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

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.

Gestión 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( ).

Auditores de eventos.

Un auditor es un objeto que es avisado cuando ocurre un evento. Tiene dos requisitos principales. Primero tiene que ser registrado o ir acompañado por una o más fuentes para recibir notificaciones sobre los tipos específicos de eventos. Segundo, tiene que implementar métodos para recibir y procesar notificaciones.

Clases de eventos principales en java.awt.event

Clase de evento	Descripción
ActionEvent	Se genera cuando se presiona un botón, se hace doble clic en un elemento de una lista, o se selecciona un elemento de tipo menú.
AdjustmentEvent	Se genera cuando se manipula un scrollbar.
ComponentEvent	Se genera cuando un componente se oculta, se mueve, se cambia de tamaño o se hace visible.

ContainerEvent	Se genera cuando se añade o se elimina un componente de un contenedor.
FocusEvent	Se genera cuando un componente gana o pierde el foco.
InputEvent	Superclase abstracta de cualquier clase de evento de entrada de componente.
ItemEvent	Se genera cuando se hace click en un checkbox o en un elemento de una lista; tambien ocurre cuando se hace una selección en una opción choice o cuando se selecciona o deselecciona un elemento de un menú de opciones.
KeyEvent	Se genera cuando se recibe una entrada desde el teclado.
MouseEvent	Se genera cuando el ratón se arrastra, se mueve, se hace clic, se presiona, o se libera; también se genera cuando el ratón entra o sale de un componente.
TextEvent	Se genera cuando se cambia el valor de un área de texto o un campo de texto
WindowEvent	Se genera cuando una ventana se activa, se cierra, se desactiva, se minimiza, se maximiza, se abre, o se sale de ella.