections framework in Java is one of the most valuable and interesting topics of Java language. How important it is?

Without using collection concepts, you cannot develop any production level software application in Java. Therefore, every Java learner has to learn the collections framework.

Introduction: Collections framework in Java

According to the Oracle document, among all the java concepts, Java collections framework is the most important concept for developing the project and crack interviews also.

An interviewer often asks questions to check whether the interviewees understand the correct usage of collection classes and is aware of available alternative solutions.

Introduction: Collections framework in Java

In this Java collections framework lecture, we will cover the following topics in depth.

- What is Collection in Java?
- Realtime examples of Collection
- Types of objects stored in Collection (container) object
- What is the need for Collections in Java?
- What is Collections Framework in Java?
- Difference between Array and Collection in Java and many more.

What is Collection in Java

Technically, a collection is an object or container which stores a group of other objects as a single unit or single entity. Therefore, it is also known as container object or collection object in java.

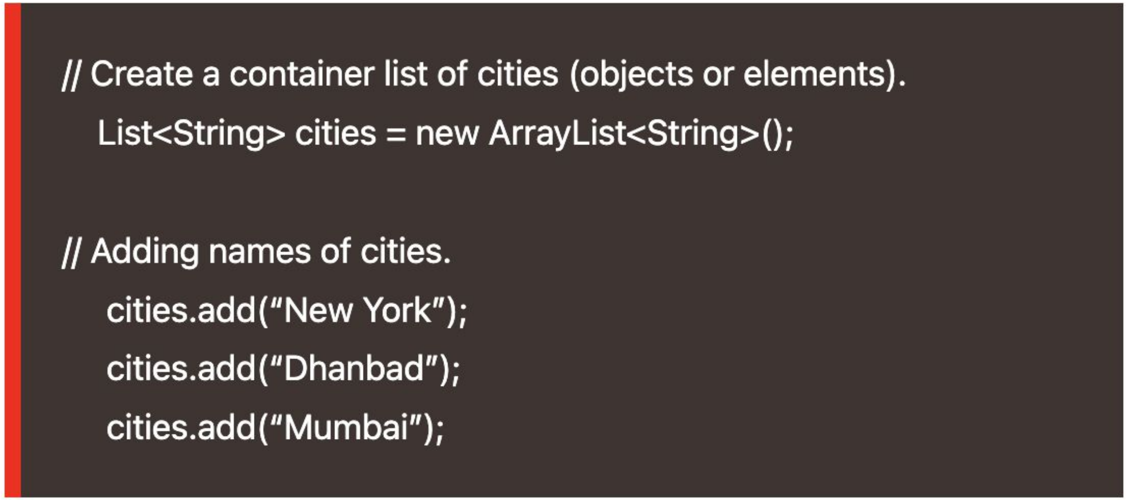
A container object means it contains other objects. In simple words, a collection is a container that stores multiple elements together.

What is Collection in Java

A collection object has a class that is known as collection class or container class. All collection classes are present in **java.util package**. Here, util stands for utility. A group of collection classes is called collections framework in java.

What is Collection in Java

A simple example of a collection is given below:



```
// Create a container list of cities (objects or elements).
    List<String> cities = new ArrayList<String>();

// Adding names of cities.
     cities.add(“New York”);
     cities.add(“Dhanbad”);
     cities.add(“Mumbai”);
```

Types of Objects Stored in Collection (Container) Object

There are two types of objects that can be stored in a collection or container object. They are as follows:

- Homogeneous objects
- Heterogeneous objects

Types of Objects Stored in Collection (Container) Object

1. Homogeneous objects:

"Homo-" means "same". Homogeneous objects are a group of multiple objects that belong to the same class.

For example, suppose we have created three objects Student s1, Student s2, and Student s3 of the same class ‘Student’. Since these three objects belong to the same class that’s why they are called homogeneous objects.

Types of Objects Stored in Collection (Container) Object

2. Heterogeneous objects:

"Hetero-" means "different". Heterogeneous objects are a group of different objects that belong to different classes.

For example, suppose we have created two different objects of different classes such as one object Student s1, and another one object Employee e1. Here, student and employee objects together are called a collection of heterogeneous objects.

Types of Objects Stored in Collection (Container) Object

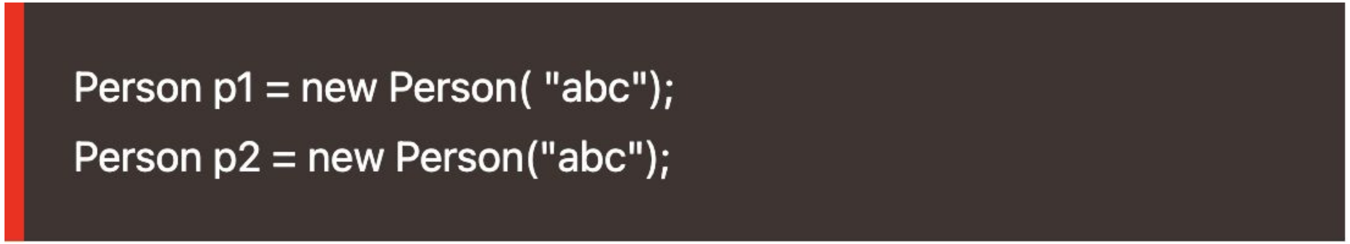
These objects can also be further divided into two types. They are as follows:

1. Duplicate objects
2. Unique objects

Types of Objects Stored in Collection (Container) Object

1. Duplicate objects:

The multiple objects of a class that contains the same data are called duplicate objects. For example, suppose we create two person objects Person p1 and Person p2. Both of these objects have the same data.



```
Person p1 = new Person( "abc");
Person p2 = new Person("abc");
```

Since the above two objects have the same data “abc” therefore, these are called duplicate objects.

Types of Objects Stored in Collection (Container) Object

2. Unique objects:

The multiple objects of a class that contains different data are called unique objects. For example:

```
Person p1 = new Person("abcd");
Person p2 = new Person("abcde");
```

A unique or duplicate object depends on its internal data.

What is need for Collections in Java

Before going to understand the need for collections in java, first, we understand different ways to store values in Java application by JVM. There is a total of four ways to store values in Java application by JVM.

1. Using a variable approach
2. Using a class object approach
3. Using an array object approach
4. Using a collection object approach

What is need for Collections in Java

Limitations of variables:

Thus, the limitations of using the variable approach are as follows:

➲ The limitation of a variable is that it can store only one value at a time.

➲ The readability and reusability of the code will be down.

➲ JVM will take more time for execution.

```
int x = 10;
int y = 20;
```

What is need for Collections in Java

Limitations of object approach:

This approach is only suitable to store a fixed number of different values.

```
class Employee {
     int eNo;
     String eName;
}
```

```
Employee e1 = new Employee();
```

What is need for Collections in Java

Limitations of array approach:

1. We can only store multiple “fixed” numbers of values of homogeneous data type. We cannot store different class objects into the same array. This is because an array can store only one data type of elements (objects).

2. An array is static in nature. It is fixed in length and size. We cannot change (increase/decrease) the size of the array based on our requirements once they created. Hence, to use an array, we must know the size of an array to store a group of objects in advance, which may not always be possible.

3. We can add elements at the end of an array easily. But, adding and deleting elements or objects in the middle of array is difficult.

4. We cannot insert elements in some sorting order using array concept because array does not support any method. We will have to write the sorting code for this but in the case of collection, ready-made method support is available for sorting using Tree set.

5. We cannot search a particular element using an array, whether the particular element is present or not in the array index. For this, we will have to write the searching code using array but in the case of collection, one readymade method called contains() method is available.

What is need for Collections in Java

Using collection object approach:

By using collection object, we can store the same or different data without any size limitation. Thus, technically, we can define the collections as:

A **collection in java** is a container object that is used for storing multiple homogeneous and heterogeneous, duplicate, and unique elements without any size limitation.

What is Collections Framework in Java?

A framework in java is a set of several classes and interfaces which provide a ready-made architecture.

A Java collections framework is a sophisticated hierarchy of several predefined interfaces and implementation classes that can be used to handle a group of objects as a single entity.

In simple words, a collections framework is a class library to handle groups of objects. It is present in java.util package. It allows us to store, retrieve, and update a group of objects.

What is Collections Framework in Java?

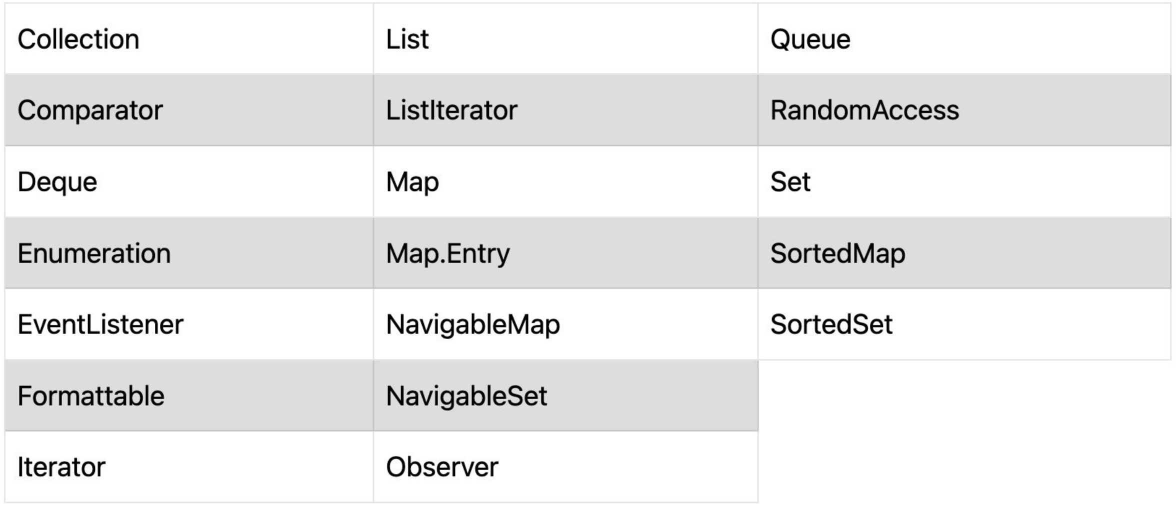
Collections framework in Java supports two types of containers:

- One for storing a collection of elements (objects), that is simply called a collection.
- The other, for storing key/value pairs, which is called a map.

The java collections framework provides an API to work with data structures such as lists, trees, sets, and maps.

What is Collections Framework in Java?

List of Interfaces defined in java.util package



Collection	List	Queue
Comparator	ListIterator	RandomAccess
Deque	Map	Set
Enumeration	Map.Entry	SortedMap
EventListener	NavigableMap	SortedSet
Formattable	NavigableSet	
Iterator	Observer	

What is Collections Framework in Java?

List of classes defined in java.util package

AbstractCollection	EventObject	Random
AbstractList	FormattableFlags	ResourceBundle
AbstractMap	Formatter	Scanner
AbstractQueue	AbstractSequentialList	HashMap
AbstractSet	HashSet	Stack
ArrayDeque	Hashtable	StringTokenizer
ArrayList	LinkedList	Vector
Collections	EnumMap	EnumSet
Calender	LinkedHashMap	TreeMap

Difference between Arrays & Collections in Java

The difference between arrays and collections are as follows:

1. Arrays are fixed in size but collections are growable in nature. We can increase or decrease size.
2. Arrays are not recommended to use with respect to memory whereas collections are recommended to use with respect to memory.
3. Arrays are recommended to use with respect to performance but collections are not recommended to use with respect to performance.
4. Arrays can store only homogeneous data elements (similar type of data) but collections can hold both homogeneous and heterogeneous elements.
5. Arrays do not support any method but collections support various kinds of methods.
6. Arrays can have both hold primitives and object types but collections can hold only objects but not primitive.

Advantage of Collections Framework in Java

The advantages of the collections framework in java are as follows:

1. The collections framework reduces the development time and the burden of designers, programmers, and users.
2. Your code is easier to maintain because it provides useful data structure and interfaces which reduce programming efforts.
3. The size of the container is growable in nature.
4. It implements high-performance of useful data structures and algorithms that increase the performance.
5. It enables software reuse.

Limitation of Collections Framework in Java

There are two limitations of the collections framework in Java. They are as follows:

1. Care must be taken to use the appropriate cast operation.
2. Compile type checking is not possible. *A fix for this is a type parameter a.k.a generics*

Q. Does a collection object store copies of other objects?

A: No, a collection object works with reference types. It stores references of other objects, not copies of other objects.

Q. Can we store a primitive data type into a collection?

A: No, collections store only objects. *Fixed with autoboxing/unboxing*

Collection Hierarchy in Java | Collection Interface

The hierarchy of the entire collection framework consists of four core interfaces such as Collection, List, Set, Map, and two specialized interfaces named SortedSet and SortedMap for sorting.

All the interfaces and classes for the collection framework are located in java.util package.

Collection Hierarchy in Java | Collection Interface

The diagram of Java collection hierarchy is shown in the rightmost figure.

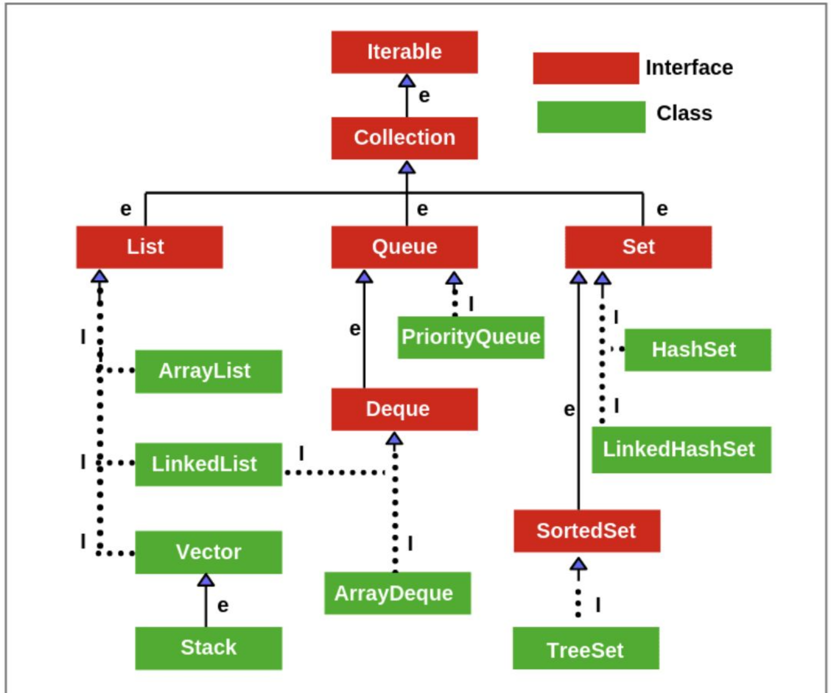


Fig: Collection Hierarchy in Java

e→ extends, I→ implements

Collection Interface in Java

➲ The basic interface of the collections framework is the Collection interface which is the root interface of all collections in the API (Application programming interface).

It is placed at the top of the collection hierarchy in java. It provides the basic operations for adding and removing elements in the collection.

➲ The Collection interface extends the Iterable interface. The iterable interface has only one method called iterator(). The function of the iterator method is to return the iterator object. Using this iterator object, we can iterate over the elements of the collection.

➲ List, Queue, and Set have three component which extends the Collection interface. A map is not inherited by Collection interface.

List Interface

- ➲ This interface represents a collection of elements whose elements are arranged sequentially ordered.
- ➲ List maintains an order of elements means the order is retained in which we add elements, and the same sequence we will get while retrieving elements.
- ➲ We can insert elements into the list at any location. The list allows storing duplicate elements in Java.
- ➲ ArrayList, vector, and LinkedList are three concrete subclasses that implement the list interface.

Set Interface

➲ This interface represents a collection of elements that contains unique elements. i.e, It is used to store the collection of unique elements.

➲ Set interface does not maintain any order while storing elements and while retrieving, we may not get the same order as we put elements. All the elements in a set can be in any order.

➲ Set does not allow any duplicate elements.

➲ HashSet, LinkedHashSet, TreeSet classes implements the set interface and sorted interface extends a set interface.

➲ It can be iterated by using Iterator but cannot be iterated using ListIterator.

SortedSet Interface

➲ This interface extends a set whose iterator transverse its elements according to their natural ordering.

➲ TreeSet implements the sorted interface.

Queue Interface

➲ A queue is an ordered of the homogeneous group of elements in which new elements are added at one end(rear) and elements are removed from the other end(front). Just like a queue in a supermarket or any shop.

➲ This interface represents a special type of list whose elements are removed only from the head.

➲ LinkedList, Priority queue, ArrayQueue, Priority Blocking Queue, and Linked Blocking Queue are the concrete subclasses that implement the queue interface.

Deque Interface

➲ A deque (double-ended queue) is a sub-interface of queue interface. It is usually pronounced “deck”.

➲ This interface was added to the collection framework in Java SE 6.

➲ Deque interface extends the queue interface and uses its method to implement deque. The hierarchy of the deque interface is shown in the below figure.

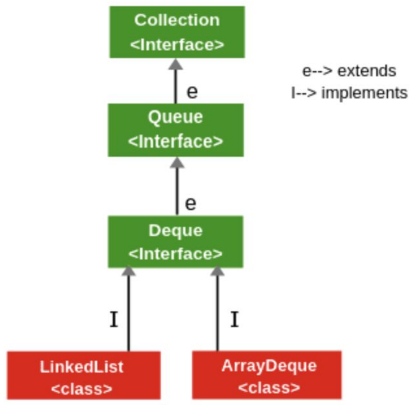


Fig: Deque Hierarchy

Deque Interface

➲ It is a linear collection of elements in which elements can be inserted and removed from either end. i.e, it supports insertion and removal at both ends of an object of a class that implements it.

➲ LinkedList and ArrayDeque classes implement the Deque interface.

Map Interface

➲ Map interface is not inherited by the collection interface. It represents an object that stores and retrieves elements in the form of a Key/Value pairs and their location within the Map are determined by a Key.

The hierarchy of the map interface is shown in the below figure.

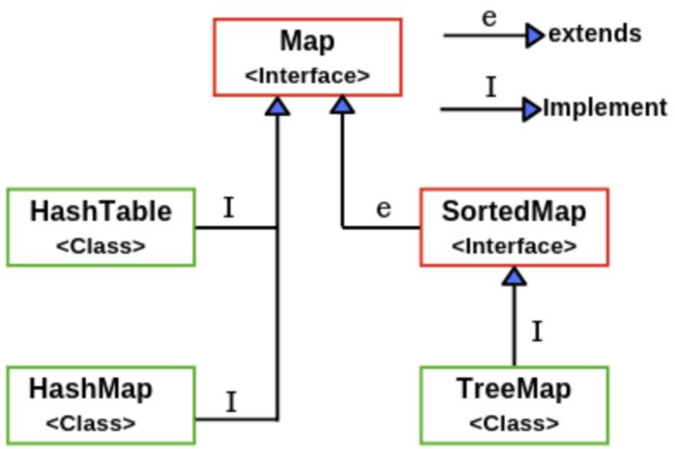


Fig: Map Hierarchy

Map Interface

- ➲ Map uses a hashing technique for storing key-value pairs.
- ➲ It doesn’t allow to store the duplicate keys but duplicate values are allowed.
- ➲ HashMap, HashTable, LinkedHashMap, TreeMap classes implements Map interface.

SortedMap Interface

➲ This interface represents a Map whose elements are stored in their natural ordering. It extends the Map interface which in turn is implemented by TreeMap classes.

Methods of Collection Interface in Java

The Collection interface consists of a total of fifteen methods for manipulating elements in the collection. They are as follows:

1. add()
2. addAll()
3. clear()
4. contains()
5. containsAll()
6. equals()
7. hashCode()
8. isEmpty()
9. iterator()
10. remove()
11. removeAll()
12. retainAll()
13. size()
14. toArray()
15. Object[] toArray()

Methods of Collection Interface in Java

1. **add():** This method is used to add or insert an element in the collection. The general syntax for add() method is as follow:

```
add(Object element) : boolean
```

If the element is added to the collection, it will return true otherwise false, if the element is already present and the collection doesn’t allow duplicates.

Methods of Collection Interface in Java

2. **addAll():** This method adds a collection of elements to the collection. It returns true if the elements are added otherwise returns false. The general syntax for this method is as follows:

```
addAll(Collection c) : boolean
```

Methods of Collection Interface in Java

3. **clear():** This method clears or removes all the elements from the collection. The general form of this method is as follows:

```
clear() : void
```

This method returns nothing.

Methods of Collection Interface in Java

4. **contains():** It checks that element is present or not in a collection. That is it is used to search an element. The general for contains() method is as follows:

```
contains(Object element) : boolean
```

This method returns true if the element is present otherwise false.

Methods of Collection Interface in Java

5. **containsAll():** This method checks that specified a collection of elements are present or not. It returns true if the calling collection contains all specified elements otherwise return false. The general syntax is as follows:

```
containsAll(Collection c) : boolean
```

Methods of Collection Interface in Java

6. **equals():** It checks for equality with another object. The general form is as follows:

```
equals(Object element) : boolean
```

Methods of Collection Interface in Java

7. **hashCode():** It returns the hash code number for the collection. Its return type is an integer. The general form for this method is:

```
hashCode() : int
```

Methods of Collection Interface in Java

8. **isEmpty():** It returns true if a collection is empty. That is, this method returns true if the collection contains no elements.

```
isEmpty() : boolean
```

Methods of Collection Interface in Java

9. **iterator():** It returns an iterator. The general form is given below:

```
iterator() : Iterator
```

Methods of Collection Interface in Java

10. **remove():** It removes a specified element from the collection. The general syntax is given below:

```
remove(Object element) : boolean
```

Methods of Collection Interface in Java

11. **removeAll():** The removeAll() method removes all elements from the collection. It returns true if all elements are removed otherwise returns false.

```
removeAll(Collection c) : boolean
```

Methods of Collection Interface in Java

12. **retainAll():** This method is used to remove all elements from the collection except the specified collection. It returns true if all the elements are removed otherwise returns false.

```
retainAll(Collection c) : boolean
```

Methods of Collection Interface in Java

13. **size():** The size() method returns the total number of elements in the collection. Its return type is an integer. The general syntax is given below:

```
size() : int
```

Methods of Collection Interface in Java

14. **toArray():** It returns the elements of a collection in the form of an array. The array elements are copies of the collection elements.

```
toArray() : Object[]
```

Methods of Collection Interface in Java

15. **Object[ ] toArray():** Returns an array that contains all the elements stored in the invoking collection. The array elements are the copies of the collection elements.

```
toArray(Object array[]) : Object[]
```

Collections Class in Java

The collections classes implement the collection interfaces. They are defined in java.util package. Some of the classes provide full implementations that can be used as it is.

Others are abstract that provide basic implementations that can be used to create concrete collections. A brief overview of each concrete collection class is given below.

1. AbstractCollection 2. AbstractList 3. AbstractQueue 4. AbstractSequentialList 5. AbstractSet 6. ArrayList	7. EnumSet 8. HashSet 9. LinkedHashSet 10. LinkedList 11. PriorityQueue 12. TreeSet

Collections Class in Java

1. **AbstractCollection:** It implements most of the collection interface. It is a superclass for all of the concrete collection classes.

2. **AbstractList:** It extends AbstractCollection and implements most of the List interface.

3. **AbstractQueue:** It extends AbstractCollection and implements the queue interface.

4. **AbstractSequentialList:** It extends AbstractList and uses sequential order to access elements.

5. **AbstractSet:** Extends AbstractCollection and implements most of the set interface.

Collections Class in Java

6. **ArrayList:** It implements a dynamic array by extending AbstractList.

7. **EnumSet:** Extends AbstractSet for use with enum elements.

8. **HashSet:** Extends AbstractSet for use with a hash table.

9. **LinkedHashSet:** Extends HashSet to allow insertion-order iterations.

10. **LinkedList:** Implements a linked list by extending AbstractSequentialList.

11. **PriorityQueue:** Extends AbstractQueue to support a priority-based queue.

12. **TreeSet:** Extends AbstractSet and implements the SortedSet interface.

END