Unidad 3

Creación de componentes visuales

Desarrollo de interfaces

3

Índice

Introducción

En este tema estudiaremos diferentes conceptos que afectan a los componentes visuales de la interfaz.

Debemos tener en cuenta que los componentes son elementos prefabricados por lo que, aunque es posible crear más o modificar los existentes, trabajaremos casi exclusivamente con ellos, pero esto no es un impedimento, ya que cuentan con la flexibilidad suficiente para adaptarse a nuestras necesidades.

Podremos ver diferentes elementos de los componentes como las propiedades y los atributos, pero nos centraremos en el estudio del evento, ya que es el componente interactivo de la interfaz.

Presentaremos los diferentes eventos y sus características, así como la función de los *listeners* como elemento que permite que el evento tenga algún efecto.

Al finalizar esta unidad

- Conoceremos los diferentes elementos que dan forma a un componente visual.
- Estudiaremos los eventos, qué son y cómo funcionan.
- Describiremos los elementos necesarios para que los eventos funcionen.
- Definiremos algunas de las propiedades de los componentes.
- Distinguiremos los distintos programas de edición de imágenes y en qué momentos emplear cada uno.
- Sabremos la función del empaquetado.

3.1. Componentes visuales

Con el fin de que una interfaz sea accesible para todos los usuarios los componentes visuales son esenciales, ya que permite una mejor comunicación entre el usuario y la aplicación al facilitar el entendimiento por parte del usuario.

3.1.1. Concepto de componente

Consideramos componente a un módulo de código creado y empaquetado que puede implementarse en una aplicación y reutilizarse directamente, ya que trabajará en conjunto con el resto de componentes, permitiéndonos desarrollar una interfaz de manera rápida y sencilla con la simple inclusión de los componentes necesarios.

Otro requisito para considerarse componentes es que estos elementos ya estén probados, es decir, que hayan pasado un periodo de prueba que demuestre que funcionan correctamente junto con el resto de componentes.

3.1.2. Propiedades o atributos

Las propiedades definen las características de los objetos tanto en su comportamiento como en su apariencia, como puede ser su color, tipo de letra, etc. Los atributos almacenan valores internos para el uso de la clase u objeto mismo.

Existen dos tipos de métodos fundamentales para las propiedades y atributos:

- **Get:** permiten analizar los valores de las propiedades y atributos.
- **Set:** permite modificar los valores de las propiedades y atributos.

Las propiedades poseen ámbitos, en función de quién posea acceso a ellas, son tres en total:

- **Ámbito público:** permite su uso por cualquier elemento de la aplicación.
- **Ámbito privado:** tan solo permite acceso a sus propiedades a la clase donde se crea.
- **Ámbito estático:** se pueden emplear sin una instancia del objeto al que refieren.

Podemos encontrar dos tipos de propiedades:

- **Simples:** Representan un solo valor sin importar el tipo de este como `String`, `int`, `boolean`, etc.
- **Indexadas:** Empleando un array representa un conjunto de valores.

3.2. Eventos

Un evento es la recepción de un estímulo por parte de la aplicación, como puede ser el reconocimiento del clic en un botón, estos eventos permitirán a la aplicación iniciar una reacción que se asocie a ellos.

Podemos encontrar dos tipos de eventos en función de su origen:

- **Externos:** los producidos por el usuario.
- **Internos:** los que se programan y producen por la propia aplicación.

NOTA

Hasta ahora hemos visto la creación de diversos eventos, pero estos solo permiten una visualización estática, ya que directamente no se podría hacer nada con ellos, es con la inclusión de eventos cuando la interfaz pasa de ser un simple cuadro a un sistema interactivo de comunicación persona-ordenador.

Tipos de eventos	
Clase	**Descripción**
`EventObject`	Es el tipo principal desde el que se derivan el resto de eventos.
`MouseEvent`	Eventos producidos por la acción del ratón sobre los componentes.
`ComponentEvent`	Eventos de cambio de tamaño, posición, etc. de un componente.
`ContainerEvent`	Eventos producidos en los componentes container al añadir o eliminar componentes.
`WindowsEvent`	Evento producido cada vez que la ventana sufre un cambio, sin importar cual sea este.
`ActionEvent`	Evento realizado por la acción sobre un componente como la pulsación de un botón.

3.2.1. Componentes y eventos

Debido a la naturaleza interactiva de las interfaces es común que los componentes lleven asociados algún tipo de evento que permita interactuar con ellos. Los más empleados para ello son los siguientes:

Ejemplos de componentes con sus eventos asociados		
Clase	**Evento**	**Descripción de la acción destacada**
`JTextField`	`ActionEvent`	Pulsación de Enter.
`JButton`	`ActionEvent`	Pulsación del botón.
`JComboBox`	`ActionEvent`	Selección de un valor.
`ItemEvent`	`ItemEven`	
`JCheckBox`	`ActionEvent`	Marcado de una celda.
`ItemEvent`	`ItemEven`	
`JTextComponent`	`TextEvent`	Cambio de texto.
`JScrollBar`	`AdjustmentEvent`	Movimiento de la barra de desplazamiento.

3.2.2. Listeners

También llamados escuchadores, los *listeners* nos permiten detectar los eventos, ya que son los encargados de detectar el suceso de ese evento e iniciar la reacción asociada a él. Son imprescindibles para que el evento realmente cumpla su función.

Existen múltiples tipos de *listeners*, como son:

KeyListener

Detecta las pulsaciones de diferentes teclas y tipos. Generalmente se asocia con eventos ActionEvent.

Ejemplos de KeyListener	
Pulsación	**Acción**
`keyPressed`	Pulsación de una tecla.
`keyTyped`	Pulsar y soltar una tecla.
`keyReleased`	Soltar una tecla

ActionListener

Detecta la pulsación sobre componentes, ya sea mediante ratón o mediante la tecla Enter. General se asocia con eventos ActionEvent.

Ejemplos de ActionListener	
Componente	**Acción**
`JButton`	Clic sobre el botón o pulsar de Enter al señalizarlo.
`JTextField`	Pulsar Enter seleccionando el componente.
`JMenuItem`	Selección de una opción entre varias.
`JList`	Doble clic sobre el componente de una lista.

MouseListener

Detecta el clic del ratón sobre un componente y los distintos tipos de clic que se pueden realizar. Se asocian con eventos MouseEvent.

Ejemplos de MouseListener	
Pulsación	**Descripción**
`mouseClicked`	Pulsar y soltar con el ratón sobre un componente.
`mouseExited`	Utilizar el puntero para salir de un componente.
`mouseEntered`	Utilizar el puntero del ratón para acceder a un componente.
`mousePressed`	Presionar con el puntero sobre un componente.
`mouseReleased`	Soltar el puntero del ratón.

FocusListener

Evento que se produce al seleccionar o deseleccionar un componente. Se asocian con eventos FocusEvent.

MouseMotionListener

Se produce al detectar el movimiento del ratón.

Ejemplos de MouseMotionListener	
Modo	Descripción
`mouseMoved`	Mover el puntero sobre un componente.
`mouseDragged`	Hacer clic sobre un botón y arrastrar el componente.

3.2.3. Métodos y eventos

Cada uno de los eventos utilizará un método para su tratamiento. El método se ejecutará tras el Listener, siendo uno u otro método en función del evento.

Ejemplos de componentes con sus eventos asociados		
Listener	Método	
`ActionListener`	`public void actionPerformed(ActionEvent e)`	
`KeyListener`	`keyPressed`	`public void keyPressed(KeyEvent e)`
	`keyTyped`	`public void keyTyped(KeyEvent e)`
	`keyRelease`	`public void keyReleased(KeyEvent e)`
`FocusListener`	Obtención del foco	`public void focusGained(FocusEvent e)`
	Pérdida de foco	`public void lostGained(FocusEvent e)`
`MouseListener`	`mouseClicked`	`public void mouseClicked(MouseEvent e)`
	`mouseExited`	`public void mouseExited(MouseEvent e)`
	`mousePressed`	`public void mousePressed(MouseEvent e)`
	`mouseReleased`	`public void mouseReleased(MouseEvent e)`
	`mouseEntered`	`public void mouseEntered(MouseEvent e)`
`MouseMotionListener`	`mouseMoved`	`public void mouseMoved(MouseEvent e)`
	`mouseDragged`	`public void mouseDragged(MouseEvent e)`

3.3. Introspección y reflexión

A pesar de que Java es un lenguaje estático, no permite modificaciones en sus objetos durante la ejecución, se pueden introducir ciertas modificaciones de funcionalidades mediante la reflexión y la introspección.

- La **reflexión.** Permite modificar y recuperar datos relacionados con la estructura de un objeto como son, entre otros:
  - Métodos y propiedades de clase
  - Constructores.
  - Interfaces.
  - Nombre.
- **Introspección.** Permite utilizar de forma dinámica métodos, propiedades y eventos de un componente al arrastrar y soltar el elemento.

3.4. Persistencia del componente

La persistencia es una característica requerida para el desarrollo del componente, ya que permite mantener el estado de una clase invariable a través de la serialización. Al emplear la serialización todo nuevo componente debe incluir, según su tipo, una de estas interfaces.

- **Serialización automática:** java.io.Serializable.
- **Serialización programada:** java.io.Externalizable.

3.5. Herramientas para el desarrollo de componentes visuales

Una interfaz gráfica personalizada es probable que requiera de imágenes propias, por lo que el empleo de herramientas de edición de imágenes llega a ser fundamental.

La elección de la herramienta de edición de imágenes es complicada, ya que la elección ideal no siempre es la mejor, debemos encontrar un equilibrio entre las necesidades de la imagen y las cualidades de los editores de imágenes. Algunos de los editores que podemos emplear son:

- **Gimp.** De carácter gratuito, permite el retoque de fotografías y la composición y creación de imágenes. Es un programa relativamente simple con resultados de gran calidad.
- **Microsoft Paint.** Es el más sencillo de emplear, muy recomendado para dibujos rápidos donde la calidad no importe mucho, permite la creación desde cero de imágenes. Las últimas versiones incluyen elementos 3D.
- **Adobe Photoshop.** Uno de los más extendidos y populares, nos ofrece una gran calidad, aunque su uso no se considera muy sencillo. Destaca por su funcionalidad de trabajo en capas. Destaca por las siguientes características:
  - Adecuado para el entorno profesional.
  - Permite la creación desde cero de una imagen.
  - Empleo del sistema de capas.
  - Compatible con múltiples formatos.
  - Inclusión de múltiples sistemas como filtros, efectos, retoques, etc.

3.6. Empaquetado de componentes

Debemos tener en cuenta que una aplicación suele conformarse de diferentes elementos, en ocasiones en diferentes formatos, librerías, ficheros ejecutables, elementos multimedia, etc., por lo que para llevar a cabo una buena distribución es recomendable el empleo de paquetes para su transporte. Estos paquetes permiten contener los múltiples componentes que conforman una aplicación, de modo que se puedan transportar como un único archivo, facilitando así su envío y evitando problemas por la pérdida de algún archivo.

www.universae.com