

# Control Flow and Enum Constants

## Control Statements

A programming language uses control statements to cause the flow of execution to advance and branch based on changes to the state of a program. Java's program control statements can be put into the following categories: selection, iteration, and jump.

Selection statements allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable. Iteration statements enable program execution to repeat one or more statements (that is, iteration statements form loops). Jump statements allow your program to execute in a nonlinear fashion.

## Conditional Statements

The conditional statement in Java has the form

**if (condition) statement;**

The condition must be surrounded by parentheses. In Java, as in most programming languages, you will often want to execute multiple statements when a single condition is true. In this case, use a block statement that takes the form

```
{  
statement1;  
statement2;  
...  
}
```

For example:

```
if (yourSales >= target){  
    performance = "Satisfactory";  
    bonus = 100;
```

# Control Flow and Enum Constants

```
}
```

```
if (condition) statement1;
```

```
else statement2;
```

```
int a, b;
```

```
if(a < b) a = 0;
```

```
else b = 0;
```

## Nested ifs

A nested if is an if statement that is the target of another if or else.

```
if (i == 10) {
```

```
    if (j < 20)
```

```
        a = b;
```

```
    if (k > 100)
```

```
        c = d; // this if is
```

```
    else
```

```
        a = c; // associated with this else
```

```
} else {
```

```
    a = d; // this else refers to if(i == 10)
```

```
}
```

## The if-else-if Ladder

A common programming construct that is based upon a sequence of nested ifs is the if-else-if ladder. It looks like this:

```
if(condition)
```

```
    statement;
```

# Control Flow and Enum Constants

```
else if(condition)
```

```
    statement;
```

```
else if(condition)
```

```
    statement;
```

```
...
```

```
else
```

```
    Statement;
```

The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

```
class IfElse {  
    public static void main(String args[]) {  
        int month = 4; // April  
        String season;  
        if (month == 12 || month == 1 || month == 2) {  
            season = "Winter";  
        } else if (month == 3 || month == 4 || month == 5) {  
            season = "Spring";  
        } else if (month == 6 || month == 7 || month == 8) {  
            season = "Summer";  
        } else if (month == 9 || month == 10 || month == 11) {  
            season = "Autumn";  
        } else {  
            season = "Bogus Month";  
        }  
        System.out.println("April is in the " + season + ".");  
    }  
}
```

# Control Flow and Enum Constants

```
}  
}
```

## Output is:

April is in the Spring.

## Switch

The switch statement is Java's multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative than a large series of if-else-if statements. Here is the general form of a switch statement:

A switch works with the **byte, short, char, and int primitive data types**. It also works with **enumerated types** (discussed in Enum Types), the **String** class, and a few special classes that wrap certain primitive types: **Character, Byte, Short, and Integer**.

```
switch(expression){  
    case value1:  
        //statement sequence  
        break;  
    case value2:  
        //statement sequence  
        break;  
    .  
    .  
    .  
    default:  
        //default statement sequence  
}
```

# Control Flow and Enum Constants

## Iteration Statements

Java's iteration statements are for, while, and do-while. These statements create what we commonly call loops. As you probably know, a loop repeatedly executes the same set of instructions until a termination condition is met.

### **while**

The while loop is Java's most fundamental loop statement. It repeats a statement or block while its controlling expression is true. Here is its general form:

```
while(condition) {  
    // body of loop  
}
```

The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true.

### **do-while**

sometimes it is desirable to execute the body of a loop at least once, even if the conditional expression is false to begin with. Java supplies a loop that does just that: the do-while. The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is

```
do {  
    // body of loop  
} while (condition);
```

# Control Flow and Enum Constants

**for loop:**

```
for(initialization; condition; iteration) {  
    // body  
}
```

If only one statement is being repeated, there is no need for the curly braces.

```
int n;  
for(n=10; n>0; n--){  
    System.out.println("tick " + n);  
}
```

## Declaring Loop Control Variables Inside the for Loop

```
// here, n is declared inside of the for loop  
for(int n=10; n>0; n--)  
    System.out.println("tick " + n);  
}
```

When you declare a variable inside a for loop, there is one important point to remember: the scope of that variable ends when the for statement does.

## Using the Comma

There will be times when you will want to include more than one statement in the initialization and iteration portions of the for loop.

```
public class WhileComma {  
    public static void main(String[] args){  
        int a, b;  
        for(a=1,b=4;a<b;a++,b--){  
            System.out.println("a="+a);  
            System.out.println("b="+b);  
        }  
    }  
}
```

# Control Flow and Enum Constants

```
}  
}
```

In this example, the initialization portion sets the values of both a and b. The two comma-separated statements in the iteration portion are executed each time the loop repeats.

## Infinite Loops

You can intentionally create an infinite loop (a loop that never terminates) if you leave all three parts of the for empty. For example:

```
for(;;) {  
    // ...  
}
```

## The For-Each Version of the for Loop

Beginning with JDK 5, a second form of for was defined that implements a “for-each” style loop.

**for(type itr-var: collection) statement-block**

## Statements That Break Control Flow

### Jump Statements

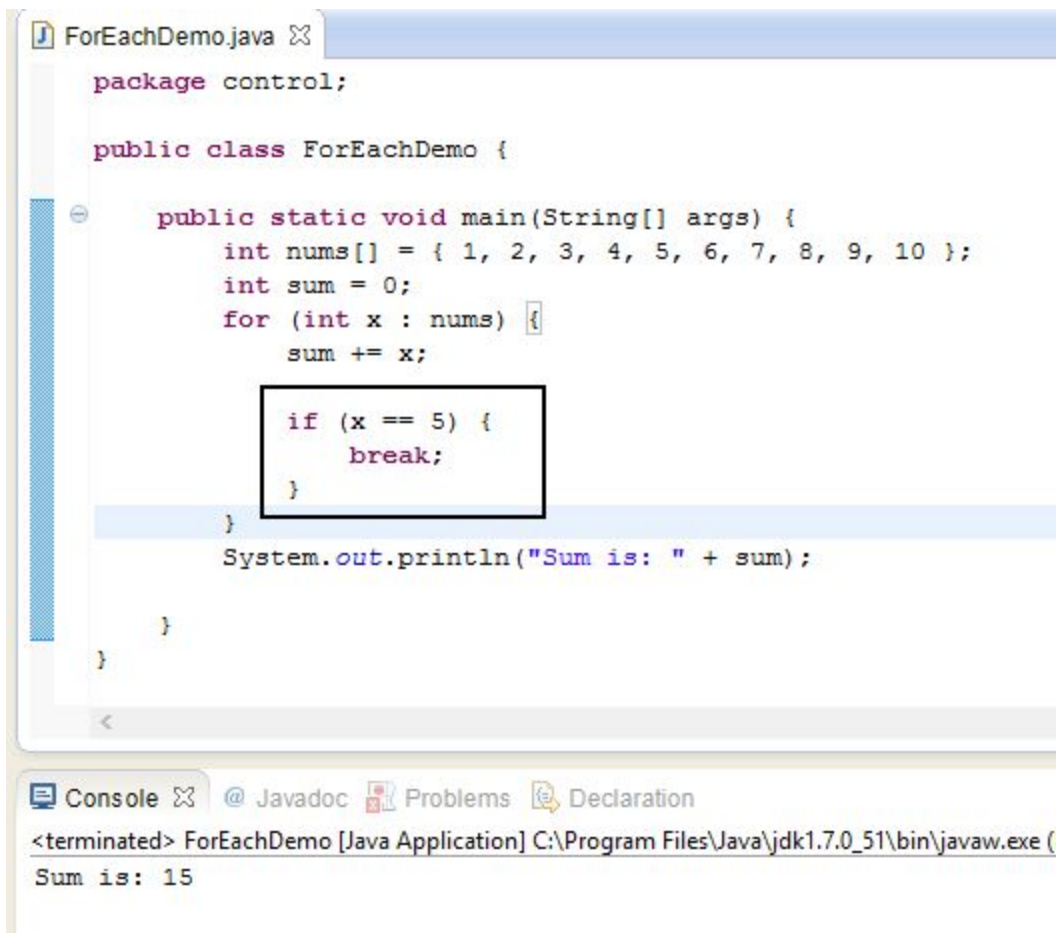
Java supports three jump statements: **break**, **continue**, and **return**. These statements transfer control to another part of your program.

### Using break

In Java, the break statement has three uses. First, as you have seen, it terminates a statement sequence in a switch statement. Second, it can be used to exit a loop. Third, it can be used as a “civilized” form of goto.

# Control Flow and Enum Constants

## Using break to Exit a Loop



```
ForEachDemo.java
package control;

public class ForEachDemo {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int sum = 0;
        for (int x : nums) {
            if (x == 5) {
                break;
            }
            sum += x;
        }
        System.out.println("Sum is: " + sum);
    }
}
```

Console

<terminated> ForEachDemo [Java Application] C:\Program Files\Java\jdk1.7.0\_51\bin\javaw.exe (

Sum is: 15

## Using break as a Form of Goto

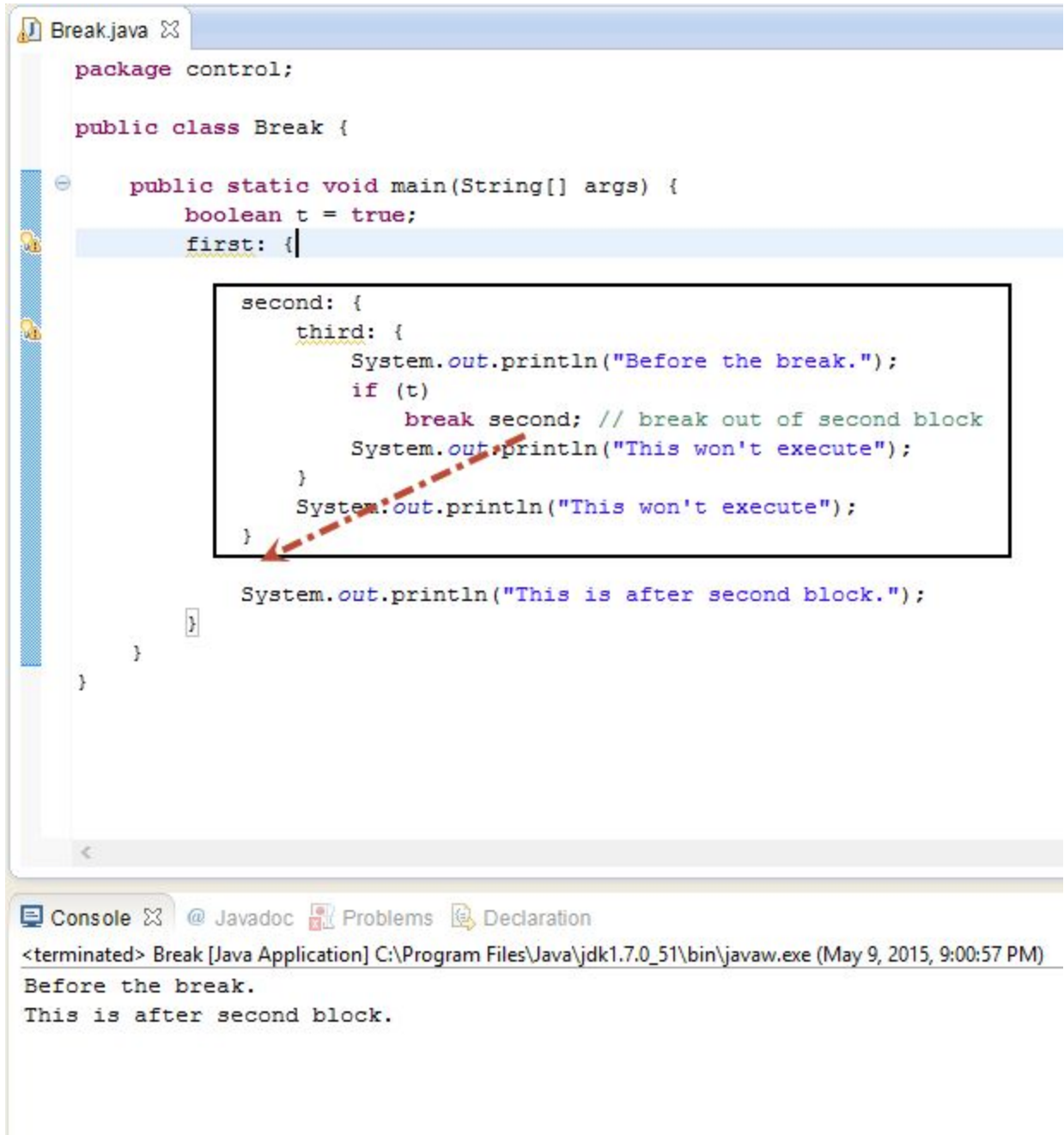
Using break as a civilized form of goto.

The general form of the labeled break statement is shown here:

break label;



# Control Flow and Enum Constants



The screenshot shows an IDE with a file named `Break.java` open. The code defines a `package control;` and a `public class Break` with a `main` method. Inside the `main` method, a `boolean t = true;` is declared. A label `first:` is placed before an opening curly brace. Inside this block, there is a `second:` block containing a `third:` block. The `third:` block contains a `System.out.println("Before the break.");` followed by an `if (t)` statement. Inside the `if` block, there is a `break second;` statement with a comment `// break out of second block`, followed by a `System.out.println("This won't execute");`. After the `third:` block, there is another `System.out.println("This won't execute");`. After the `second:` block, there is a `System.out.println("This is after second block.");`. A red dashed arrow points from the `break second;` statement to the closing curly brace of the `second:` block, indicating the jump. The console output shows the execution results: `<terminated> Break [Java Application] C:\Program Files\Java\jdk1.7.0_51\bin\javaw.exe (May 9, 2015, 9:00:57 PM)`, `Before the break.`, and `This is after second block.`

```
package control;

public class Break {

    public static void main(String[] args) {
        boolean t = true;
        first: {

            second: {
                third: {
                    System.out.println("Before the break.");
                    if (t)
                        break second; // break out of second block
                    System.out.println("This won't execute");
                }
                System.out.println("This won't execute");
            }

            System.out.println("This is after second block.");
        }
    }
}
```

Console Output:

```
<terminated> Break [Java Application] C:\Program Files\Java\jdk1.7.0_51\bin\javaw.exe (May 9, 2015, 9:00:57 PM)
Before the break.
This is after second block.
```

// Using break to exit from nested loops

```
public class BreakLoop4 {
    public static void main(String[] args) {
        outer: for (int i = 0; i < 3; i++) {
```

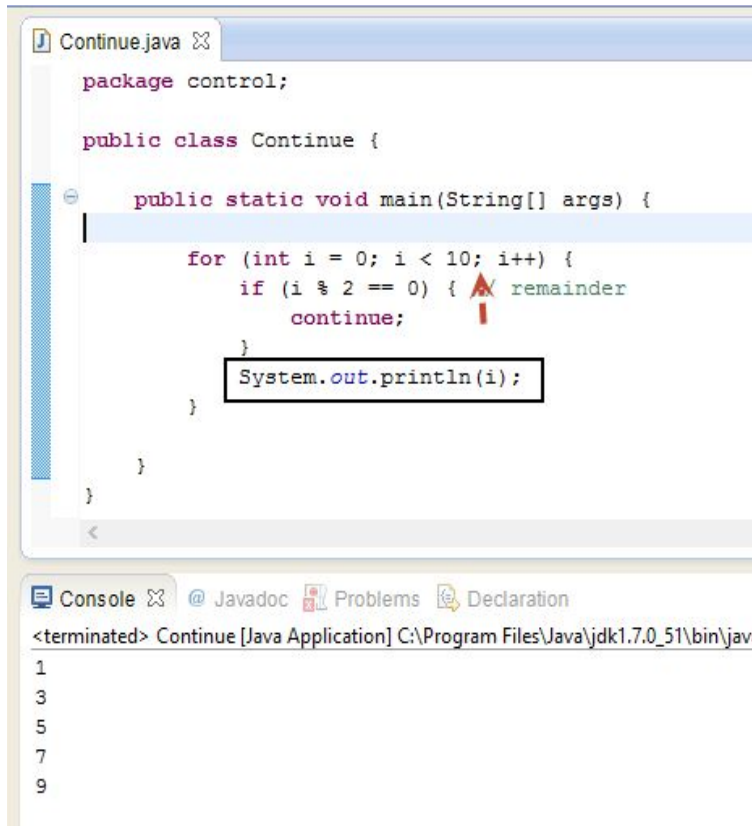
## Control Flow and Enum Constants

```
System.out.print("Pass " + i + ": ");
    for (int j = 0; j < 100; j++) {
        if (j == 10)
            break outer; // exit both loops
        System.out.print(j + " ");
    }
    System.out.println("This will not print");
}
System.out.println("Loops complete.");
}
```

**This program generates the following output:**

Pass 0: 0 1 2 3 4 5 6 7 8 9 Loops complete.

# Control Flow and Enum Constants



```
Continue.java
package control;

public class Continue {

    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i % 2 == 0) {
                remainder
                continue;
            }
            System.out.println(i);
        }
    }
}
```

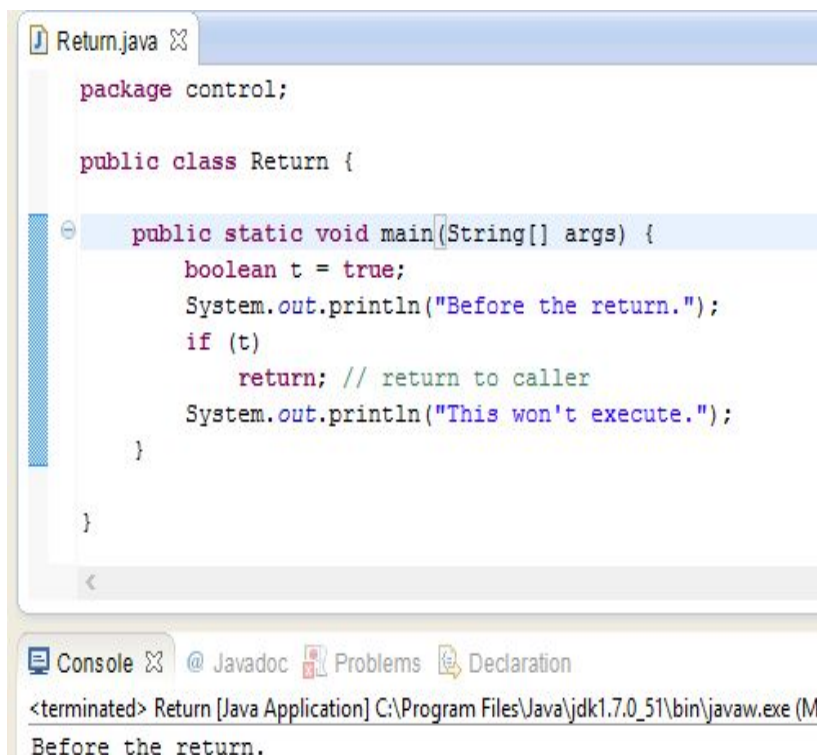
Console

<terminated> Continue [Java Application] C:\Program Files\Java\jdk1.7.0\_51\bin\java

1  
3  
5  
7  
9

## continue:

Sometimes it is useful to force an early iteration of a loop.



```
Return.java
package control;

public class Return {

    public static void main(String[] args) {
        boolean t = true;
        System.out.println("Before the return.");
        if (t)
            return; // return to caller
        System.out.println("This won't execute.");
    }
}
```

Console

<terminated> Return [Java Application] C:\Program Files\Java\jdk1.7.0\_51\bin\javaw.exe (M

Before the return.

## return:

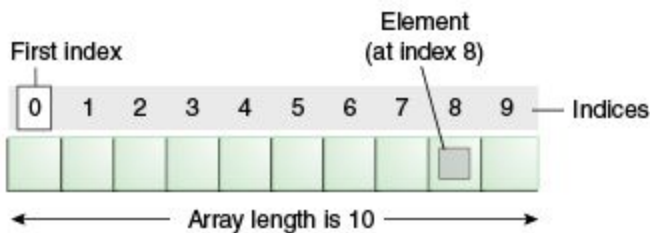
The last control statement is return. The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method.

# Control Flow and Enum Constants

## Java Array:

Normally, array is a collection of similar type of elements that have contiguous memory location.

Java array is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. Array in Java is index based, first element of the array is stored at 0 index.



## Advantage of Java Array

**Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.

**Random access:** We can get any data located at any index position.

## Disadvantage of Java Array

**Size Limit:** We can store only a fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java.

## Types of Array in Java

There are two types of array.

- **Single Dimensional Array**
- **Multidimensional Array**

## Single Dimensional Array in Java

Syntax to Declare an Array in Java

**dataType[] arr; (or)**

**dataType []arr; (or)**

# Control Flow and Enum Constants

**dataType arr[];**

Instantiation of an Array in java

**arrayRefVar=new datatype[size];**

Example of single dimensional java array

Let's see a simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
class Testarray{  
    public static void main(String args[]){  
        int a[]=new int[5];//declaration and instantiation  
        a[0]=10;//initialization  
        a[1]=20;  
        a[2]=70;  
        a[3]=40;  
        a[4]=50;  
        //printing array  
        for(int i=0;i<a.length;i++)//length is the property of array  
            System.out.println(a[i]);  
    }  
}
```

**Output:**

**10  
20  
70  
40  
50**

# Control Flow and Enum Constants

## Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

## Passing Array to method in java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see a simple example to get minimum number of an array using method.

```
class Testarray2{
    static void min(int arr[]){
        int min=arr[0];
        for(int i=1;i<arr.length;i++)
            if(min>arr[i])
                min=arr[i];
        System.out.println(min);
    }
    public static void main(String args[]){
        int a[]={33,3,4,5};
        min(a);//passing array to method
    }
}
```

**Output:**

**3**

# Control Flow and Enum Constants

## Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

### Syntax to Declare Multidimensional Array in java

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

### Example to instantiate Multidimensional Array in java

```
int[][] arr=new int[3][3]; //3 rows and 3 columns
```

### Example to initialize Multidimensional Array in java

```
arr[0][0]=1;
```

```
arr[0][1]=2;
```

```
arr[0][2]=3;
```

```
arr[1][0]=4;
```

```
arr[1][1]=5;
```

```
arr[1][2]=6;
```

```
arr[2][0]=7;
```

```
arr[2][1]=8;
```

```
arr[2][2]=9;
```

### Example of Multidimensional java array

Let's look at the simple example to declare, instantiate, initialize and print the 2Dimensional array.

# Control Flow and Enum Constants

```
class Testarray3{
    public static void main(String args[]){
        //declaring and initializing 2D array
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        //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:

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

## What is class name of java array?

In java, array is an object. For array object, a proxy class is created whose name can be obtained by `getClass().getName()` method on the object.

## Copying a java array

We can copy an array to another by the `arraycopy` method of `System` class.

```
System.arraycopy(copyFrom, 2, copyTo, 0, 7);
```



# Control Flow and Enum Constants

## Arrays: (Array Sorting)

```
int[] a = new int[10000];  
Arrays.sort(a);
```

## Need for the Java-Arrays Class:

There are often times when loops are used to do some tasks on an array like:

- Fill an array with a particular value.
- Sort an Arrays.
- Search in an Arrays.
- And many more

Arrays class provides several static methods that can be used to perform these tasks directly without the use of loops.

## Some Methods in Java Array:

The Arrays class of the java.util package contains several static methods that can be used to fill, sort, search, etc in arrays. These are:

**toString(originalArray):** This method returns a String representation of the contents of this Arrays. The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters a comma followed by a space. Elements are converted to strings as by String.valueOf() function.

**static <T> List<T> asList(T... a):** This method returns a fixed-size list backed by the specified Arrays.

**static int binarySearch(elementToBeSearched):** These methods searches for the specified element in the array with the help of Binary Search algorithm and return index of array otherwise negative value.

**compare(array1, array2):** This method compares two arrays passed as parameters lexicographically.

# Control Flow and Enum Constants

**copyOf(originalArray, newLength):** This method copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length

**copyOfRange(originalArray, fromIndex, endIndex):** This method copies the specified range of the specified array into a new Arrays.

**fill(originalArray, fillValue):** This method assigns this fillValue to each index of this Arrays.

**sort(originalArray):** This method sorts the complete array in ascending order.

**sort(originalArray, fromIndex, endIndex):** This method sorts the specified range of array in ascending order.

# Control Flow and Enum Constants

## Enum Types and its application:

Enumerations serve the purpose of representing a group of named constants in a programming language.

Enums are used when we know all possible values at compile time, such as choices on a menu, rounding modes, command line flags, etc. It is not necessary that the set of constants in an enum type stay fixed for all time. In Java, we can also add variables, methods and constructors to it. The main objective of enum is to define our own data types(Enumerated Data Types).

### Declaration of enum in java :

Enum declaration can be done outside a Class or inside a Class but not inside a Method.

```
enum Color {  
    RED, GREEN, BLUE;  
}  
  
public class Test {  
    // Driver method  
    public static void main(String[] args) {  
        Color c1 = Color.RED;  
        System.out.println(c1);  
    }  
}
```

// enum declaration inside a class.

```
public class Test {  
    enum Color {
```

# Control Flow and Enum Constants

```
    RED, GREEN, BLUE;
}
// Driver method
public static void main(String[] args) {
    Color c1 = Color.RED;
    System.out.println(c1);
}
}
```

- First line inside enum should be list of constants and then other things like methods, variables and constructor.
- According to Java naming conventions, it is recommended that we name constant with all capital letters

## Important points of enum :

- Every enum internally implemented by using Class
- Every enum constant represents an object of type enum.
- enum type can be passed as an argument to switch statement.
- Every enum constant is always implicitly public static final. Since it is static, we can access it by using enum Name. Since it is final, we can't create child enums.
- We can declare main() method inside enum. Hence we can invoke enum directly from the Command Prompt.

## Enum and Inheritance :

- All enums implicitly extend java.lang.Enum class. As a class can only extend one parent in Java, so an enum cannot extend anything else.
- toString() method is overridden in java.lang.Enum class, which returns enum constant name.
- enum can implement many interfaces.

# Control Flow and Enum Constants

## **values(), ordinal() and valueOf() methods :**

These methods are present inside java.lang.Enum.

- values() method can be used to return all values present inside enum.
- Order is important in enums. By using ordinal() method, each enum constant index can be found, just like array index.
- valueOf() method returns the enum constant of the specified string value, if exists.

## **enum and constructor :**

- enum can contain constructor and it is executed separately for each enum constant at the time of enum class loading.
- We can't create enum objects explicitly and hence we can't invoke enum constructor directly.