Collections

Collections are used to group objects into single entity. Once the objects are stored in a collection you can then perform operations on single object or multiple objects (bulk operations).

You can insert, update, delete, and retrieve objects into entity. Also we can query the entity for size and other things.

Let us first understand what Java Collections are.

Java collections are made up of following:

**1. Interfaces**

Interfaces are abstract data types which represents the collections. Interfaces allow the collection classes to be manipulated independently given that the implementing class adheres to the contract of the interface.

**2. Implementations**

Implementations are class that implements the collection interfaces. There are abstract as well as concrete implementations of collection interfaces. These implementations are ready to use, highly efficient, well tested and reusable data structures.

**3. Algorithms**

On the core of the implementation of the interfaces there reside the powerful algorithm implementations done on object. This includes searching sorting, shuffling, frequency and others. These algorithms are finely tuned and suit the needs of many applications as per the data size.

**Benefits of Java Collections:**

**1. Tuned algorithms and quality**

There are several algorithms implemented in collections framework. We don’t know them but we use them daily. For example Arrays.sort(int[]).

These algorithms are finely tuned and suit the better needs of applications.

Because of these implementations developers are freed to design their own data structures and test them for errors or memory leaks.

**2. Reduced Programming effort**

As the algorithms and data structures are implemented in collections developer can focus on other important parts of application.

**3. Reduced learning effort**

It is very easy to learn collections because of its API names and documentation.

**4. Reusability**

Collections can be reused. Its interfaces and implementations are flexible enough to be reused given that it adheres contract of interface.

**Hierarchy of Collection Framework**

Let us see the hierarchy of Collection framework. The **java.util** package contains all the [classes](https://www.javatpoint.com/object-and-class-in-java) and [interfaces](https://www.javatpoint.com/interface-in-java) for the Collection framework.

Top of Form

## **Iterable Interface**

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

Iterator<T> iterator()

It returns the iterator over the elements of type T.

## **Collection Interface**

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

## **List Interface**

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

##### **ArrayList**

ArrayList is the implementation class of List Interface which is used to store a group of individual objects where duplicate values are allowed. ArrayList internally follows array structure, which means in ArrayList all the elements are stored in contiguous memory locations same as an array, but ArrayList size is not fixed.

ArrayList is not a synchronized class. If any object is synchronized we can access only one thread at a time but if an object is not. synchronized then we can access multiple threads.

**package** Collections;

**import** java.util.\*;

**public** **class** ArrayList1 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Integer> li=**new** ArrayList<>();

li.add(10);

li.add(20);

li.add(30);

li.add(40);

li.add(50);

**for**(**int** i=0;i<li.size();i++)

{

System.***out***.println("List elements are:"+li.get(i));

}

li.add(2, 1000);

li.set(0, 500);

li.remove(0);

**for**(Integer i:li)

{

System.***out***.println("List elements usig foreach are:"+i);

}

li.remove(Integer.*valueOf*(30));

Iterator<Integer> it=li.iterator();

**while**(it.hasNext())

{

System.***out***.println("List elements usig Iterator are:"+it.next());

}

List<String> list=**new** ArrayList<>();

list.add("Anu");

list.add("Priya");

list.add("Tarun");

list.add("Kiran");

list.add("Hari");

list.remove("Tarun");

System.***out***.println(list);

System.***out***.println(list.contains("Sindhu"));

}

}

## **LinkedList**

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

## **Stack**

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

**package** Collections;

**import** java.util.\*;

**public** **class** Stack1 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Stack<String> fruits=**new** Stack<>();

fruits.push("Apple");

fruits.push("Banana");

fruits.push("Mango");

fruits.push("Kiwi");

fruits.push("Grapes");

fruits.push("Pomogranate");

System.***out***.println(fruits);

System.***out***.println(fruits.peek());

System.***out***.println(fruits.pop());

System.***out***.println(fruits.peek());

}

}