ArrayList

The `ArrayList` class in Java is a part of the Java Collections Framework and is one of the most commonly used classes for dynamically resizing arrays. It provides powerful methods for managing collections of data and is preferred when the size of the array can vary.

---

1. Introduction to ArrayList

- **ArrayList** is a resizable array implementation of the `List` interface.
- It allows duplicate elements and maintains the insertion order.
- ArrayList is part of `java.util` package and can grow and shrink dynamically.

Basic Characteristics:

- **Dynamic Resizing**: Unlike arrays, `ArrayList` can change its size dynamically.
- **Indexed Access**: Elements can be accessed by their index, just like arrays.
- **Non-Synchronized**: `ArrayList` is not synchronized, meaning it is not thread-safe.

---

2. Creating an ArrayList

To use an `ArrayList`, you first need to import it from the `java.util` package:

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

Example: Creating and Initializing an ArrayList

```
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String[] args) {
        // Create an ArrayList of String type
        ArrayList<String> fruits = new ArrayList<>();
```

```
        // Adding elements to the ArrayList
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");

        // Display the ArrayList
        System.out.println("Fruits: " + fruits);
    }
}
```

Output:

```
Fruits: [Apple, Banana, Orange]
```

---

3. Common Methods of ArrayList

`ArrayList` provides several useful methods to manipulate the list. Here are some of the most commonly used ones:

1. **add(E e)**: Adds an element to the end of the list.
2. **add(int index, E element)**: Inserts an element at the specified index.
3. **get(int index)**: Returns the element at the specified index.
4. **set(int index, E element)**: Replaces the element at the specified index.
5. **remove(int index)**: Removes the element at the specified index.
6. **remove(Object o)**: Removes the first occurrence of the specified element.
7. **size()**: Returns the number of elements in the list.
8. **contains(Object o)**: Checks if the list contains the specified element.
9. **isEmpty()**: Checks if the list is empty.
10. **clear()**: Removes all elements from the list.

Example: Using ArrayList Methods

```
import java.util.ArrayList;

public class ArrayListMethodsExample {
    public static void main(String[] args) {
        ArrayList<String> fruits = new ArrayList<>();

        // Adding elements
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");

        // Accessing elements
        System.out.println("First fruit: " + fruits.get(0));
// Outputs: Apple

        // Modifying elements
        fruits.set(1, "Mango"); // Changes "Banana" to "Mang
o"

        // Removing elements
        fruits.remove("Orange"); // Removes "Orange"

        // Checking size
        System.out.println("Number of fruits: " + fruits.size
()); // Outputs: 2

        // Checking if the list contains a specific element
        if (fruits.contains("Apple")) {
            System.out.println("Apple is in the list.");
        }

        // Iterating through the list
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
```

```
        // Clearing the list
        fruits.clear();
        System.out.println("List after clearing: " + fruits);
// Outputs: []
    }
}
```

Output:

```
First fruit: Apple
Number of fruits: 2
Apple is in the list.
Apple
Mango
List after clearing: []
```

---

4. Iterating Over an ArrayList

You can iterate over an `ArrayList` using different methods:

Example: Using For-Each Loop

```
import java.util.ArrayList;

public class ArrayListIterationExample {
    public static void main(String[] args) {
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");

        // Iterating using for-each loop
        for (String fruit : fruits) {
            System.out.println(fruit);
```

```
        }
    }
}
```

Output:

```
Apple
Banana
Orange
```

Example: Using Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListIteratorExample {
    public static void main(String[] args) {
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");

        // Iterating using Iterator
        Iterator<String> iterator = fruits.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

Output:

```
Apple
Banana
```

```
Orange
```

5. ArrayList vs. Array

While `ArrayList` is similar to arrays, it has several advantages:

- **Dynamic Size**: Unlike arrays, the size of an `ArrayList` can grow and shrink dynamically.
- **Built-in Methods**: `ArrayList` provides built-in methods to perform various operations such as adding, removing, and searching for elements.
- **Type-Safe**: Using generics, you can create type-safe collections that prevent runtime errors.

Example: Array vs. ArrayList

```
public class ArrayVsArrayList {
    public static void main(String[] args) {
        // Array Example
        String[] array = new String[3];
        array[0] = "Apple";
        array[1] = "Banana";
        array[2] = "Orange";

        // ArrayList Example
        ArrayList<String> arrayList = new ArrayList<>();
        arrayList.add("Apple");
        arrayList.add("Banana");
        arrayList.add("Orange");

        // Adding another element to ArrayList (not possible
with a fixed-size array)
        arrayList.add("Mango");

        // Printing Array
        System.out.println("Array:");
```

```
        for (String fruit : array) {
            System.out.println(fruit);
        }

        // Printing ArrayList
        System.out.println("\\nArrayList:");
        for (String fruit : arrayList) {
            System.out.println(fruit);
        }
    }
}
```

Output:

```
Array:
Apple
Banana
Orange

ArrayList:
Apple
Banana
Orange
Mango
```

---

6. Converting ArrayList to Array and Vice Versa

You can easily convert an `ArrayList` to an array and vice versa.

Example: Converting ArrayList to Array

```
import java.util.ArrayList;

public class ArrayListToArray {
    public static void main(String[] args) {
```

```
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Orange");

        // Converting ArrayList to Array
        String[] fruitArray = new String[fruits.size()];
        fruits.toArray(fruitArray);

        // Printing the array
        System.out.println("Array:");
        for (String fruit : fruitArray) {
            System.out.println(fruit);
        }
    }
}
```

Output:

```
Array:
Apple
Banana
Orange
```

Example: Converting Array to ArrayList

```
import java.util.ArrayList;
import java.util.Arrays;

public class ArrayToArrayList {
    public static void main(String[] args) {
        String[] fruitArray = {"Apple", "Banana", "Orange"};

        // Converting Array to ArrayList
        ArrayList<String> fruits = new ArrayList<>(Arrays.asL
```

```
ist(fruitArray));

        // Printing the ArrayList
        System.out.println("ArrayList:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

Output:

```
ArrayList:
Apple
Banana
Orange
```

---

7. Best Practices for Using ArrayList

1. **Specify Initial Capacity**: If you know the expected size of your `ArrayList`, specify an initial capacity to optimize memory usage.

   ```
   ArrayList<String> fruits = new ArrayList<>(100);
   ```

2. **Avoid Using Raw Types**: Always specify the type of objects stored in the `ArrayList` using generics.

   ```
   ArrayList<String> fruits = new ArrayList<>();
   ```

3. **Use `ArrayList` When Frequent Insertions/Deletions at the End**: For frequent insertions/deletions at the beginning or middle, consider using `LinkedList`.

4. **Iterate Efficiently**: Use the enhanced `for` loop or `Iterator` for better performance when iterating through `ArrayList`.

5. **Remove Elements Safely**: When removing elements while iterating, use `Iterator.remove()` to avoid `ConcurrentModificationException`.

Summary

- `ArrayList` is a versatile, resizable array implementation in Java that is part of the Collections Framework.
- It allows for dynamic sizing, easy manipulation of elements, and provides a wide range of methods for managing collections.
- Understanding and using `ArrayList` efficiently can significantly improve your Java programming skills,