# L56

Monday, February 7, 2022 2:10 PM

# Data Structures - Fundamentals, Collection Hierarchy, Collection Root Interface, Iterable and Iterator

## Fundamentals of Data Structures

Q1. In programming, a **data structure** refers to a structured way of organizing data which facilitates efficient storage and retrieval.

Even though a user defined class such as **Person** allows for storing data in a structured manner, when we use the term **data structures**, we usually refer to containers such as arrays, lists, stacks, queues etc.

Each data structure is designed to be efficient for a specific kind of problem. For example, an array (the most primitive data structure) is designed to allow fast (constant time) **access to elements by index**.

Similarly, if we want to store only unique elements, we would choose set instead of an array, since array allows for duplicate elements.

An array as we have learnt earlier, is of fixed length. Once an array of a particular size is declared, we cannot alter the size to add more elements dynamically.

Every programming language provides some built in support for most commonly used data structures.

The **data structures** that are provided in Java language, are grouped in a package called java.util, they are commonly referred as the collection framework or simply collections.

These classes which are called collections have a common root interface by the name Collection.

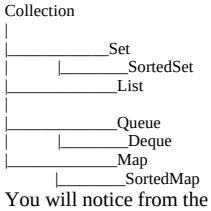
Select all the correct statements given below.

- [ ] Any collection or a data structure can contain only one element.
- [ ] Any collection or a data structure can have multiple elements but all the elements should be of the same data type.
- [ ]
```
int x = 3;
int[] myArr = {3, 4, 5};
```
In the above code snippet, `x` can be called a collection.
- [x] The implementation classes of common data structures like `list`, `set` or `queue` which are present in `java.util` package are referred as `collection` classes.

## Collection Hierarchy

Q1. In Java, Collection is the root interface for the collection hierarchy.

The main interfaces in the **collection framework** are given below:




You will notice from the above hierarchy that even though Map is not a subtype of Collection interface, Map is part of the collection framework.

The Java collection framework consists of collection hierarchy, Map interface with its hierarchy along with many other utility classes present in the java.util package like Date, StringTokenizer etc.

The three main sub-interfaces of Collection interface are Set, List and Queue. We will learn in detail about each interface in the ensuing sections.

Select all the correct statements.

- [ ] **SortedSet**, **Deque** and **SortedMap** are `concrete classes` while the remaining **Collection**, **Map**, **Set**, **List** and **Queue** are `interfaces`.
- [ ] `Collection` interface is a supertype for interface `SortedMap`
- [ ] `SortedSet` is a subtype of `Queue`.
- [x] `List` is a subtype of `Collection`.

## Collection Root Interface

Q1. Collection is the root interface of the collection hierarchy, meaning all other collection interfaces like Set, List and Queue extend Collection.

A collection is a group of objects. These objects are referred as elements of the collection.

The most commonly used methods in the Collection interface to manipulate and query the underlying collection are : add(E e) , remove(Object obj), contains(Object obj), size().

Note: In most of the interfaces and classes in the collection framework, you will find notations like E, K, V used for parameter types or return types. They are related to generic syntax. We will learn more about them in the sections related to generics. For now you can safely assume Object as the type, whenever you see these characters.

Select all the correct statements given below.
[**Hint**: Click on the methods to learn what they do.]

- [x] The size() method in the collection interface returns an `int` value which contains the count of elements present in that collection.
- [x] The clear() method in the collection interface removes all the elements present in that collection.
- [x] The isEmpty() method in the collection interface returns `true`, if the collection is empty and returns `false` otherwise.
- [ ] `Set` interface implements `Collection` interface.

# Set, List, Queue, Map

Q1. The interfaces Set, List and Queue extend Collection. Some of these also add additional methods apart from the methods present in Collection interface.

A Set does not allow duplicate elements. Meaning, when the add method is called by passing an element which is already present, the underlying Set implementation does not add the element again and returns false (false indicates that the collection was not modified by the add method call).

A List is essentially a flexible array. It maintains the order of elements and also allows index-based access of elements like an array (which a Set does not). It allows duplicate elements.

A Queue interface provides methods for a queue data structure implementing first-in-first-out (FIFO). There are special implementations of Queue interface like PriorityQueue, which may alter this behaviour.

A Map is a collection of ***key-value*** *pairs* and not just elements.

Select all the correct statements given below.
[**Note**: Make sure to click on the methods to learn what they do.]

- [ ] A `Queue` and `List` will not accept duplicate elements.
- [x] A `Set` does not accept duplicate elements.
- [x] A `List` allows index-based access of elements.
- [ ] A `Map` extends `List` interface.

# Iterable and Iterator

Q1. A class which implements Iterable interface can be used as a target in the **for-each loop** statement, which is also called the **enhanced for statement**.

Programmers do not normally implement this interface.

However, we need to know that the Collection interface which is the root interface of all collections, extends this Iterable interface. Which means that instances of all subclasses of Collection interface can be used in a **for-each** statement.

**Note**: Whenever we use collection classes, we need to import the relevant classes or import all classes in the util package using a statement like: import java.util.*;

For example, if we have a collection of type List called aList with names of persons, the below code demonstrates the syntax for iterating over the elements contained in aList and printing them.

```
for (Object name : aList) {
        System.out.println(name);
}
```

**Note**: While using a **for-each** loop to iterate over instances of Collection interface, the element type will be Object. After learning how to use Generics in Java, we will be able to declare instances of Collection interface of a specify data-type and use that type information in the **for-each** loop.

The below program assumes some strings are passed to its main(...) method during execution. It internally creates an instance of ArrayList (a class which implements List interface) and adds the elements it receives as arguments in the main(...) method. Read the instructions given in the comments and fill in the missing code.

```java
package q11364;
import java.util.*;
public class ForEachDemo
{
  public static void main (String[]args)
  {
    List namesList = new ArrayList ();
    for (String name:args)
      {
    namesList.add (name);
      }

    /*
       Write a for-each loop below this comment
       which iterates over the nameslist and
       prints each element on a seperate line using println(...)
     */

    for (Object name:namesList)
      {

    System.out.println (name);

      }

  }
}

```

Q2. The Iterable interface has a method called iterator(), which returns an object of type Iterator.

The Iterator interface has two methods hasNext() and next() which can also be used for iterating over all the elements present in a collection. See and retype the below code which has an example for iterating over ArrayList using an Iterator.

Note: Whenever we use collection classes, we need to import the relevant classes or import all classes in the util package us ing a statement like: import java.util.*;

```java
package q11365;
import java.util.*;
public class IteratorDemo {
    public static void main(String[] args) {
        List planetsList = new ArrayList();
        planetsList.add("Mercury");
        planetsList.add("Venus");
        planetsList.add("Earth");
        planetsList.add("Mars");
        planetsList.add("Jupiter");
        planetsList.add("Saturn");
        planetsList.add("Uranus");
        planetsList.add("Neptune");
        planetsList.add("Pluto");
        Iterator itr = planetsList.iterator();
        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}

```