Chapter 05 ArrayLists

In Java, we need to declare the size of an array before we can use it. Unlike arrays, ArrayList can automatically adjust (grows or shrink) their capacity when we add or remove elements from them. Hence, ArrayList are also known as dynamic arrays.

ArrayList is a part of collection framework and is present in java. `java.util` package. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

ArrayList cannot be used for primitive types, like `int`, `char`, etc. We need a wrapper class for such cases.

1. Creating a ArrayLists

To use the ArrayList, we need to import the `java.util.ArrayList` package first.

Syntax for creating an Arraylist:

```
ArrayList<element-type> reference-variable = new ArrayList<>();
```

For example:

```
// Create Integer type arraylist
ArrayList<Integer> arrayList = new ArrayList<>();

// Create String type arraylist
ArrayList<String> arrayList = new ArrayList<>();
```

In the above program, we have used `Integer` not `int`. It is because ArrayLists store references. If you need to store primitives in an ArrayList, you cannot do it directly. Instead, you have to use its corresponding wrapper class. These wrapper classes are available in `java.lang` package.

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Primitive Data types and their Corresponding Wrapper class.

2. `ArrayList` Methods

The `ArrayList` class provides various methods to perform different operations on ArrayLists.

Method	*Description*
`add()`	Inserts the element to the ArrayList
`addAll()`	Adds all elements of a collection to ArrayList
`get()`	Returns the element present in the specified index
`set()`	Replace the single element from an ArrayList
`remove()`	Removes the single element from the ArrayList
`removeAll()`	Removes multiple elements from the arraylist
`clear()`	Removes all the elements from ArrayList
`size()`	Returns the length of the ArrayList.
`sort()`	Sort the ArrayList elements.
`clone()`	Creates a new ArrayList with the same element, size, and capacity.
`contains()`	Searches the ArrayList for the specified element and returns a `boolean` result.
`ensureCapacity()`	Specifies the total element the ArrayList can contain.
`isEmpty()`	Checks if the ArrayList is empty.
`indexOf()`	Searches a specified element in an ArrayList and returns the index of the element.
`size()`	Returns the length of the ArrayList.
`sort()`	Sort the ArrayList elements.

More ArrayList methods here: Java ArrayList Methods

3. `ArrayList` Initialization

3.1 Initialization with `add()` method

We can use `add()` method to initialize ArrayLists. For example:

```
ArrayList<String> arrayList = new ArrayList<>();
arrayList.add("Java");
arrayList.add("C++");
arrayList.add("Python");
```

We can use the shorthand version of the `add` () method. For example:

```
ArrayList<String> arrayList = new ArrayList<>() {
    {
        add("Java");
        add("C++");
        add("Python");
    }
};
```

3.2 Initialization using `asList()` method

We can use `asList()` method of `java.util.Arrays` class to initialize ArrayLists. For example:

```
ArrayList<String> arrayList = new ArrayList<>(
    Arrays.asList("Java", "C++","Python")
);
```

3.3 Initialization using Another Collection

We can initialize an ArrayList using another ArrayList:

```
ArrayList<String> arrayList = new ArrayList<>(anotherArrayList);
```

We can initialize an ArrayList using an array:

```
ArrayList<String> arrayList = new ArrayList<>(Arrays.asList(array));
```

Example 01: Create and initialize ArrayLIsts.

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

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

        // Create and initialize an ArrayList
        ArrayList<String> arrayList1 = new ArrayList<>();
        arrayList1.add("Java");
        arrayList1.add("C++");
        arrayList1.add("Python");

        // Create and initialize another ArrayList
        ArrayList<String> arrayList2 = new ArrayList<>() {
            {
                add("Java");
                add("C++");
                add("Python");
            }
        };

        // Create and initialize another ArrayList
        ArrayList<String> arrayList3 = new ArrayList<>(
            Arrays.asList("Java", "C++","Python")
        );

        // Create and initialize another ArrayList
        ArrayList<String> arrayList4 = new ArrayList<>(arrayList1);

        // Display the ArrayLists
        System.out.println(arrayList1);
        System.out.println(arrayList2);
        System.out.println(arrayList3);
        System.out.println(arrayList4);
    }
}
```

Output:

```
[Java, C++, Python]
[Java, C++, Python]
[Java, C++, Python]
[Java, C++, Python]
```

4. Basic Operations on `ArrayList`

4.1 Adding Elements

To add a single element to the Arraylist, we use the `add`() method of the `ArrayList` class.

Example 02: Add elements to an ArrayList using the `add`() method without the index parameter.

```
import java.util.ArrayList;

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

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

        // Add elements to the ArrayList
        languages.add("Java");
        languages.add("C/C++");
        languages.add("Python");

        System.out.println(languages);
        //System.out.println(languages.toString());
    }
}
```

Output:

```
ArrayList: [Java, C/C++, Python]
```

We can also pass an index number as an additional parameter to the add() method to add an element at the specified position.

Example 03:

```
import java.util.ArrayList;

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

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

        // Add elements to the ArrayList
        languages.add("Java");
        languages.add("C/C++");
        languages.add("Python");

        // add JavaScript at index 1
        languages.add(1, "JavaScript");

        // add SQL at index 3
        languages.add(3, "SQL");

        System.out.println(languages);
    }
}
```

Output:

```
[Java, JavaScript, C/C++, SQL, Python]
```

The `addAll`() method adds all the elements of a collection to the Arraylist.

Syntax of `ArrayList addAll()` method is:

```
arraylist.addAll(int index, Collection collection)
```

Here,

- `arraylist` is an object of the `ArrayList` class.
- `index` (optional) - index at which all elements of a collection is inserted. If the index parameter is not passed the collection is appended at the end of the arraylist.
- `collection` - collection that contains elements to be inserted

Example 04: Inserting Elements using `ArrayList addAll()` method.

```
import java.util.ArrayList;

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

        // Create an Arraylist
        ArrayList<String> languages1 = new ArrayList<>();

        // Create another Arraylist
        ArrayList<String> languages2 = new ArrayList<>();
        languages2.add("PHP");
        languages2.add("C#");

        // Create another Arraylist
        ArrayList<String> languages3 = new ArrayList<>();
        languages3.add("Java");
        languages3.add("Python");

        // Create another Arraylist
        ArrayList<String> languages4 = new ArrayList<>();
        languages4.add("R");
        languages4.add("C++");

        //Call addAll method
        languages1.addAll(languages4);
        languages2.addAll(languages4);
        languages3.addAll(1, languages4);

        System.out.println(languages1);
        System.out.println(languages2);
        System.out.println(languages3);
    }
}
```

Output:

```
[R, C++]
[PHP, C#, R, C++]
[Java, R, C++, Python]
```

The `addAll()` method:

- returns `true` if the collection is successfully inserted into the Arraylist
- raises `NullPointerException` if the specified collection is null
- raises `IndexOutOfBoundsException` if the index is out of range

4.2 Accessing Elements

To access an element from the Arraylist, we use the `get()` method of the `ArrayList` class.

Example 05:

```
import java.util.ArrayList;

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

        // Add elements in the arraylist
        animals.add("Cat");
        animals.add("Dog");
        animals.add("Cow");
        System.out.println("ArrayList: " + animals);

        // Get the element from the Arraylist
        String str = animals.get(1);
        System.out.print("Element at index 1: " + str);
    }
}
```

Output:

```
ArrayList: [Cat, Dog, Cow]
Element at index 1: Dog
```

4.3 Updating Elements

To change elements of the Arraylist, we use the `get()` method of the `ArrayList` class.

Example 06: In this example, we use the set() method changes the element at index 2 to `"JavaScript"`.

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

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

        ArrayList<String> languages = new ArrayList<>(
            Arrays.asList("Java", "C++", "Python")
        );

        languages.set(2, "JavaScript"); // Change the element of the ArrayList

        System.out.println(languages);
    }
}
```

Output:

```
[Java, C++, JavaScript]
```

4.4 Removing Elements

To remove an element from the Arraylist, we can use the `remove()` method of the `ArrayList` class.

Example 07: Remove an element from ArrayList.

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

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

        ArrayList<String> animals = new ArrayList<>();
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Horse");

        // Remove element from index 2
        String removeAnimal = animals.remove(2);
        System.out.println("Updated ArrayList: " + animals);
        System.out.println("Removed Element: " + removeAnimal);
    }
}
```

Output:

```
Updated ArrayList: [Dog, Cat]
Removed Element: Horse
```

We can also remove all the elements from the Arraylist at once using `removeAll()` and `clear()` methods. For example:

```
animals.removeAll(animals);    // remove all elements
animals.clear();               // remove all elements
```

4.5 Iterating Through an ArrayList

We can use the Java for-each loop to loop through each element of the Arraylist.

Example 08: Iterate an ArrayList

```
import java.util.ArrayList;

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

        // creating an array list
        ArrayList<String> animals = new ArrayList<>();
        animals.add("Cow");
        animals.add("Cat");
        animals.add("Dog");

        // Iterate using for-each loop
        for (String e : animals) {
            System.out.print(e + "\t");
        }
    }
}
```

Output:

```
Cow     Cat     Dog
```

4.6 Converting an `ArrayList` to an Array

We can convert the ArrayList into an array using the `toArray()` method.

Example 09:

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

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

        ArrayList<String> languages = new ArrayList<>(
            Arrays.asList("Java", "Python", "C++")
        );

        // Create a new ArrayList of String type
        String[] arrayList = new String[languages.size()];

        // Convert ArrayList into an array
        languages.toArray(arrayList);

        // Display the elements of the array
        for (String item : arrayList) {
            System.out.print(item + "\t");
        }
    }
}
```

Output:

```
Java    Python    C++
```

5. ArrayLists and Methods

We can pass an ArrayList to a method as well as return an ArrayList from a method.

5.1 Passing ArrayLists to a Method

When an ArrayList passes to a method, the address of the ArrayList is passed. Thus, any changes made to the ArrayList in the method will affect the original ArrayList.

Example 10: Passing an ArrayList to a method.

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

public class Example {
    static void addElement(ArrayList<Integer> arrayList){
        arrayList.add(30);
    }
    public static void main(String args[]) {

        // Create and initilize an ArrayList
        ArrayList<Integer> arrayList = new ArrayList<>(
            Arrays.asList(1, 2, 3, 4, 5)
        );

        addElement(arrayList);

        // Display the ArrayList
        System.out.println(arrayList);
    }
}
```

Output:

```
[1, 2, 3, 4, 5, 30]
```

5.2 Returning an ArrayList from a Method

Example 11: Return an ArrayList from a method.

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

public class Example {
    static ArrayList<Integer> removeElement(ArrayList<Integer> arrayList){
        if(arrayList.contains(3)){
            arrayList.remove(arrayList.indexOf(3));
        }

        return arrayList;
    }
    public static void main(String args[]) {

        // Create and initilize a 2D ArrayList
        ArrayList<Integer> arrayList = new ArrayList<>(
            Arrays.asList(1, 2, 3, 4, 5)
        );

        arrayList = removeElement(arrayList);

        // Display the 2D arrayList
        System.out.println(arrayList);
    }
}
```

Output:

```
[1, 2, 4, 5]
```

6. Multi-dimensional ArrayLists

An ArrayList can be placed inside anther ArrayList to create a multi-dimensional ArrayList. Multidimensional ArrayLists refer to an ArrayList of ArrayLists. A multi-dimensional ArrayList is almost similar to a multi-dimensional array but the difference is only the dynamic characteristic. Here, we do not need to predefine the size of rows and columns. When it is filled completely, the size increases automatically to store the next element.

Syntax:

```
ArrayList<ArrayList<element-type>> reference-variable  = new ArrayList<>();
```

For example:

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

6.1 Initializing Two-dimensional ArrayLists

Example 12: Using 2D ArrayLists.

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

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

        // Create and initilize a 2D ArrayList
        ArrayList<ArrayList<Character>> arrayList2D = new ArrayList<>(
            Arrays.asList(
                new ArrayList<>(Arrays.asList('A', 'B')),
                new ArrayList<>(Arrays.asList('C', 'D')),
                new ArrayList<>(Arrays.asList('E', 'F'))
            )
        );
        // Display the 2D arrayList
        System.out.println(arrayList2D);
    }
}
```

Output

```
[[A, B], [C, D], [E, F]]
```

Example 13: Another example of initializing 2D ArrayLists.

```
import java.util.ArrayList;

public class Example {
    public static void main(String args[]) {
        ArrayList<ArrayList<Integer>> arrayList2D = new ArrayList<>();

        for(int i = 0; i < 3; i++) {
            ArrayList<Integer> arrayList2D = new ArrayList<>();

            for(int j = 0; j < 4; j++) {
                arrayList1D.add((i * 4) + j);
            }
            arrayList2D.add(arrayList1D);
        }

        // Display the 2D arrayList
        System.out.println(arrayList2D);
    }
}
```

Output

```
[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
```

6.2 Iterating Through Two-dimensional ArrayLists

Example 14: Access 2D ArrayList elements.

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

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

        // Create and initilize a 2D ArrayList
        ArrayList<ArrayList<Integer>> arrayList2D = new ArrayList<>(
            Arrays.asList(
                new ArrayList<>(Arrays.asList(1)),
                new ArrayList<>(Arrays.asList(2, 3)),
                new ArrayList<>(Arrays.asList(4, 5, 6))
            )
        );

        // Access the 2D arrayList
        for (int i = 0; i < arrayList2D.size(); i++){
            for (int j = 0; j < arrayList2D.get(i).size(); j++){
                System.out.printf("%-3d", arrayList2D.get(i).get(j));
            }
            System.out.println();
        }

    }
}
```

Output:

```
1
2 3
4 5 6
```

Example 15: Access 2D ArrayList elements using range-base loop.

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

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

        // Create and initilize a 2D ArrayList
        ArrayList<ArrayList<Integer>> arrayList2D = new ArrayList<>(
            Arrays.asList(
                new ArrayList<>(Arrays.asList(1)),
                new ArrayList<>(Arrays.asList(2, 3)),
                new ArrayList<>(Arrays.asList(4, 5, 6))
            )
        );

        // Iterate the 2D ArrayList
        for (ArrayList<Integer> arrayList1D : arrayList2D){
            for (int element : arrayList1D){
                System.out.printf("%-3d", element);
            }
            System.out.println();
        }
    }
}
```

Output:

```
1
2 3
4 5 6
```

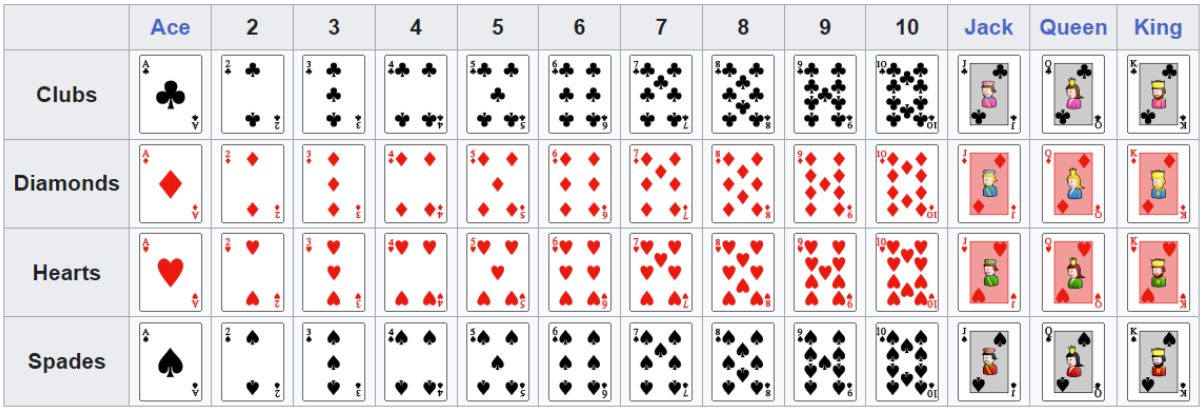
Exercises

Write the following programs using ArrayLists; you are **NOT** allowed to use standard arrays.

1. Write a program that first will read integers from the user into an ArrayList until the user enters 0. Then, display the ArrayList that contains no duplicate. Hint: store only the new entered integer; if the entered integer already existed in the ArrayList, do not store it.
2. *(Emirp)* An emirp (prime spelled backward) is a nonpalindromic prime number whose reversal is also a prime. For example, 17 is a prime and 71 is a prime, so 17 and 71 are emirps. Write a method called `generateRandomEmirp`(n) that will randomly generate n emirp(s). Display ten emirps per line as follows:

```
Enter the number of emirps you want the program to generate: 20
13    17    31    37    71    73    79    97    107   113
149   157   167   179   199   311   337   347   359   389
```

3. Write a method called `removeInt` that will accept an integer and an ArrayList, then remove all the occurrences of the integer from the ArrayList.
4. Write a method called `removeDupplicate` that will remove the duplicates from an integer ArrayList that is passed to it, then returns the ArrayList.
5. Write the method `rotateRight` that takes an array of integers and rotates the contents of the array to the right by two slots. Numbers that fall off the right should cycle back to the left. For example:
   - if the input array is {1, 3, 5, 7} then the rotated array should be {5, 7, 1, 3}.
   - If the input array is {1, 2, 3} then the rotated array should be {2, 3, 1}.
6. *(Locate the smallest element)* Write the following method called `locateSmallest` that returns the location of the smallest element in a 2D ArrayList. The return value is a 1D ArrayList that contains two elements. These two elements indicate the row and column indices of the smallest element in the 2D ArrayList. Write a test program that randomly generates a 3-by-4 2D ArrayList of integers in range of [10, 99], and displays the location (row index and column index) of the smallest element in the ArrayList.
7. Write a program to simulate two players (user and computer) playing a card game (ដុកកាំង). Here is a standard 52-card deck:



Reference

[1] Y. Daniel Liang. 'Introduction to Java Programming', 11e – 2019

[2] https://www.programiz.com/java-programming