**Methods With Variable Argument Lists:**

**void dostuff(int …x)**

**{**

**}**

Remember, the three dots

Expects from 0 to as many ints as parameters.

**void dostufff2(char c,int …x)**

**{**

**}**

This expects a first char, then 0 to many ints.

This syntax tells the compiler that fun( ) can be called with zero or more arguments. As a result, here a is implicitly declared as an array of type int[]. Below is a code snippet for illustrating the above concept :

**An Example:**

// Java program to demonstrate varargs

class Test1

{

// A method that takes variable number of intger

// arguments.

static void fun(int ...a)

{

System.out.println("Number of arguments: " + a.length);

// using for each loop to display contents of a

for (int i: a)

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

System.out.println();

}

// Driver code

public static void main(String args[])

{

// Calling the varargs method with different number

// of parameters

fun(100); // one parameter

fun(1, 2, 3, 4); // four parameters

fun(); // no parameter

}

}

**Note:**

1. The … syntax tells the compiler that varargs has been used and these arguments should be stored in the array referred to by a.
2. The variable a is operated on as an array. In this case, we have defined the data type of a as int. So it can take only integer values. The number of arguments can be found out using a.length, the way we find the length of an array in Java.

**Another example:**

// Java program to demonstrate varargs with normal

// arguments

class Test2

{

// Takes string as a argument followed by varargs

static void fun2(String str, int ...a)

{

System.out.println("String: " + str);

System.out.println("Number of arguments is: "+ a.length);

// using for each loop to display contents of a

for (int i: a)

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

System.out.println();

}

public static void main(String args[])

{

// Calling fun2() with different parameter

fun2("GeeksforGeeks", 100, 200);

fun2("CSPortal", 1, 2, 3, 4, 5);

fun2("forGeeks");

}

}

**Some Rules:**

1. Remember, there cannot be two varargs which belong to the same function.

**void method(String... gfg, int... q)**

**{**

**// Compile time error as there are two**

**// varargs**

**}**

1. Also, varargs must be the last parameter of a functions, if there is more than one parameter to the function.

**void method(int... gfg, String q)**

**{**

**// Compile time error as vararg appear**

**// before normal argument**

**}**

1. Vararg Methods can also be overloaded but overloading may lead to ambiguity.