Unidad I: Eventos

De acuerdo con Javasoft, las principales características de partida que han originado el nuevo modelo de manejo de eventos en el AWT, son:

- Que sea simple y fácil de aprender
- Que soporte una clara separación entre el código de la aplicación y el código del interfaz
- Que facilite la creación de robustos controladores de eventos, con menos posibilidad de generación de errores (chequeo más potente en tiempo de compilación)
- Suficientemente flexible para permitir el flujo y propagación de eventos
- Para herramientas visuales, permitir en tiempo de ejecución ver cómo se generan estos eventos y quien lo hace
- Que soporte compatibilidad binaria con el modelo anterior

1.1. Tipos de eventos

Los eventos ahora están organizados en jerarquías de clases de eventos.

El nuevo modelo hace uso de *fuentes* de eventos (**Source**) y *receptores* de eventos (**Listener**). Una fuente de eventos es un objeto que tiene la capacidad de detectar eventos y notificar a los receptores de eventos que se han producido esos eventos. Aunque el programador puede establecer el entorno en que se producen esas notificaciones, siempre hay un escenario por defecto.

Un objeto receptor de eventos es una clase (o una subclase de una clase) que implementa un interfaz receptor específico. Hay definidos un determinado número de interfaces receptores, donde cada interfaz declara los métodos adecuados al tratamiento de los eventos de su clase. Luego, hay un emparejamiento natural entre clases de eventos y definiciones de interfaces. Por ejemplo, hay una clase de eventos de ratón que incluye muchos de los eventos asociados con las

acciones del ratón, y hay un interfaz que se utiliza para definir los receptores de esos eventos.

Un objeto receptor puede estar registrado con un objeto fuente para ser notificado de la ocurrencia de todos los eventos de la clase para los que el objeto receptor está diseñado. Una vez que el objeto receptor está registrado para ser notificado de esos eventos, el suceso de un evento en esta clase automáticamente invocará al método sobreescrito del objeto receptor. El código en el método sobreescrito debe estar diseñado por el programador para realizar las acciones específicas que desee cuando suceda el evento.

Algunas clases de eventos, como los de ratón, involucran a un determinado conjunto de eventos diferentes. Una clase receptor que implemente el interfaz que recoja estos eventos debe sobreescribir todos los métodos declarados en el interfaz. Para prevenir esto, de forma que no sea tan tedioso y no haya que sobreescribir métodos que no se van a utilizar, se han definido un conjunto de clases intermedias, conocida como clases *Adaptadoras* (**Adapter**).

Estas clases Adaptadores implementan los interfaces receptor y sobreescriben todos los métodos del interfaz con métodos vacíos. Una clase receptor puede estar definida como clase que extiende una clase **Adapter** en lugar de una clase que implemente el interfaz. Cuando se hace esto, la clase receptor solamente necesita sobreescribir aquellos métodos que sean de interés para la aplicación, porque todos los otros métodos serán resueltos por la clase **Adapter**

Uno de los objetos receptor que se implementan con mayor frecuencia son los de la interfaz **WindowListener** en el manejo de ventanas, lo que haria necesario sobreescribir los seis métodos de la interfaz. Por lo que la otra clase receptor que se extiende es la clase **WindowAdapter** en vez de implementar la interfaz **WindowListener**. La clase **WindowAdapter** sobrescribe los seis métodos de la interfaz con métodos vacíos, por lo que la clase receptor no necesita sobreescribir esos seis métodos solo el que necesita.

1.2. Generación y propagación de eventos

El paquete java.awt.event es el que contiene la mayor parte de las clases e interfaces de eventos. El modelo de delegación de eventos es un concepto que trabaja de la siguiente manera:

Una fuente genera un evento y lo envía a uno a más oyentes o auditores, que han estado simplemente esperando hasta que reciben ese evento y una vez recibido lo procesan y lo devuelven.

Una fuente es un objeto que genera un evento. Esto ocurre cuando cambia de alguna manera el estado interno de ese objeto. Las fuentes pueden generar más de un tipo de eventos.

Una fuente tiene que ir acompañada de auditores para que estos reciban las notificaciones sobre el tipo específico de evento, cada tipo de evento tiene su propio método de registro. La forma general es:

Public void addTypeListener(TypeListener el)

Por ejemplo el método que registra o acompaña a un auditor de evento de teclado es addKeyListener( ). Cuando ocurre un evento, se notifica a todos los auditores registrados, que reciben una copia del objeto evento. Esto es lo que se conoce como multicasting del evento.

Una fuente también puede proporcionar un método que permita a un auditor eliminar un registro en un tipo específico de evento y la forma general es:

Public void removeTypeListener(TypeListener el);

Aquí Type es el nombre del evento y el es una referencia al auditor. Por ejemplo para borrar un auditor del teclado se llamaría removeKeyListener( ).

1.3. Métodos de control de eventos

Cuando una acción sobre un componente genera un evento, se espera que suceda algo, entendiendo por evento un mensaje que un objeto envía a algún otro objeto.

1.4. Creación de eventos

Evento	Se produce cuando
AutoSizeChanged	La propiedad **AutoSize** de un objeto cambia.
BackColorChanged	El color de fondo de un objeto cambia.
Click	Se hace clic sobre un objeto.
ContextMenuStrip-Changed	El valor de la propiedad **ContextMenuStrip** de un objeto cambia.
ControlAdded	Se añade un nuevo control a la colección **ControlCollection**.
ControlRemoved	Se elimina un control de la colección **ControlCollection**.
CursorChanged	El valor de la propiedad **Cursor** de un objeto cambia.
DoubleClick	Se hace doble clic sobre un objeto.
EnabledChanged	El valor de la propiedad **Enabled** de un objeto cambia.
FontChanged	El valor de la propiedad **Font** de un objeto cambia.
ForeColorChanged	El color del primer plano de un objeto cambia.
Load	Se inicia la carga de un formulario por primera vez.
Paint	El control se tiene que repintar.
Resize	El objeto es redimensionado.
SizeChanged	El valor de la propiedad **Size** cambia.
TextChanged	El valor de la propiedad **Text** de un objeto cambia.
Del foco	*(se exponen en el orden en el que se producen)*
Enter	Se entra en el control.
GotFocus	El control recibe el foco.
Leave	Se sale del control.
Validating	El control se está validando.
Validated	El control está validado.
LotFocus	El control pierde el foco.
Del teclado	*(se exponen en el orden en el que se producen)*
KeyDown	Se pulsa una tecla mientras el control tiene el foco.
KeyPress	Una tecla está pulsada mientras el control tiene el foco.
KeyUp	Una tecla es soltada mientras el control tiene el foco.
Del ratón	*(se exponen en el orden en el que se producen)*
MouseEnter	El puntero del ratón entra en un objeto.