JAVAWORLD

JAVA TIPS
By John D. Mitchell

About |
The continuation of JavaWorld's most popular long-running series, featuring new and classic Java programming tips from your peers in the Java developer community.

HOW-TO

Java Tip 10: Implement callback routines in Java

Using interfaces to implement the equivalent of callback functions in Java

JavaWorld | Jun 1, 1996 1:00 AM PT

Developers conversant in the event-driven programming model of MS-Windows and the X Window System are accustomed to passing function pointers that are invoked (that is, "called back") when something happens. Java's object-oriented model does not currently support method pointers, and thus seems to preclude using this comfortable mechanism. But all is not lost!

Java's support of interfaces provides a mechanism by which we can get the equivalent of callbacks. The trick is to define a simple interface that declares the method we wish to be invoked.

For example, suppose we want to be notified when an event happens. We can define an interface:

```
public interface InterestingEvent
{
    // This is just a regular method so it can return something or
    // take arguments if you like.
    public void interestingEvent ();
}
```

This gives us a grip on any objects of classes that implement the interface. So, we need not concern ourselves with any other extraneous type information. This is much nicer than hacking trampoline C functions that use the data field of widgets to hold an object pointer when using C++ code with Motif.

The class that will signal the event needs to expect objects that implement the **InterestingEvent** interface and then invoke the **interestingEvent()** method as appropriate.

```
public class EventNotifier
{
    private InterestingEvent ie;
    private boolean somethingHappened;
    public EventNotifier (InterestingEvent event)
    {
    // Save the event object for later use.
    ie = event;
    // Nothing to report yet.
    somethingHappened = false;
    }
    //...
    public void doWork ()
    {
    // Check the predicate, which is set elsewhere.
    if (somethingHappened)
        {
        // Signal the even by invoking the interface's method.
        ie.interestingEvent ();
        }
    //...
    }
    // ...
}
```

In that example, I used the **somethingHappened** predicate to track whether or not the event should be triggered. In many instances, the very fact that the method was called is enough to warrant signaling the **interestingEvent()**.

The code that wishes to receive the event notification must implement the **InterestingEvent** interface and just pass a reference to itself to the event notifier.

```
public class CallMe implements InterestingEvent
{
    private EventNotifier en;
    public CallMe ()
    {
    // Create the event notifier and pass ourself to it.
    en = new EventNotifier (this);
    }
    // Define the actual handler for the event.
    public void interestingEvent ()
    {
    // Wow!  Something really interesting must have occurred!
    // Do something...
    }
    //...
}
```

That's all there is to it. I hope use this simple Java idiom will make your transition to Java a bit less jittery.

Subsisting on caffeine, sugar, and too little sleep, John D. Mitchell has been consulting for most of the last nine years, and has developed PDA software in OO assembly language at Geoworks. He funds his Java addiction by writing compilers, Tcl/Tk, C++, and Java systems. He coauthored the hot new Java book Making Sense of Java and is currently developing a Java compiler.