Slides to Accompany *Programming Languages and Methodologies*

R. J. Schalkoff

Chapter 13, Part 1: Event-driven Programming (with GUI)

Event Driven Programming

Examples:

- Xwindows
- Windows (API)
- Windows (MFC)
- PalmOS
- OpenGL
- wxWidgets

A Long Time Ago...

A long time ago, in a galaxy far, far away...

When a program executed, the program code controlled what happened through the sequence of program statements..... The program might go so far as to occasionally prompt the user for input and print output, but the user had no or little control over the order in which events took place while the program was running.....

The (CLI) Typical Sequence of Events

In the 'old days' of command-line interface-based programs, the program followed a deterministic sequence of events:

`input (read)` → `compute` → `output (print)` → `halt`

- This paradigm is 'programmer driven control flow', since the *programmer* decides which part of the program will be run when and when.
- Input is solicited from the user under program control.

Event-Driven Programming- The Paradigm Shift

- *Instead of prompting for user input, it is the job of the program to listen for and respond to user actions (events) which may occur.*
- Typical events include mouse movements, engaging push buttons and sliders, menu selections, and data entry in prespecified fields on the GUI.

This basic programming change required in event-driven program leads to a number of conceptual changes on the part of the programmer. Specifically:

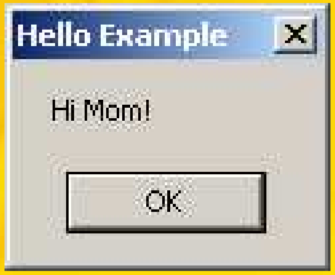
- Event-driven programs have an event loop that waits for and handles events.
- Program control is 'given up' to this loop. When run under a window(ing) system, event-driven systems rely on window(ing) software libraries and utilities.

Creating Event-Driven (and GUI) Software

1. We must determine what the graphical or visual user interface looks like, i.e., we must provide code which specifies the GUI on the chosen hardware display device.
2. We must anticipate all possible events. Otherwise, at best, the user may create inputs the program cannot 'see'.
3. We must associate each possible event with some code which reacts to or 'handles' the event. This mapping of events to handling code is often referred to as 'registering the handlers for events'.
4. We usually interact with the OS or libraries. In fact, the OS may serve as an intermediary between the event and our code handler.

5. Typically, we program to an API, which in most cases is a library (e.g., Xlib, Motif or OpenGL) or an OS (e.g., PalmOS, MS Windows).

Example: Microsoft Windows (API)



```
#include <windows.h>

   int WINAPI WinMain (HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       PSTR szCmdLine,
                       int iCmdShow)
   {
      MessageBox (NULL, "Hi Mom!", "Hello Example", MB_OK);
      return (0);
   }
```