**Advanced Programming (Java)**

Week 4 – Reading, Weekly Coding Task and Practice Material

# Tasks to be done in this week

1. Practice Array Programs
2. Implement ArrayList and try its methods
3. Implement Hashtable for dictionary and demonstrate adding, retrieving, and removing any word.

# Arrays

**Try to Practice as many as you can do**

1. Java Method to Calculate Sum & Average of all elements of an integer Array size 10.
2. [Java Method to Increment Every Element of the Array by One & Print Incremented Array](http://www.sanfoundry.com/c-program-increment-element-array/)
3. Java Method to Input an Array, Store the Squares and cubes of these elements in new Arrays & Print those.
4. Java Method that finds largest element present in an integer array. It must also print the location or index at which maximum element occurs in array.
5. Java Method to Read an Array and Search for an Element
6. Java Method to Print the Number of Odd & Even Numbers in an Array
7. Java Method to Sort the Array in an Ascending Order and Descending Order
8. Java Method to Find the Second Largest & Smallest Elements in an Array
9. Java Method to Print All the Repeated Numbers with Frequency in an Array
10. Java Method to Merge the Elements of 2 Sorted Array
11. Java Method to Insert an Element in a Specified Position in a given Array
12. Java Method to Delete the Specified Integer from an Array

# ArrayList

**Objectives:**

In this lab we shall look at how the Java Collections framework handles collections,  Specifically we concentrate on the ArrayList class. In general, an **ArrayList** serves the same purpose as an array, except that an **ArrayList** can change length while the program is running. An iterator is an object that is used with a collection to provide sequential access to the collection elements. This access allows examination and possible modification of the elements. An iterator imposes an ordering on the elements of a collection even if the collection itself does not impose any order on the elements it contains. If the collection does impose an ordering on its elements, then the iterator will use the same ordering

## ArrayList in java with example programs – Collections Framework

Arraylist is a class which implements List interface. It is widely used because of the functionality and flexibility it offers. Most of the developers **choose Arraylist over Array** as it’s a very good alternative of traditional java arrays.

The issue with arrays is that they are of fixed length so if it is full we cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn’t shrink. On the other ArrayList can dynamically grow and shrink as per the need. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy. Let’s see the ArrayList example first then we will discuss it’s methods and their usage.

### ArrayList Example in Java

import java.util.\*;

public class ArrayListExample {

public static void main(String args[]) {

/\*Creation of ArrayList: I'm going to add String

\*elements so I made it of string type \*/

ArrayList<String> obj = new ArrayList<String>();

/\*This is how elements should be added to the array list\*/

obj.add("Ajeet");

obj.add("Harry");

obj.add("Chaitanya");

obj.add("Steve");

obj.add("Anuj");

/\* Displaying array list elements \*/

System.out.println("Currently the array list has following elements:"+obj);

/\*Add element at the given index\*/

obj.add(0, "Rahul");

obj.add(1, "Justin");

/\*Remove elements from array list like this\*/

obj.remove("Chaitanya");

obj.remove("Harry");

System.out.println("Current array list is:"+obj);

/\*Remove element from the given index\*/

obj.remove(1);

System.out.println("Current array list is:"+obj);

}

}

Output:

Currently the array list has following elements:[Ajeet, Harry, Chaitanya, Steve, Anuj]

Current array list is:[Rahul, Justin, Ajeet, Steve, Anuj]

Current array list is:[Rahul, Ajeet, Steve, Anuj]

## Methods of ArrayList class

In the above example we have used methods such as add and remove. However there are number of methods available which can be used directly using object of ArrayList class. Let’s try few of the important methods.

1) **add( Object o)**: This method adds an object o to the arraylist.

obj.add("hello");

This statement would add a string hello in the arraylist at last position.

2) **add(int index, Object o)**: It adds the object o to the array list at the given index.

obj.add(2, "bye");

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) **remove(Object o)**: Removes the object o from the ArrayList.

obj.remove("Chaitanya");

This statement will remove the string “Chaitanya” from the ArrayList.

4) **remove(int index)**: Removes element from a given index.

obj.remove(3);

It would remove the element of index 3 (4th element of the list – List starts with o).

5) **set(int index, Object o)**: Used for updating an element. It replaces the element present at the specified index with the object o.

obj.set(2, "Tom");

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6)**int indexOf(Object o)**: Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

int pos = obj.indexOf("Tom");

This would give the index (position) of the string Tom in the list.

7) **Object get(int index)**: It returns the object of list which is present at the specified index.

String str= obj.get(2);

Function get would return the string stored at 3rd position (index 2) and would be assigned to the string “str”. We have stored the returned value in string variable because in our example we have defined the ArrayList is of String type. If you are having integer array list then the returned value should be stored in an integer variable.

8) **int size()**: It gives the size of the ArrayList – Number of elements of the list.

int numberofitems = obj.size();

9) **boolean contains(Object o)**: It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

obj.contains("Steve");

It would return true if the string “Steve” is present in the list else we would get false.

10) **clear():** It is used for removing all the elements of the array list in one go. The below method will remove all the elements of ArrayList whose object is obj.

obj.clear();

# Hashtable in Java

[*Hashtable*](https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html) is the oldest implementation of a hash table data structure in Java. The *HashMap* is the second implementation, which was introduced in JDK 1.2. A hash table supports fast insertion, fast retrieval, fast removal of data. It Provides virtually direct access to objects based on a key (a unique String or Integer). The **Hashtable** class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

## **Features of Hashtable**

* It is similar to HashMap, but is synchronized.
* Hashtable stores key/value pair in hash table.
* In Hashtable we specify an object that is used as a key, and the value we want to associate to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.
* The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

## Load Factor

## **The load factor** is a measure of how full the hash table is allowed to get before its capacity is automatically increased.

The **load factor** is the average number of key-value pairs per bucket.

load factor = (total number of key-value pairs)/number of buckets

## When to use *Hashtable*

Let’s say we have a dictionary, where each word has its definition. Also, we need to get, insert and remove words from the dictionary quickly.

Hence, *Hashtable* (or *HashMap*) makes sense. Words will be the keys in the *Hashtable*, as they are supposed to be unique. Definitions, on the other hand, will be the values.

**Example of Use (to be done as a Daily Task)**

Let’s continue with the dictionary example. We’ll model *Word* as a key:

public class Word {  
 private String name;  
  
 public Word(String name) {  
 this.name = name;  
 }  
   
 // ...  
}

Let’s say the values are *Strings*. Now we can create a *Hashtable*:

**Hashtable<Word, String> table = new Hashtable<>();**

First, let’s add an entry:

**Word word = new Word("cat");  
table.put(word, "an animal");**

Also, to get an entry:

**String definition = table.get(word);**

Finally, let’s remove an entry:

**definition = table.remove(word);**

**Declaration:**

public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable

**Type Parameters:**

* **K** – the type of keys maintained by this map
* **V** – the type of mapped values

**Constructors:**

In order to create a Hashtable, we need to import it from **java.util.Hashtable**. There are various ways in which we can create a Hashtable.

**1. Hashtable():** This creates an empty hashtable with the default load factor of 0.75 and an initial capacity is 11.

*Hashtable<K, V> ht = new Hashtable<K, V>();*

|  |
| --- |
| **// Java program to demonstrate adding elements to Hashtable**    import java.io.\*;  import java.util.\*;    class AddElementsToHashtable {      public static void main(String args[])      {          // No need to mention the Generic type twice          Hashtable<Integer, String> ht1 = new Hashtable<>();            // Initialization of a Hashtable using Generics          Hashtable<Integer, String> ht2 = new Hashtable<Integer, String>();            // Inserting the Elements using put() method          ht1.put(1, "one");          ht1.put(2, "two");          ht1.put(3, "three");            ht2.put(4, "four");          ht2.put(5, "five");          ht2.put(6, "six");            // Print mappings to the console          System.out.println("Mappings of ht1 : " + ht1);          System.out.println("Mappings of ht2 : " + ht2);      }  } |

**Output**

Mappings of ht1 : {3=three, 2=two, 1=one}

Mappings of ht2 : {6=six, 5=five, 4=four}

**2. Hashtable(int initialCapacity):** This creates a hash table that has an initial size specified by initialCapacity and the default load factor is 0.75.

*Hashtable<K, V> ht = new Hashtable<K, V>(int initialCapacity);*

**3. Hashtable(int size, float fillRatio):** This version creates a hash table that has an initial size specified by size and fill ratio specified by fillRatio. fill ratio: Basically, it determines how full a hash table can be before it is resized upward and its Value lies between 0.0 to 1.0.

*Hashtable<K, V> ht = new Hashtable<K, V>(int size, float fillRatio);*

**4. Hashtable(Map<? extends K,? extends V> m):** This creates a hash table that is initialized with the elements in m.

### **Performing Various Operations on Hashtable**

**1. Adding Elements:** In order to add an element to the hashtable, we can use the [put()](https://www.geeksforgeeks.org/hashtable-put-method-in-java/#:~:text=Hashtable.,replaced%20by%20the%20new%20value.) method. However, the insertion order is not retained in the hashtable. Internally, for every element, a separate hash is generated and the elements are indexed based on this hash to make it more efficient.

|  |
| --- |
| // Java program to demonstrate adding elements to Hashtable    import java.io.\*;  import java.util.\*;    class AddElementsToHashtable {      public static void main(String args[])      {          // No need to mention the Generic type twice          Hashtable<Integer, String> ht1 = new Hashtable<>();            // Initialization of a Hashtable using Generics          Hashtable<Integer, String> ht2  = new Hashtable<Integer, String>();            // Inserting the Elements using put() method          ht1.put(1, "Geeks");          ht1.put(2, "For");          ht1.put(3, "Geeks");            ht2.put(1, "Geeks");          ht2.put(2, "For");          ht2.put(3, "Geeks");            // Print mappings to the console          System.out.println("Mappings of ht1 : " + ht1);          System.out.println("Mappings of ht2 : " + ht2);      }  } |

**Output**

Mappings of ht1 : {3=Geeks, 2=For, 1=Geeks}

Mappings of ht2 : {3=Geeks, 2=For, 1=Geeks}

|  |
| --- |
| **2. Changing Elements:** After adding the elements if we wish to change the element, it can be done by again adding the element with the [put()](https://www.geeksforgeeks.org/hashtable-put-method-in-java/#:~:text=Hashtable.,replaced%20by%20the%20new%20value.) method. Since the elements in the hashtable are indexed using the keys, the value of the key can be changed by simply inserting the updated value for the key for which we wish to change.  // Java program to demonstrate updating Hashtable    import java.io.\*;  import java.util.\*;  class UpdatesOnHashtable {      public static void main(String args[])      {          // Initialization of a Hashtable          Hashtable<Integer, String> ht = new Hashtable<Integer, String>();            // Inserting the Elements using put method          ht.put(1, "Geeks");          ht.put(2, "Geeks");          ht.put(3, "Geeks");            // print initial map to the console          System.out.println("Initial Map " + ht);            // Update the value at key 2          ht.put(2, "For");            // print the updated map          System.out.println("Updated Map " + ht);      }  } |

**Output**

Initial Map {3=Geeks, 2=Geeks, 1=Geeks}

Updated Map {3=Geeks, 2=For, 1=Geeks}

**3. Removing Element:** In order to remove an element from the Map, we can use the [remove()](https://www.geeksforgeeks.org/hashtable-remove-method-in-java/#:~:text=util.,particular%20key%20in%20the%20Table.) method. This method takes the key value and removes the mapping for a key from this map if it is present in the map.

|  |
| --- |
| // Java program to demonstrate the removing mappings from Hashtable   import java.io.\*;  import java.util.\*;  class RemovingMappingsFromHashtable {        public static void main(String args[])      {          // Initialization of a Hashtable          Map<Integer, String> ht = new Hashtable<Integer, String>();            // Inserting the Elements using put method          ht.put(1, "Geeks");          ht.put(2, "For");          ht.put(3, "Geeks");          ht.put(4, "For");            // Initial HashMap          System.out.println("Initial map : " + ht);            // Remove the map entry with key 4          ht.remove(4);            // Final Hashtable          System.out.println("Updated map : " + ht);      }  } |

**Output**

Initial map : {4=For, 3=Geeks, 2=For, 1=Geeks}

Updated map : {3=Geeks, 2=For, 1=Geeks}

**References:**

1. <https://www.javatpoint.com/array-in-java>
2. <https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/util/Arrays.html>
3. <http://beginnersbook.com/2013/12/java-arraylist/>
4. <https://www.javatpoint.com/java-hashtable>
5. <https://www.youtube.com/watch?v=KyUTuwz_b7Q>
6. <https://programming.guide/hash-table-load-factor-and-capacity.html>
7. <https://www.geeksforgeeks.org/hashtable-in-java/>