

# INTERN CODING CHALLENGE

## The Challenge

The challenge is to write a function that accepts an arbitrarily-deep nested Array-like structure and returns a flattened structure with any null values removed. Include some documentation about how your solution works, including any cases in which it might fail. If time permits, please include a few test cases. You may implement the challenge using whatever programming language you're most comfortable with.

## LANGUAGE – JAVA

### APPROACH

I am using the generic type in Java, initially passing the collection type, which is an array-like structure, it includes ArrayList, List, LinkedList and various other structures. Later when I am executing the function. I am checking whether the passed argument is null or not.

- a. **If null**, executing the else condition and printing “Collection is null”.
- b. **If not null**, Using for loop for each element. In the loop, we again check for null, so that we pass only the not null element. Every element can be an another collection of object type, an array or a data type.
  - a. If it is an instance of collection, call the flatten function (recursion).
  - b. If it is an array, we are going to get the component type of the object, and then check whether it is of primitive data type or not. Once we established that it is of primitive datatype, we are going to identify the datatype and then traverse through that array and store the value.
  - c. Else, store the element in an ArrayList (array structure) of object type, which can store any data type.
- c. Return the resultant ArrayList.

**Below is the step by step explanation of my code:**

```
/* Declaring and initializing an ArrayList (array like datastructure) to
store the resultant array. ArrayList can store any data type.
```

```
It is of object type */
```

```
public static ArrayList<Object> result = new ArrayList<>();
```

```
/*
```

```
flatten function whose return type is array like structure
and taking the Generic Collection as input type.
```

It can be any Array like structure (example List, LinkedList, HashSet) structure and can be deep nested. This fulfill are requirement for deep-nested array like structure.

## INTERN CODING CHALLENGE

<T> is used for defining the Generic datatype, It would be of object type, which can be primitive data type or other collection  
Collection is used for Generic Array like structure.

```
*/  
public static <T> ArrayList<Object> flatten(Collection<T> input) {  
  
    /* This if condition checks, if the passed argument is null or not.  
    If the collection is null, the function will not  
    enter the if loop and just pass the message,  
    collection is null  
    */  
    if(input != null) {  
        /*  
        Once we determine the collection is not null.  
        We are going to iterate through each element using  
        for loop. Each element can be another collection  
        (array-like structure) or primitive data type.  
        */  
        for(T type : input) {  
  
            // If the object is not null, then only it will pass through this  
            if(type != null) {  
  
                // this if condition checks whether the object is instance of collection  
                if(type instanceof Collection) {  
  
                    /*  
                    If the object is of type Collection,  
                    same flatten function is going to be  
                    called using recursion  
                    */  
                    flatten((Collection)type);  
                }  
                // this condition checks whether the object is an array or not  
                else if(type.getClass().isArray()) {  
  
                    // get the componentType of the object  
                    Class<?> componentType;  
                    componentType = type.getClass().getComponentType();  
  
                    // this condition checks whether the object is an array of primitive datatype  
                    // or user defined datatype. This will work only for primitive data type  
                    if (componentType.isPrimitive()) {
```

## INTERN CODING CHALLENGE

```
// If the array is of int datatype, this condition for traversing through
// that and insert the data into the array
if (int.class.isAssignableFrom(componentType)) {
    for (int element : (int[]) type) {
        result.add(element);
    }
}
else if (boolean.class.isAssignableFrom(componentType)) {
    for (boolean element : (boolean[]) type) {
        result.add(element);
    }
}
else if (byte.class.isAssignableFrom(componentType)) {
    for (byte element : (byte[]) type) {
        result.add(element);
    }
}
else if (char.class.isAssignableFrom(componentType)) {
    for (char element : (char[]) type) {
        result.add(element);
    }
}
else if (double.class.isAssignableFrom(componentType)) {
    for (double element : (double[]) type) {
        result.add(element);
    }
}
else if (float.class.isAssignableFrom(componentType)) {
    for (float element : (float[]) type) {
        result.add(element);
    }
}
else if (long.class.isAssignableFrom(componentType)) {
    for (long element : (long[]) type) {
        result.add(element);
    }
}
else if (short.class.isAssignableFrom(componentType)) {
    for (short element : (short[]) type) {
        result.add(element);
    }
}
}
```

## INTERN CODING CHALLENGE

```
/* this condition checks whether the object is anything
other than collection or array, i.e., it is a primitive datatype */
else {

    //element is going to be save in the resultant arrayList */
    result.add(type);
}
}
}
}
// Else loop will execute, if the collection is null
else {
    System.out.println("Passed collection is null");
}

/*
Returning the resultant arrayList(flattened array) whose
has all the values except null. It is not deep nested array
*/
return result;
}
```

### Test Cases:

1. {0, 2, {{2, 3}, 8, 100, null, {{null}}}, -2}};
2. null
3. {}
4. {{{{}}}}
5. {{null, null}, {3, -10, 4.7}, {}}
6. Large Values of 10 digits