We say that two method signatures are matching if and only if they are identical or they differ only in the variables used for the formal parameters. In other words, matching method signatures involve the same method name and formal parameters with the same data types in the same order. For example,

check( int n, String s ) and check( int s, String n )

are matching signatures, but

check( int n, String s ) and check( String s, int n )

are not. Nor are

check( int n, String s ) and czech( int n, String s ).

When a method's definition includes a call to a method with a matching signature, the method is said to be recursive. In the program below, the static method mystery is recursive. Its signature is mystery( int n ) and its definition includes the statement:

mystery( n - 1 );

which, since n - 1 is an int, is a call to a method with a signature that matches that of mystery itself. Execute this program by clicking the **Run** button, and you will discover that it outputs an appropriate message each time it is called.

public static void mystery( int n )

{

  System.out.println( "mystery called with n = " + n );

  if ( n == 0 )

  {

    System.out.println( "n is zero so no more calls!" );

    return;

  }

  System.out.println( "Let's do it again ..." );

  mystery( n - 1 );

}

public static void main( String[] args )

{

  mystery( 5 );

}

mystery called with n = 5   
Let's do it again ...   
mystery called with n = 4   
Let's do it again ...   
mystery called with n = 3   
Let's do it again ...   
mystery called with n = 2   
Let's do it again ...   
mystery called with n = 1   
Let's do it again ...   
mystery called with n = 0   
n is zero so no more calls!

The expression mystery( n - 1 ) is called the recursive call. Notice that the argument to mystery in the recursive call is one less than the current value of n. Notice also that when n is zero, the recursive call is not reached. Instead, the empty return statement causes execution of the method to stop. The boolean expression n == 0 — which tests for this "exit now" state of affairs — is called the terminating condition.

If the argument of the recursive call did not get smaller each time, or if we had omitted the terminating condition, Java would find itself in an infinite loop. You may test this by changing the recursive call to mystery( n ), say, or by removing the part of the program that includes the terminating condition.

If you do either of these things here, the Java compiler that is embedded in this web page will "timeout", that is, after a few seconds it will give up trying to evaluate the expression mystery( 5 ). Typically, other Java compilers will go on trying to evaluate the expression until they run out of resources or until you forcibly interrupt the program.

The following program has been obtained by making a small change to the above definition of mystery. We have moved the last call to System.out.println to after the recursive call. Try to predict what the output will be from this new program, then run the program to see if you are correct. Try to explain why the messages appear in the output window in the order that they do.

public static void mystery( int n )

{

  System.out.println( "mystery called with n = " + n );

  if ( n == 0 )

  {

    System.out.println( "n is zero so no more recursive calls!" );

    return;

  }

  mystery( n - 1 );

  System.out.println( "We did it again with n = " + n );

}

public static void main( String[] args )

{

  mystery( 5 );

}

mystery called with n = 5   
mystery called with n = 4   
mystery called with n = 3   
mystery called with n = 2   
mystery called with n = 1   
mystery called with n = 0   
n is zero so no more recursive calls!   
We did it again with n = 1   
We did it again with n = 2   
We did it again with n = 3   
We did it again with n = 4   
We did it again with n = 5

Top of Form

**Exercise 87**

The messages that are displayed are as follows:

mystery called with n = 5  
mystery called with n = 4  
mystery called with n = 3  
mystery called with n = 2  
mystery called with n = 1  
mystery called with n = 0  
n is zero so no more recursive calls!  
We did it again with n = 1  
We did it again with n = 2  
We did it again with n = 3  
We did it again with n = 4  
We did it again with n = 5

Each call to mystery involves a recursive call to mystery before the final statement of the current call is executed. So the messages printed upon entry to an application of the method appear first, one after the other in an order that shows that the argument is decreasing in value by 1 from one call to the next. When the argument is 0, the terminating condition is satisfied. So the message about n being zero is printed, and the empty return statement returns control to the point of the method immediately after the most recent application of mystery. This was mystery( n - 1 ) when n was 1. So the message printed upon exit from that application appears next. This completes that call to mystery, which was mystery( n - 1 ) when n was 2. So the message printed upon exit from that application appears next. And so on until the message printed upon exit from the initial application, mystery( 5 ), finally appears in the output window.

Bottom of Form