# A Bit More About Arrays

In this lesson, an explanation of Character Arrays, String Arrays, Arrays & Methods is provided.

**We'll cover the following**



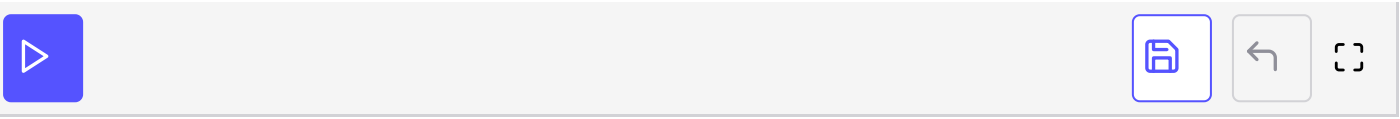
As we have covered the basics of Arrays in the previous lesson, let's discuss how to declare and use arrays of `char` & `String` data types.

## Character Arrays #

We can declare an Array of `char` in which we can store characters. Let's write a code to declare and initialize a character array:

```
class CharArr {
    public static void main(String args[]) {
        char[] chArray1 = {
            'a',
            'b',
            'c'
        }; //initialization of first char array
        char[] chArray2 = new char[5]; //instantiation of second char array
        int index = 0;
        for (char i = 'v'; i <= 'z'; i++) { //Assiging char values to second array using loop

            chArray2[index] = i;
            index++;
        }
        //printing out the stored values in the arrays
        System.out.print("The Values stored in ChArray1 are: ");

        for (int i = 0; i < chArray1.length; i++) {

            System.out.print(chArray1[i] + " ");
        }

        System.out.println();
        System.out.print("The Values stored in ChArray2 are: ");

        for (int i = 0; i < chArray2.length; i++) {
```

```
            System.out.print(chArray2[i] + " ");
        }
    }
}
```



Character Arrays

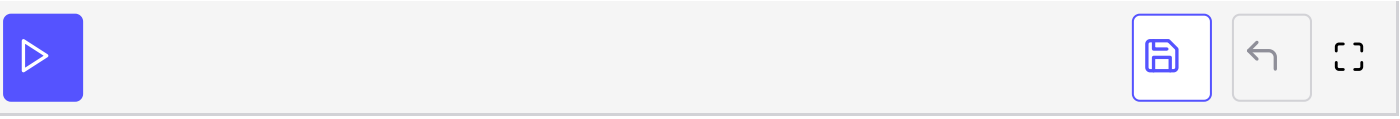
The above example has a loop that uses `char` variable. Each `char` value is represented by a number, so `i++` actually increments the internal code, which makes us hop from `v` to `w` and so on.

> Note that while storing a value in `char` array the literal should be inside pair of single quote like `chArray = 'a';`

## String Arrays #

We can also store *Strings* in an Array, let's get familiar to `String` arrays through the below code:

```
class StrArray {
    public static void main(String args[]) {
        String[] strArray = {
            "A",
            "Quick",
            "Brown",
            "Fox",
            "Jumps",
            "Over",
            "a",
            "Lazy",
            "Dog"
        }; //initialization of first char array


        System.out.print("The Values stored in strArray are: ");

        for (int i = 0; i < strArray.length; i++) {
            System.out.print(strArray[i] + " ");
        }
    }
}
```



String Array

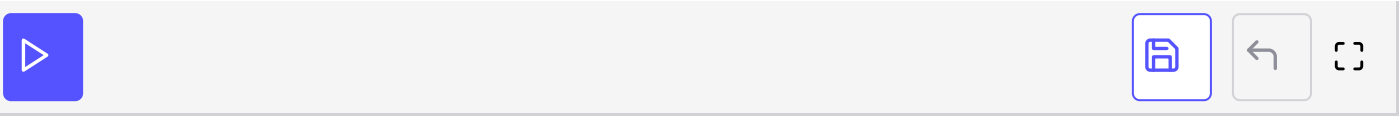
> Note that while storing a value in `String` array the literal should be inside pair of double quote like `strArray = "a";`

## Index out of bound exception

If we try to index beyond `array.length-1`, the runtime throws an exception and ends the program.

> **Warning:** The program given below will generate an exception.

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

        int[] myArray = new int[10]; // Declaration and Instantiation of Array with length 10

        for (int i = 0; i < myArray.length; i++) // Iterate through indexes 0-9
        {
            myArray[i] = i + 1; // Initialize values to 1-10
        }

            System.out.println("The value at myArray[11]] is:" + myArray[11]);

    }
}
```



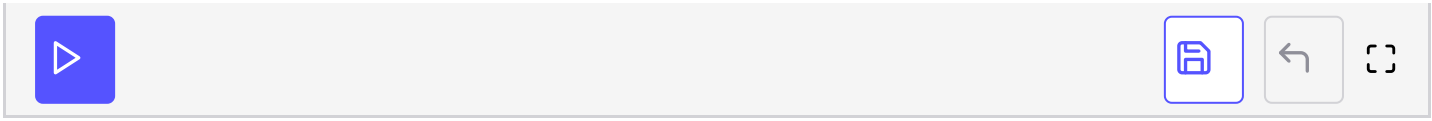
Initialization using loops

# Arrays & Methods

An array can be declared in a calling function and passed to a function as an argument. Any changes made to the argument array in the called function are actually performed directly on the array defined in the calling function. Let's see this in action.

```
class ArrayToMethod {
    public static void main(String args[]) {
        int[] arr = {
            1,
            2,
            3,
            4
```

```
            4,
            5
        }; //initialize
        System.out.println("The Values before calling the method are:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " "); //printing the array before calling method

        }
        int[] returnedArr = multiply(arr, 3); //storing the returned array
        System.out.println();
        System.out.println("The Values from the returned Array are:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(returnedArr[i] + " "); //printing the returned array

        }
    } //end of main()

    static int[] multiply(int[] arr, int num) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = arr[i] * num;
        }
        return arr; //returning Array
    }
}
```



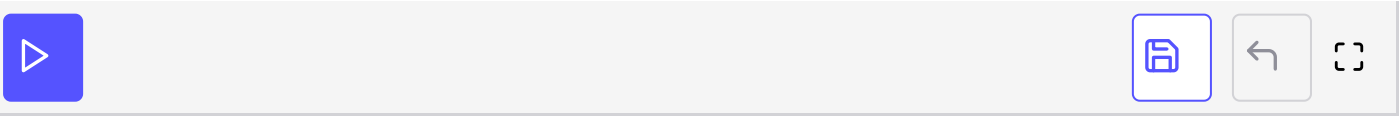
Arrays & Methods

In the above program, the *called* method is `multiply(int[] arr, int num)` and the *calling* method is `main()`.

In Java whenever **primitive** datatypes are passed as an argument to a method a copy of the variable is made and passed to that method which means if *called method* makes any changes to the passed values it is not visible to the *calling method*. As an array is a data structure rather than a primitive data type, it is passed by reference to a method which means that if the *called method* alters any value of an array this alteration will be visible to the *calling method* also.
Let's experience the above said through making some changes to the coding example given before:

```
class ArrayToMethod {
    public static void main(String args[]) {
        int[] arr = {
            1,
            2,
            3,
            4,
            5
        }; //initialize

        System.out.println("The Values before calling the method are:");
```

```
        for (int i = 0; i < arr.length; i++) {
            System.out.print("arr[" + i + "]= " + arr[i] + "  "); //printing the array before call
        }

        System.out.println();
        multiply(arr, 3); //nothing is being returned
        System.out.println("The Values after calling the method are:");
        for (int i = 0; i < arr.length; i++) {
            System.out.print("arr[" + i + "]= " + arr[i] + "  "); //printing the array after calli

        }

    } //end of main()

    static void multiply(int[] arr, int num) { //will change the values of passed array
        for (int i = 0; i < arr.length; i++) {
            arr[i] = arr[i] * num;
        }
        //Not returning anything
    }
}
```



Arrays by Reference

We can notice in the above code that the *called method* has changed the values of the array passed to it and this change is visible to the *calling method*.

---

In the next lesson, we will discuss another type of Arrays known as two-dimensional Arrays.