**What Is a Native Method?**

Simply put, a native method is the Java interface to non-Java code. It is Java's link to the "outside world." More specifically, a native method is a Java method whose implementation is provided by non-Java code, most likely c or cpp.

*A native method is a Java method whose implementation is provided by non-java code.*

In Java, this is done via native methods. In your Java class, you mark the methods you wish to implement outside of Java with the native method modifier-much like you would use the public or static modifiers. Then, rather than supplying the method's body, you simply place a semicolon in its place. As an example, the following class defines a variety of native methods:

public class IHaveNatives  
{  
  native public void Native1( int x ) ;  
  native static public long Native2() ;  
  native synchronized private float Native3( Object o ) ;  
  native void Native4( int[] ary ) throws Exception ;  
}

This sample class shows a number of possible native methods. As you may have noticed, native methods look much like any other Java method, except a single semicolon is in the place of the method body. Naturally, the body of the method is implemented outside of Java. What you basically define is the interface to this *external* method. This method declaration describes the Java view of some foreign code.

The only thing special about this declaration is that the keyword native is used as a modifier. Every other Java method modifier can be used along with native, except abstract. This is logical, because the native modifier implies that an implementation exists, and the abstract modifier insists that there is no implementation. Your native methods can be static methods, thus not requiring the creation of an object (or instance of a class). This is often convenient when using native methods to access an existing C-based library. Naturally, native methods can limit their visibility with the public, private, private protected, protected, or unspecified *default* access. Native methods can also be synchronized. In the case of a synchronized native method, the Java VM will perform the monitor locking prior to entering the native method implementation code. So, as in Java, the developer is not burdened with doing the actual monitor locking and unlocking.

A native method can accept and return any of the Java types-including class types. Of course, the power of exception handling is also available to native methods. The implementation of the native method can create and throw exceptions similar to a Java method.

The presence of native methods does not affect how other classes call those methods. The caller does not even realize it is calling a native method, so no special code is generated, and the calling convention is the same as for any other method. The Java virtual machine will handle all the details to make the call in the native method implementation. When a class containing native methods is subclassed, the subclass will inherit the native method and also will have the capability of overriding the native method-even with a Java method (that is, the overridden method can be implemented in Java). If a native method is also marked with the final modifier, a subclass is still prevented from overriding it.

Native methods are very powerful, because they effectively extend the Java virtual machine. In fact, your Java code already uses native methods. In the current implementation from Sun, native methods are used in many places to interface to the underlying operating system. This enables a Java program to go beyond the confines of the Java Runtime. With native methods, a Java program can virtually do any application level task.