Continue statement inside for loop

In this program, illustration for how to use the continue statement within For loop. When the value of ‘i’ becomes 10 or 12, the continue statement plays its role and skip their execution but for other values of’ ‘i’ the loop will run smoothly.

```
// Java Program to illustrate the use of continue statement

// Importing Classes/Files

import java.util.*;

public class GFG {

	// Main driver method

	public static void main(String args[])

	{

		// For loop for iteration

		for (int i = 0; i <= 15; i++) {

			// Check condition for continue

			if (i == 10 || i == 12) {

				// Using continue statement to skip the

				// execution of loop when i==10 or i==12

				continue;				}

			// Printing elements to show continue statement

			System.out.print(i + " ");

		}

	}

}
```

Continue statement inside while loop

```
// Java Program to illustrate the use of continue statement

// inside the While loop

public class GFG {

        // Main driver method

        public static void main(String args[])

        {

        // Initializing a variable say it count to a value

                // greater than the value greater among the loop

                // values

                int count = 20;

                // While loop for iteration

                while (count >= 0) {

                        if (count == 7 || count == 15) {

                                count--;

                                // Decrementing variable initialized above

                                // Showing continue execution inside loop

                                // skipping when count==7 or count==15

                                continue;

                        }

                        // Printing values after continue statement

                        System.out.print(count + " ");



                        // Decrementing the count variable

                        count--;

                }

        }
}
```

Continue statement inside Inner loop(Nested Loop)

In the below program, we give example, how to use the continue statement within Nested loops. When the value of i becomes 3 and j become 2, the continue statement plays its role and skip their execution but for other values of i and j, the loop will run smoothly.

// Java Program to illustrate the use of continue statement

```
// inside an inner loop or simply nested loops

// Importing generic Classes/Files

import java.util.*;

public class GFG {

        // Main drive method

        public static void main(String[] args)

        {

                // Outer loop for iteration

                for (int i = 1; i <= 4; i++) {

                        // Inner loop for iteration

                        for (int j = 1; j <= 3; j++) {

                                if (i == 3 && j == 2) {

                                        // Continue statement in inner loop to

                                        // skip the execution when i==3 and j==2

                                        continue;

                                }

                                // Print elements to showcase keyword affect

                                System.out.println(i + " * " + j);

                        }

                }

        }

}
```

Break statement in Java

Break Statement is a loop control statement that is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

```
// Java program to illustrate using
// break to exit a loop
class BreakLoopDemo {
	public static void main(String args[])
	{
		// Initially loop is set to run from 0-9
		for (int i = 0; i < 10; i++) {
			// terminate loop when i is 5.
			if (i == 5)
				break;

			System.out.println("i: " + i);
		}
		System.out.println("Loop complete.");
	}
}
```

Another Example:

```
// Java program to illustrate
// using break with goto
class BreakLabelDemo {
	public static void main(String args[])
	{
		boolean t = true;
	// label first
	first : {
		// Illegal statement here
	// as label second is not
	// introduced yet break second;
	second : {
	third : {
		// Before break
		System.out.println("Before the break statement");
		// break will take the control out of
		// second label
		if (t)
			break second;
```

```
                        System.out.println("This won't execute.");
            }
                        System.out.println("This won't execute.");
            }
                        // First block
                        System.out.println("This is after second block.");
            }
            }
}
```

return keyword in Java

In Java, **return** is a reserved keyword i.e, we can’t use it as an identifier. It is used to **exit** from a method, with or without a value. Usage of **return keyword** as there exist two ways as listed below as follows:

- **Case 1:** Methods returning a value
- **Case 2:** Methods not returning a value

```
// Java Program to Illustrate Usage of return Keyword
// Main method
class GFG {
            // Method 1
            // Since return type of RR method is double
            // so this method should return double value
            double RR(double a, double b) {
                        double sum = 0;
                        sum = (a + b) / 2.0;
                                    // Return statement as we already above have declared
                        // return type to be double
                        return sum;
            }
            // Method 2
            // Main driver method
            public static void main(String[] args)
            {
                        // Print statement
                        System.out.println(new GFG().RR(5.5, 6.5));
            }
}
```

Methods not returning a value

For methods that do not return a value, return statement in Java can be skipped. here there arise two cases when there is no value been returned by the user as listed below as follows:

- **#1:** Method not using return statement in void function
- **#2:** Methods with return type void

#1: Method not using return statement in void function

Example

```
// Java program to illustrate no return
// keyword needed inside void method
// Main class
class GFG {

	// Since return type of RR method is
	// void so this method shouldn't return any value
	void demoSum(int a, int b)
	{
		int sum = 0;
		sum = (a + b) / 10;
		System.out.println(sum);

		// No return statement in this method
	}

	// Method 2
	// Main driver method
	public static void main(String[] args)
	{
		// Calling the method
		// Over custom inputs
		new GFG().demoSum(5, 5);

		// Display message on the console for successful
		// execution of the program
		System.out.print(
			"No return keyword is used and program executed successfully");
	}

	// Note here we are not returning anything
	// as the return type is void
}
```

Methods with ***void return type***

```
// Java program to illustrate usage of
// return keyword in void method
// Class 1
// Main class
class GFG {
        // Method 1
        // Since return type of RR method is
        // void so this method should not return any value
        void demofunction(double j)
        {
        if (j < 9)
                // return statement below(only using
                        // return statement and not returning
                        // anything):
                        // control exits the method if this
                        // condition(i.e, j<9) is true.
                        return;
                ++j;
        }
        // Method 2
        // Main driver method
        public static void main(String[] args)
        {
                // Calling above method declared in above class
                new GFG().demofunction(5.5);
                // Display message on console to illustrate
                // successful execution of program
                System.out.println("Program executed successfully");
        }
}
```

Note: return statement **need not to be last statement in a method, but it must be last statement to execute** in a method.

```
// Java program to illustrate return must not be always

// last statement, but must be last statement

// in a method to execute

// Main class

class GFG {

        // Method 1

        // Helper method

        // Since return type of RR method is void

        // so this method should not return any value

        void demofunction(double i)

        {

                // Demo condition check

                if (i < 9)

                        // See here return need not be last

                        // statement but must be last statement

                        // in a method to execute

                        return;

                else

                        ++i;

        }


        // Method 2

        // main driver method

        public static void main(String[] args)

        {

                // Calling the method

                new GFG().demofunction(7);


                // Display message to illustrate

                // successful execution of program

                System.out.println("Program executed successfully");

        }

}
```

Example

```
// Java program to illustrate usage of
// statement after return statement
// Main class
class GFG {
        // Since return type of RR method is void
        // so this method should return any value
        // Method 1
        void demofunction(double j)
        {
                return;
                // Here get compile error since can't
                // write any statement after return keyword
                ++j;
        }
        // Method 2
        // Main driver method
        public static void main(String[] args)
        {
                // Calling the above defined function
                new GFG().demofunction(5);
        }
}
```

For-each loop in Java

For-each is another array traversing technique like for loop, while loop, do-while loop introduced in Java5.

- It starts with the keyword **for** like a normal for-loop.
- Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
- In the loop body, you can use the loop variable you created rather than using an indexed array element.

- It's commonly used to iterate over an array or a Collections class (eg, ArrayList)

Syntax:

```
for (type var : array)
{
    statements using var;
}
```

is equivalent to:

```
for (int i=0; i<arr.length; i++)
{
    type var = arr[i];
    statements using var;
}
```

```
import java.util.Arrays;


public class forEach {
    public static void main(String[] args) {
        String[] letters = new String[]{"a", "b", "c"};
        Arrays.stream(letters).forEach((String letter) -> {
            System.out.println(letter);
        });
    }
}
```

Output

```
a
b
c
```

forEach on a List

Using `forEach` on a `List` is just as simple as using it on a `Stream` in the above array example:

```
List<Integer> numbers = Arrays.asList(1,2,3);
numbers.forEach((Integer number) -> {
    System.out.println(number);
});
```

Output

```
1
2
3
```

`Here we used a anonymous function, but we could have used a method reference had we wanted to, just like with arrays.`

forEach on a Map

When using `forEach` on a `Map`, things are slightly different than with arrays or a `List` since we have both a key and a value so we need to use an anonymous function that implements the `BiConsumer` interface rather than just a simple `Consumer` as before:

```
Map<String, String> countries = new HashMap<>();


countries.put("fr", "France");
countries.put("us", "United States");
countries.put("nz", "New Zealand");

countries.forEach((String key, String value) -> {
    System.out.println("Key: " + key + " Value: " + value);
});
```

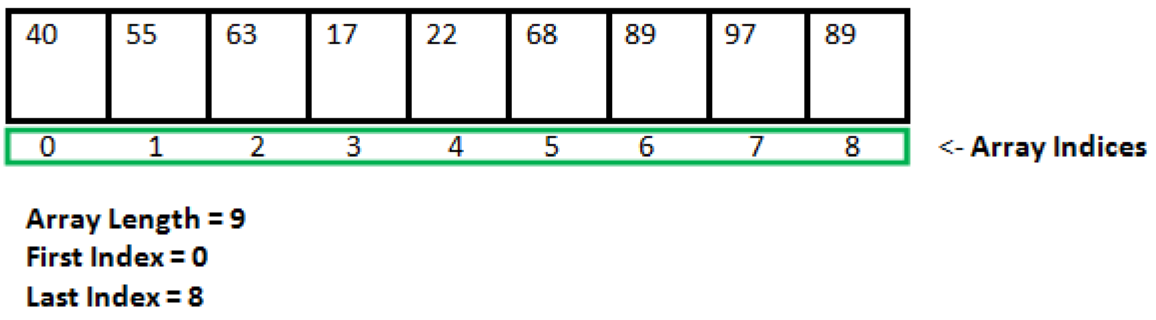
Output

```
Key: fr Value: France
Key: nz Value: New Zealand
Key: us Value: United States
```

Arrays in Java

An array in Java is a group of like-typed variables referred to by a common name. Arrays in Java work differently than they do in C/C++.

An array can contain primitives (int, char, etc.) and object (or non-primitive) references of a class depending on the definition of the array. In the case of primitive data types, the actual values are stored in contiguous memory locations. In the case of class objects



Instantiating an Array in Java

When an array is declared, only a reference of an array is created. To create or give memory to the array, you create an array like this: The general form of *new* as it applies to one-dimensional arrays appears as follows:

```
var-name = new type [size];
```

Here, *type* specifies the type of data being allocated, *size* determines the number of elements in the array, and *var-name* is the name of the array variable that is linked to the array. To use *new* to allocate an array, **you must specify the type and number of elements to allocate.**

Example:

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

OR

```
int[] intArray = new int[20]; // combining both statements in one
```

Note :

1. The elements in the array allocated by *new* will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types).
2. Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory to hold the array, using new, and assign it to the array variable. Thus, **in Java**, **all arrays are dynamically allocated.**

Array Literal

In a situation where the size of the array and variables of the array are already known, array literals can be used.

```
 int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
 // Declaring array literal
```

- The length of this array determines the length of the created array.
- There is no need to write the new int[] part in the latest versions of Java.

Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

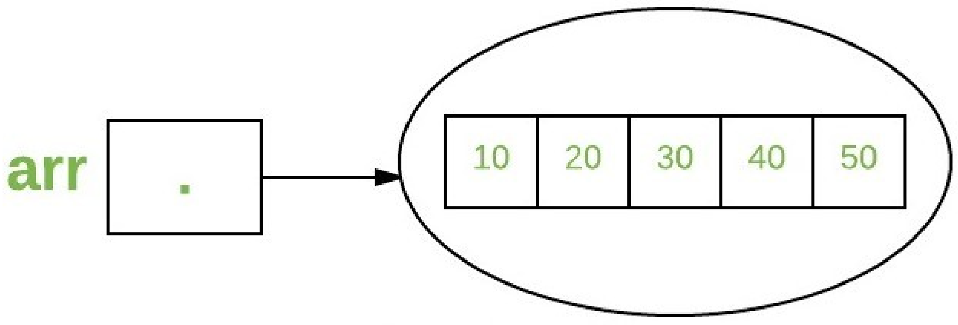
```
 // accessing the elements of the specified array
for (int i = 0; i < arr.length; i++)
  System.out.println("Element at index " + i +
                                " : "+ arr[i]);
```

Implementation:

```
// Java program to illustrate creating an array

// of integers, puts some values in the array,

// and prints each value to standard output.


class GFG {

      public static void main(String[] args)

      {

            // declares an Array of integers.

            int[] arr;

            // allocating memory for 5 integers.

            arr = new int[5];

            // initialize the first elements of the array

            arr[0] = 10;

            // initialize the second elements of the array

            arr[1] = 20;


            // so on...

            arr[2] = 30;

            arr[3] = 40;
```

```
                arr[4] = 50;

                // accessing the elements of the specified array

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

                        System.out.println("Element at index " + i

                                                           + " : " + arr[i]);

        }

}
```

You can also access java arrays using for each loops.



One-Dimensional Array

Arrays of Objects

An array of objects is created like an array of primitive type data items in the following way.

```
Student[] arr = new Student[7]; //student is a user-defined class
```

The studentArray contains seven memory spaces each of the size of student class in which the address of seven Student objects can be stored. The Student objects have to be instantiated using the constructor of the Student class, and their references should be assigned to the array elements in the following way.

```
Student[] arr = new Student[5];
```

Example

```
// Java program to illustrate creating

// an array of objects

class Student {

        public int roll_no;

        public String name;

        Student(int roll_no, String name)
```

```
	{
		this.roll_no = roll_no;
		this.name = name;
	}
}
// Elements of the array are objects of a class Student.
public class GFG {
	public static void main(String[] args)
	{
		// declares an Array of integers.
		Student[] arr;

		// allocating memory for 5 objects of type Student.
		arr = new Student[5];

		// initialize the first elements of the array
		arr[0] = new Student(1, "aman");

		// initialize the second elements of the array
		arr[1] = new Student(2, "vaibhav");
		// so on...
		arr[2] = new Student(3, "shikar");
		arr[3] = new Student(4, "dharmesh");
		arr[4] = new Student(5, "mohit");
		// accessing the elements of the specified array
		for (int i = 0; i < arr.length; i++)
			System.out.println("Element at " + i + " : "
							+ arr[i].roll_no + " "
							+ arr[i].name);
	}
}
```

Output

```
Element at 0 : 1 aman
Element at 1 : 2 vaibhav
Element at 2 : 3 shikar
Element at 3 : 4 dharmesh
Element at 4 : 5 mohit
```

Example

An array of objects is also created like an array of primitive type data items.

```
Student[] studentArray; // No array created

Student[] studentArray = new Student[5]; // Array of 5 students
created but No students are there in the array
```

The studentArray contains five memory spaces each of the size of the student class in which the address of seven Student objects can be stored. The Student objects have to be instantiated using the constructor of the Student class, and their references should be assigned to the array elements in the following way.

```
// Java program to illustrate creating
// an array of objects

class Student
{
	public String name;
	Student(String name)
	{
		this.name = name;
	}
	@Override
	public String toString(){
		return name;
	}
}

// Elements of the array are objects of a class Student.
public class GFG
{
	public static void main (String[] args)
	{
		// declares an Array and initializing the elements of the array
		Student[] myStudents = new Student[]{new Student("Dharma"),new
Student("sanvi"),new Student("Rupa"),new Student("Ajay")}; // Array of 5 students created
but No students are there in the array
			// accessing the elements of the specified array
		for(Student m:myStudents){
			System.out.println(m);
		}
	}
}
```

Output

```
Dharma
sanvi
Rupa
Ajay
```

What happens if we try to access elements outside the array size?

JVM throws ArrayIndexOutOfBoundsException to indicate that the array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of an array.

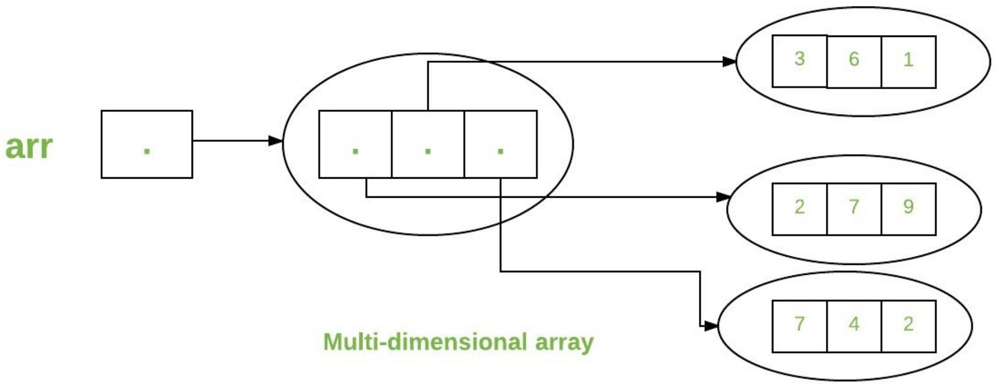
```
public class GFG {
        public static void main(String[] args)
        {
                int[] arr = new int[2];
                arr[0] = 10;
                arr[1] = 20;

                for (int i = 0; i < arr.length; i++)
                        System.out.println(arr[i]);
        }
}
```

Output

```
10
20
```

Multidimensional Arrays:

Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other arrays. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.



```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```

Example

```
public class multiDimensional {
        public static void main(String args[])
        {
                // declaring and initializing 2D array
                int arr[][]
                        = { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

                // printing 2D array
                for (int i = 0; i < 3; i++) {
                        for (int j = 0; j < 3; j++)
                                System.out.print(arr[i][j] + " ");

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

Output

```
2 7 9
3 6 1
7 4 2
```