DAY 4

ARRAY, ARRAY LIST ,LOOPS

- ARRAY

In Java, **array** and **ArrayList** are the well-known data structures. An array is a basic functionality provided by Java, whereas **ArrayList** is a class of Java Collections framework. It belongs to **java.util** package.

Java Array

An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location. Once an array is created, we cannot change its size. We can create an array by using the following statement:

int array[]=**new int**[size];

The above statement creates an array of the specified size. When we try to add more than its size, it throws **ArrayIndexOutOfBoundsException**. For example:

```
int arr[]=new int[3];   //specified size of array is 3
//adding 4 elements into array
arr[0]=12;
arr[1]=2;
arr[2]=15;
arr[3]=67;
```

- Java ArrayList class

In Java, **ArrayList** is a class of Collections framework. It implements **List<E>, Collection<E>, Iterable<E>, Cloneable, Serializable**,
and **RandomAccess** interfaces. It extends **AbstractList<E>** class.

We can create an instance of ArrayList by using the following statement:

ArrayList<Type> arrayList=**new** ArrayList<Type>();

ArrayList is internally backed by the array in Java.

We cannot store primitive type in ArrayList. So, it stores only objects. It automatically converts primitive type to object. For example, we have create an ArrayList object,

ArrayList <Integer> list=**new** ArrayList<Integer>(); //object of ArrayList
arrayObj.add(12); //trying to add integer primitive to the ArrayList

Similarities

- Array and ArrayList both are used for storing elements.
- Array and ArrayList both can store null values.
- They can have duplicate values.
- They do not preserve the order of elements.

Example of Array in Java

In the following example, we have simply created an array of length four.

```
public class ArrayExample
{
public static void main(String args[])
{
//creating an array of integer type
int arr[]=new int[4];
//adding elements into array
arr[0]=12;
arr[1]=2;
arr[2]=15;
arr[3]=67;
for(int i=0;i<arr.length;i++)
{
System.out.println(arr[i]);
}
}
}
```

Output:

```
12
2
15
67
```

Example of ArrayList in Java

In the following example, we have created an instance of ArrayList and performing iteration over the ArrayList.

```
import java.util.*;
public class ArrayListExample
{
public static void main(String args[])
{
//creating an instance of ArrayList
List<Float> list = new ArrayList<Float>();
//adding element to arraylist
list.add(12.4f);
list.add(34.6f);
list.add(56.8f);
list.add(78.9f);
//iteration over ArrayList using for-each loop
for(Float f:list)
{
System.out.println(f);
}
}
}
```

Output:

```
12.4
34.6
56.8
78.9
```

ARRAY
VERSUS
ARRAYLIST

ARRAY	ARRAYLIST
A data structure consisting of a collection of elements each identified by the array index	A class that supports dynamic arrays which can grow as needed
A part of core Java programming	A part of Collection framework with other classes such as Vector, HashMap, etc.
Programmer can use the assignment operator to store elements into the array	Programmer can use the add method to insert elements
Can contain primitives or objects	Can only store objects
Helps to implement a fixed size data structure	Helps to implement dynamic size arrays

- LOOPS

There are two kind of loops in Java, for and while.

- **For**

The for loop has three sections:

```
for (int i = 0; i < 3; i++) {}
```

First section runs once when we enter the loop.

Second section is the gate keeper, if it returns true, we run the statements in the loop, if it returns false, we exit the loop. It runs right after the first section for the first time, then every time the loop is finished and the third section is run.

The third section is the final statement that will run every time the loop runs.

So in the case we have just seen, the loop will run 3 times. Here is the order of the commands:

```
int i = 0;
i < 3 // 0 < 3 = true
    // Inside of loop
i++ // i is now 1
i < 3 // 1 < 3 = true
    // Inside of loop
i++ // i is now 2
i < 3 // 2 < 3 = true
    // Inside of loop
i++ // i is now 3
i < 3 // 3 < 3 = false
    // Loop is done...
```

We can omit the first and third section of the loop (although it will be weird), and the loop will still work:

```
for (;i < 5;) {}
```

For cases where we want to use a loop that look like that, we use a while loop

- **While**

The syntax is very similar to the previous for we looked at:

```
while (condition) {}
```

The condition will run for the first time when entering and every time the loop is done. If it returns false, the loop will not run.

If we want the loop to always run at least one, we can use do-while

```
do {


} while(condition);
```

Notice the ; in the end of the do-while.

- **Foreach**

Another version of for, is the foreach. The keyword we use is still for, but when we want to iterate on the elements inside an array we can simply use it:

```
int[] arr = {2, 0, 1, 3};
for (int el : arr) {
   System.out.println(el);
}
```

This is a short version and equivalent to:

```
int[] arr = {1, 9, 9, 5};
for (int i = 0; i < arr.length; i++) {
   int el = arr[i];
   System.out.println(el);
}
```

Notice that if you want to use the index of the element inside the loop, you have to use the longer version and can't use foreach.

break and continue

These two keywords help us control the loop from within it. break will cause the loop to stop and will go immediately to the next statement after the loop:

```
int i;
```

```
for (i = 0; i < 5; i++) {
    if (i >= 2) {
        break;
    }
    System.out.println("Yuhu");
}
System.out.println(i);
// Output:
// Yuhu
// Yuhu
// 2
```

continue will stop the current iteration and will move to the next one. Notice that inside a for loop, it will still run the third section.

```
int i;
for (i = 0; i < 5; i++) {
    if (i >= 3) {
        break;
    }
    System.out.println("Yuhu");
    if (i >= 1) {
        continue;
    }
    System.out.println("Tata");
}
System.out.println(i);
// Output
// Yuhu
// Tata
// Yuhu
// Yuhu
// 3
```