Programación dirigida por eventos

Hasta ahora hemos visto programas orientados por lotes. Recordá que los programas pueden estar *manejados por lotes* (comienzan, hacen algo y finalizan) o *manejados por eventos* (comienzan, esperan algún tipo de *evento* y finalizan cuando se les da la orden de terminar por medio de un evento. ¿Cómo se crean los programas dirigidos por eventos? Veremos dos maneras de hacerlo: una primera en la que simularemos un entorno de eventos, y otra en la que crearemos una interfaz gráfica muy sencilla que utiliza el sistema operativo para generar eventos.

Simulando una bifurcación de eventos

Todo programa dirigido por eventos tiene en algún momento una bifurcación que toma los eventos y los procesa. Los eventos pueden ser generados por el sistema operativo (como ocurre con prácticamente todos los programas que contienen una interfaz gráfica) o el mismo programa puede funcionar buscando los eventos (como en los sistemas de control embebidos, etc.)

Crearemos un programa que espera precisamente un solo tipo de evento: ver si una tecla ha sido pulsada. Nuestro programa procesará los resultados del evento hasta que encuentre un nuevo evento de cierre que le ordenará detener la ejecución. En nuestro caso el evento de cierre será la tecla de la barra espaciadora. Procesaremos los eventos recibidos de una manera muy sencilla: simplemente imprimiremos en la pantalla el código ASCII de la tecla presionada. Usaremos BASIC para esto ya que tiene una función sencilla para leer la tecla que se ha presionado. Esta función es INKEY$.

Primero implementamos el cuerpo central del programa que simplemente pone en funcionamiento el bucle que recibirá los eventos y llama a las subrutinas de manejo de eventos cuando se detecta un evento válido.

```
' Declaramos las subrutinas que manejarán los eventos
DECLARE SUB eventoTecla (tecla AS STRING)
DECLARE SUB teclaSalir (tecla AS STRING)

' Primero borramos la pantalla y advertimos al usuario
' de cómo tiene que hacer para detener el programa
CLS
PRINT "Pulse ESPACIO para finalizar..."
PRINT

' Bucle eterno
WHILE 1
        ky$ = INKEY$
        largo = LEN(ky$)
        IF length <> 0 THEN
        ' enviamos el evento hacia las rutinas de manejo de eventos
            IF ky$ <> " " THEN
                CALL eventoTecla(ky$)
            ELSE
                CALL teclaSalir(ky$)
            END IF
        END IF
WEND
```

Fijate que lo que hacemos con los eventos no es importante para la parte principal del programa, la cual simplemente recibe los eventos y los pasa a los manejadores del evento. La independencia de la captura de eventos y el procesamiento de los eventos es una de las características más importantes de la programación dirigida por eventos.

Ahora es el momento de implementar nuestros dos manejadores de eventos. El primero, `eventoTecla` simplemente imprime el valor ASCII de la tecla pulsada:

```
SUB teclaEvento (tecla AS STRING)
    ' Imprime las teclas válidas
    largo = LEN(tecla)
    IF largo = 1 THEN ' es simple ASCII
       PRINT ASC(tecla)
    ELSE
       IF largo = 2 THEN
          'no es alfanumérica, así que utilizamos el segundo caracter
          PRINT ASC(MID$(tecla, 2, 1))
       END IF
    END IF
END SUB
```

El evento `teclaSalir` es muy sencillo: simplemente detiene la ejecución del programa con la instrucción `STOP`:

```
SUB teclaSalir (tecla AS STRING)
    STOP
END SUB
```

Si usaramos esto como base en muchos proyectos, probablemente deberíamos incluir una función de inicialización al comienzo y una de terminación al final. De esta manera, el programador puede usar las bifurcaciones y utilizar sus propias rutinas de inicialización y terminación. De esta manera, nuestro código base puede ser reutilizado para muchas tareas distintas.

Esto es exactamente lo que hacen muchos entornos gráficos, ya que la recepción de los eventos depende del sistema operativo, pero las aplicaciones deben proveer las funciones que manejan los resultados del evento. Dichas funciones deben relacionarse con un tipo de evento al comienzo del programa, y después, el sistema se encarga de procesar todo.

Veamos cómo funciona esto al explorar la biblioteca gráfica de Python, Tkinter:

Un programa con interfaz gráfica

Para este ejercicio usaremos el kit de Tkinter que acompaña a la distribución de Python. Tkinter es una interfaz que envuelve las funciones originales de Tk para ser utilizadas desde Python. También está disponible para Perl. La versión de Python es un entorno orientado a objetos que, en mi opinión, es considerablemente más sencillo que la versión original de Tk. No me ocuparé específicamente de los aspectos gráficos de esto ya que me interesa focalizar la técnica de programación al utilizar a Tkinter para recibir los eventos y dejando en manos del programador la creación de la interfaz gráfica inicial y el procesamiento de los eventos.

En el ejemplo creamos una clase `Teclas` que crea la interfaz gráfica del usuario mediante el método `__init__` y *vincula* la barra espaciadora con el método `eventoCerrar`. La clase también define el método `eventoCerrar` requerido.

La interfaz gráfica consiste simplemente en un cuadro para el ingreso de texto cuyo comportamiento por defecto es mostrar los caracteres correspondientes a las teclas que se presionan.

Crear una clase para la aplicación es muy común en los entornos de programación orientada a objetos, dado que los conceptos de enviar eventos a un programa y de enviar mensajes a un objeto son muy similares. Los dos conceptos se combinan muy bien de forma sencilla. Por esta razón, una función que maneje un evento se convierte en un método de la clase.

Una vez definida la clase, simplemente creamos una instancia de ella y le enviamos el mensaje `mainloop`.

Este sería nuestro código:

```
# Usamos "from X import *" para evitar tener que cualificar las funciones
# del tipo "tkinter.xxx"
from Tkinter import *

# Creamos la clase de aplicación que define la interfaz gráfica
# y los métodos que manejan los eventos
class Teclas(Frame):
    def __init__(self):
        Frame.__init__(self)
        self.txtBox = Text(self)
        self.txtBox.bind("", self.eventoCerrar)
        self.txtBox.pack()
        self.pack()

    def eventoCerrar(self,event):
        import sys
        sys.exit()


# Ahora creamos una instancia y comenzamos el bucle
miprograma = Teclas()
miprograma.mainloop()
```

En la versión con BASIC imprimíamos los códigos ASCII de todas las teclas y no la versión alfanumérica de las teclas como hicimos en Python. No hay nada que nos impida recibir todas las pulsaciones de teclas y hacer lo mismo. Para lograr esto debemos agregar la siguiente línea al método `__init__`:

```
self.txtBox.bind("< Key >", self.eventoTecla)
```

Y el siguiente método para procesar el evento:

```
def eventoTecla(self,event):
    str = "%d\n" % event.keycode
    self.txtBox.insert(END, str)
    return "break"
```

Nota 1: El valor de la tecla presionada se guarda en el campo `keycode` del evento. Para enterarme de esto tuve que revisar el código fuente de Tkinter.py... ¿recuerdas que la curiosidad es una de las principales características de un buen programador?

Nota 2: `return "break"` es una señal mágica que le indica a Tkinter que no invoque el procesamiento de eventos por defecto del cuadro gráfico. Sin esta línea, el cuadro de texto mostraría el código ASCII seguido del caracter tipeado (algo que nosotros no buscábamos).

Esto es suficiente por ahora. Esto no es un tutorial de Tkinter, por lo cual te recomiendo que busques en las páginas de referencia algún link a diversos tutoriales sobre Tkinter. También existen buenos libros acerca del uso de TK y Tkinter. Volveremos a encontrarnos con Tkinter en el "Caso de Estudio" en el que ilustro la manera de encapsular un programa orientado a lotes dentro de una interfaz gráfica para mejorar su utilización.

Anterior Próxima Contenido

Si tenés sugerencias o dudas podés enviar un email en inglés a: alan.gauld@btinternet.com o en español a: manilio@xoommail.com