ArrayList

Ratna Mufidah, S.Kom., M.Kom.

ArrayList

The ArrayList is **a resizable array**

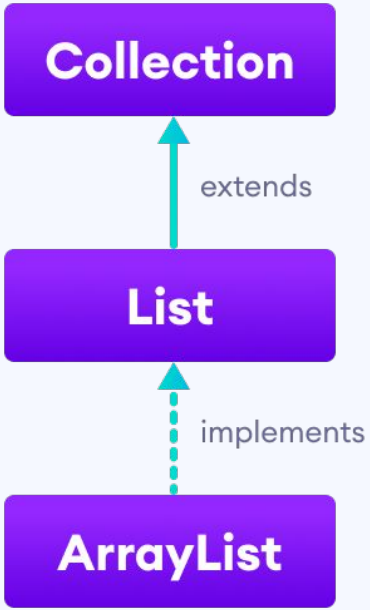
It is a class of **java.util** package

It is like an array, **but** there is **no size limit**

It is much **more flexible** than the traditional array

ArrayList in Java

- **ArrayList** implemented using the **List interface**
- **ArrayList** is a part of the **Collection** framework, it has many features not available with arrays



https://www.programiz.com/java-programming/arraylist

ArrayList in Java

- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because the array works on an index basis.
- In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

ArrayList in Java

- We **can not create an array list of the primitive types**, such as int, float, char, etc. It is required to use the required wrapper class in such cases.

```
ArrayList<int> al = ArrayList<int>(); // does not work
ArrayList<Integer> al = new ArrayList<Integer>(); // works fine
```

- Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

ArrayList in Java

- ArrayList Duplicates Are Allowed.
- Heterogeneous objects are allowed.
- Null insertion is possible.

Constructors in ArrayList

ArrayList()

```
ArrayList arr = new ArrayList();
```

ArrayList(Collection c)

```
ArrayList arr = new ArrayList(c);
```

ArrayList(int Capacity)

```
ArrayList arr = new ArrayList(N);
```

Operations performed in ArrayList

- Adding element to List/ **Add** element
- Changing elements/ **Set** element
- Removing elements/**Delete** element
- **get** elements
- **Iterating** elements
- **add** elements in between two number
- **Sorting** elements
- ArrayList **size**

Operations performed in ArrayList

Adding Elements→ add() method

1. **add(Object)**: This method is used to add an element at the end of the ArrayList.
2. **add(int index, Object)**: This method is used to add an element at a specific index in the ArrayList.

Operations performed in ArrayList

Adding Elements

```java
import java.util.ArrayList;

public class AddTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // list 1 with default capacity 10
        ArrayList arr = new ArrayList();
        arr.add("Emily");
        arr.add("Bob");
        arr.add(20);
        arr.add("Cindy");
        arr.add(null);
        System.out.println(arr);

        // list 2.
        ArrayList arr1 = new ArrayList();
        arr1.add("Emily");
        arr1.add("Bob");
        arr1.add("Cindy");
        System.out.println(arr1);

    /*Call addAll(Collection c) method
      using reference variable arr
      to add all elements at the end of the list1.*/
        arr.addAll(arr1);
        System.out.println(arr);

    /* Call addAll(int index, Collection c)
       method using reference arr1
       to add all elements at specified position 2.*/
        arr1.addAll(2, arr);
        System.out.println(arr1);
    }

}
```

- duplicate elements
- heterogeneous elements (i.e. String and Integer)
- a null element.

```
System.out.println(al); -->
System.out.println(al.toString());
```

Operations performed in ArrayList

Changing Elements→ set() method

The **set() method** replaces element at a particular position in the list with the specified element.

set(int index, Object o)

Operations performed in ArrayList

Changing Elements

```java
import java.util.ArrayList;

public class SetTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // list 1 with default capacity 10
        ArrayList arr = new ArrayList();
        arr.add("Emily");
        arr.add("Bob");
        arr.add(20);
        arr.add("Cindy");
        arr.add(null);
        System.out.println(arr);

        /*Call set() method to replace an element null
        in listwith "Deasy" element at position 4.*/
        arr.set(4, "Deasy");
        System.out.println(arr);

        /*Call set() method to replace an element 20
        in listwith "Anne" element at position 2.*/
        arr.set(2, "Anne");
        System.out.println(arr);

    }

}
```

Operations performed in ArrayList

Removing Elements→ remove() method

1. **remove(Object)**: This method is used to simply remove an object from the ArrayList. If there are multiple such objects, then the first occurrence of the object is removed.
2. **remove(int index)**: Since an ArrayList is indexed, this method takes an integer value which simply removes the element present at that specific index in the ArrayList. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

Operations performed in ArrayList

Removing Elements

```java
import java.util.ArrayList;

public class SetTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // list 1 with default capacity 10
        ArrayList arr = new ArrayList();
        arr.add("Emily");
        arr.add("Bob");
        arr.add(20);
        arr.add("Cindy");
        arr.add(null);
        System.out.println(arr);

        /*Call set() method to replace an element null
        in listwith "Deasy" element at position 4.*/
        arr.set(4, "Deasy");
        System.out.println(arr);

        /*Call set() method to replace an element 20
        in listwith "Anne" element at position 2.*/
        arr.set(2, "Anne");
        System.out.println(arr);

    }

}
```

Operations performed in ArrayList

Removing Elements→ remove() method

1. **remove(Object)**: This method is used to simply remove an object from the ArrayList. If there are multiple such objects, then the first occurrence of the object is removed.
2. **remove(int index)**: Since an ArrayList is indexed, this method takes an integer value which simply removes the element present at that specific index in the ArrayList. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

Operations performed in ArrayList

Removing Elements

```java
import java.util.ArrayList;

public class RemoveTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Default capacity is 10.
        ArrayList<String> arr = new ArrayList<String>();
        arr.add("Andrew");
        arr.add("Bob");
        arr.add("Cella");
        arr.add("Denlie");
        arr.add(null);
        arr.add("Emely");
        System.out.println(arr);

        arr.remove("Denlie");
        System.out.println(arr);

        arr.remove(3);
        System.out.println(arr);

    }
```

Operations performed in ArrayList

Get Elements→ get () method

It returns the element at the specified position in this ArrayList.

Operations performed in ArrayList

Get Elements

```java
import java.util.ArrayList;

public class GetTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ArrayList<String> arr = new ArrayList<String>();
        arr.add("Andrew");
        arr.add("Bob");
        arr.add("Cella");
        arr.add("Denlie");
        arr.add(null);
        arr.add("Emely");
        System.out.println(arr);

        // get method
        String s= arr.get(1);
        System.out.println("at indext 1 number is:"+s);
    }

}
```

Operations performed in ArrayList

Iterating Elements

The most famous ways are by using the **basic for loop** in combination with a **get() method** to get the element at a specific index.

Operations performed in ArrayList

Iterating Elements

```java
import java.util.ArrayList;

public class IterationTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ArrayList<String> arr = new ArrayList<String>();
        arr.add("Object");
        arr.add("Programming");
        arr.add(1,"Oriented");

        // Using the Get method and the for loop
        for (int i = 0; i < arr.size(); i++) {
            System.out.print(arr.get(i) + " ");
        }

        System.out.println();

        // Using the for each loop
        for (String str : arr)
            System.out.print(str + " ");
    }
}
```

Operations performed in ArrayList

Add Elements Between Two Numbers

```java
import java.util.ArrayList;

public class BetweenTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ArrayList<Integer> arr = new ArrayList();
        arr.add(1);
        arr.add(2);
        arr.add(4);
        System.out.println(arr);
        // insert missing element 3
        arr.add(2, 3);
        System.out.println(arr);
    }

}
```

Operations performed in ArrayList

ArrayList Sort

```java
import java.util.ArrayList;
import java.util.Collections;

public class SortTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ArrayList<Integer> arr = new ArrayList();
        arr.add(4);
        arr.add(2);
        arr.add(3);
        arr.add(1);
        System.out.println("Before sorting list:");
        System.out.println(arr);
        Collections.sort(arr);
        System.out.println("after sorting list:");
        System.out.println(arr);

    }
}
```

Operations performed in ArrayList

Size of Elements

```java
import java.util.ArrayList;

public class SizeTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ArrayList<Integer> arr = new ArrayList();
        arr.add(4);
        arr.add(2);
        arr.add(3);
        arr.add(1);
        System.out.println(arr);
        int b = arr.size();
        System.out.println("The size is :" + b);
    }

}
```

Advantages of ArrayList in Java?

1. **Dynamic size**: Unlike arrays, an ArrayList can grow or shrink dynamically, meaning that we do not need to specify its size at the time of declaration.

2. **Easy to use**: ArrayList is easy to use and provides a variety of methods for adding, removing, and accessing elements.

3. **Type safety**: ArrayList provides type-safety in Java, meaning that we can only add elements of the same data type.

When to use ArrayList in Java?

- We want to store duplicate elements.
- We want to store null elements.
- It is more preferred in Java when getting of the element is more as compared to adding and removing elements.
- We are not working in the multi-threading environment in Java because ArrayList is non-synchronized.

Thanks !

Ada pertanyaan?

References

https://www.geeksforgeeks.org/
https://www.w3schools.com/
https://www.javatpoint.com/
https://www.codejava.net/
https://www.scientecheasy.com/

slidesgo