Array vs ArrayList in Java

In Java, following are two different ways to create an array.

Array:

Simple fixed sized arrays that we create in Java, like below

```
int arr[] = new int[10]
```

ArrayList :

Dynamic sized arrays in Java that implement List interface.

```
ArrayList<Type> arrL = new ArrayList<Type>();
Here Type is the type of elements in ArrayList to
be created
```

Differences between Array and ArrayList

1. An array is basic functionality provided by Java. ArrayList is part of collection framework in Java. Therefore array members are accessed using [], while ArrayList has a set of methods to access elements and modify them.

```
// A Java program to demonstrate differences between array and ArrayList
import java.util.ArrayList;
import java.util.Arrays;

class Test
{
   public static void main(String args[])
   {
      /* ..... Normal Array..... */
      int[] arr = new int[2];
      arr[0] = 1;
      arr[1] = 2;
      System.out.println(arr[0]);

      /*.....ArrayList.....*/
      // Create an arrayList with initial capacity 2
      ArrayList<Integer> arrL = new ArrayList<Integer>(2);

      // Add elements to ArrayList
      arrL.add(1);
      arrL.add(2);

      // Access elements of ArrayList
      System.out.println(arrL.get(0));
```

```
  }
}
```
Output:

```
1
1
```

2. Array is a fixed size data structure while ArrayList is not. One need not to mention the size of Arraylist while creating its object. Even if we specify some initial capacity, we can add more elements.

```
// A Java program to demonstrate differences between array and ArrayList
import java.util.ArrayList;
import java.util.Arrays;
class Test
{
   public static void main(String args[])
   {
      /* ..... Normal Array..... */
      // Need to specify the size for array
      int[] arr = new int[3];
      arr[0] = 1;
      arr[1] = 2;
      arr[2] = 3;
      // We cannot add more elements to array arr[]

      /*.....ArrayList.....*/
      // Need not to specify size
      ArrayList<Integer> arrL = new ArrayList<Integer>();
      arrL.add(1);
      arrL.add(2);
      arrL.add(3);
      arrL.add(4);
      // We can add more elements to arrL

      System.out.println(arrL);
      System.out.println(Arrays.toString(arr));
   }
}
```
Output:

```
[1, 2, 3, 4]
[1, 2, 3]
```

3. Array can contain both primitive data types as well as objects of a class depending on the definition of the array. However, ArrayList only supports object entries, not the primitive data types.

Note: When we do arraylist.add(1); : it converts the primitive int data type into an Integer object.

```
import java.util.ArrayList;
class Test
{
   public static void main(String args[])
   {
      // allowed
       int[] array = new int[3];

       // allowed, however, need to be initialized
       Test[] array1 = new Test[3];

       // not allowed (Uncommenting below line causes
       // compiler error)
       // ArrayList<char> arrL = new ArrayList<char>();

       // Allowed
       ArrayList<Integer> arrL1 = new ArrayList<>();
       ArrayList<String> arrL2 = new ArrayList<>();
       ArrayList<Object> arrL3 = new ArrayList<>();
   }
}
```

Since ArrayList can't be created for primitive data types, members of ArrayList are always references to objects at different memory locations. Therefore in ArrayList, the actual objects are never stored at contiguous locations. References of the actual objects are stored at contiguous locations.

In array, it depends whether the arrays is of primitive type or object type. In case of primitive types, actual values are contiguous locations, but in case of objects, allocation is similar to ArrayList.