

UI - conceptos

Autor: Carlos Lombardi

Fecha: 18/05/2017

Estas notas intentan reunir varias ideas que aparecieron en las clases sobre Arena. De ninguna manera pretenden reemplazar a las clases, son apuntes que (espero) ayuden a recordar qué se dijo, y a entender el código que hay que leer, y el que hay que escribir.

Idea inicial

Para resolver la interfaz de usuario, usamos lo mismo que venimos trabajando hasta ahora: objetos. Vamos a tener objetos que van a representar los elementos que forman la interfaz. Estos objetos se van a relacionar con los que vinimos desarrollando hasta ahora. Ahora vamos a tener p.ej. objetos que representan pedidos, productos y renglones, y también objetos que representan cada elemento de la ventana que muestra el detalle de un pedido. Los objetos que forman la pantalla tienen que relacionarse de alguna forma con los que modelan al pedido, para saber qué mostrar.

Modelo y vista

El modelo está formado por los objetos que describen un dominio, p.ej. el de los pedidos de mercadería. Tenemos objetos que representan pedidos, renglones, productos.

La vista son otros objetos, que describen cómo presentar visualmente el dominio. En este caso, cómo mostrar un pedido.

Objetos que forman parte de la vista

Estos son los objetos principales que se usan en Arena para definir una vista:

- ventanas,
- paneles,
- controles,
- layouts.

Los conceptos de ventana, panel, control y layout no son exclusivos de Arena, van a aparecer en cualquier tecnología de UI que usemos¹.

Ventana, control, panel

Un objeto **ventana** representa ... una ventana. Maneja los botones de minimizar, maximizar y cerrar, tiene un título, y un espacio.

Un objeto **control** representa cada elemento que se coloca en una ventana. Distintos tipos de controles son: despliegue de un dato simple (que muestra el dato pero no permite editarlo), campo editable, botón, tabla, lista de selección.

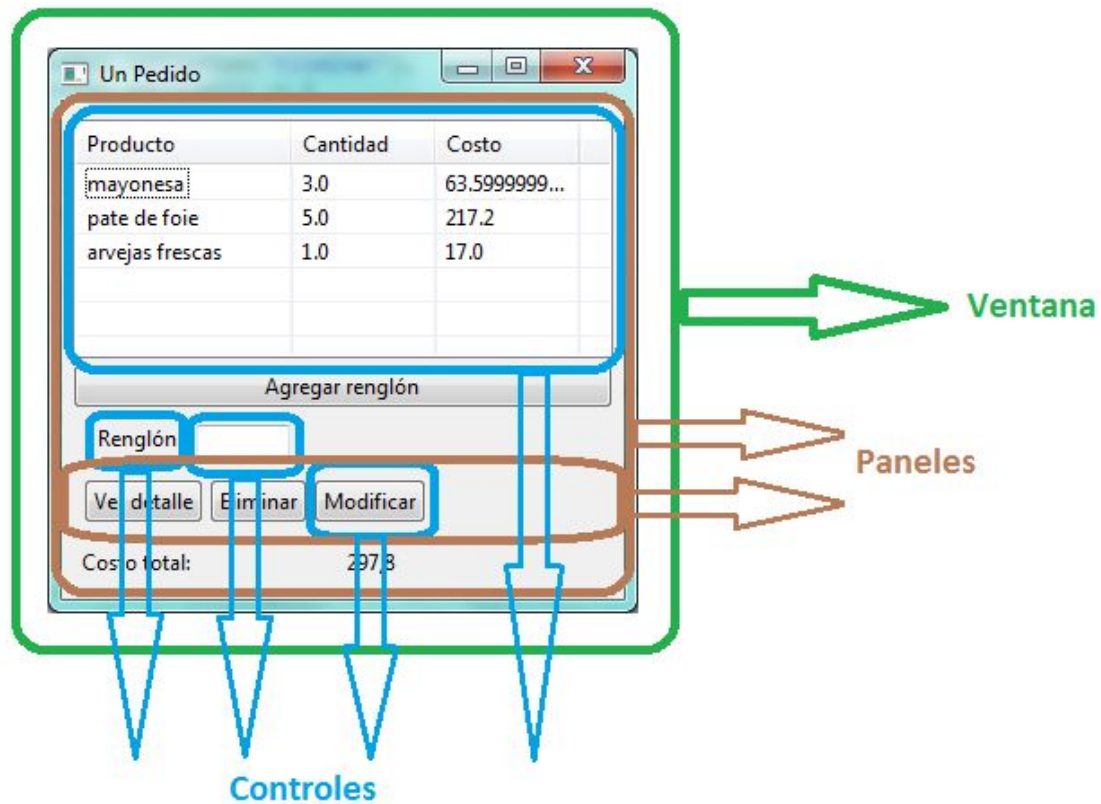
¹ Dependiendo de la tecnología, la forma de describir o modelar ventanas, paneles, controles y layouts, y la relevancia de cada uno de estos conceptos, y la forma de reflejarlos, va a variar. En este sentido, Arena es bien explícito: hay objetos ventana, objetos panel, objetos control y objetos layout.

despliegue de un dato simple (que muestra el dato pero no permite editarlo)

Un objeto **panel** representa un área del espacio de una ventana. Cada control se pone dentro de un panel. La ubicación de los controles se maneja a nivel panel.

Todas las ventanas tienen un panel principal. También puedo poner un panel adentro de otro panel. O sea, los paneles de una ventana forman un composite o árbol. Las hojas de este árbol son los controles.

Ubiquemos paneles y controles en la ventana que muestra un pedido.



Layout

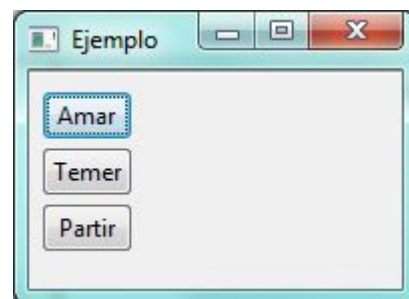
Un objeto layout es el que decide cómo se organizan los controles que se incluyen en un panel. Cada panel va a tener un layout asociado.

P. ej., en un panel con un layout horizontal, los controles se van a ubicar de izquierda a derecha. En un panel con un layout vertical, los controles se van a ubicar de arriba hacia abajo.

Tres botones en layout horizontal.



Los mismos tres botones, en layout vertical.



Bindings

Varias tecnologías de interfaz gráfica, entre ellas Arena, permiten definir *bindings*, o sea, una vinculación directa entre elementos gráficos y elementos del modelo.

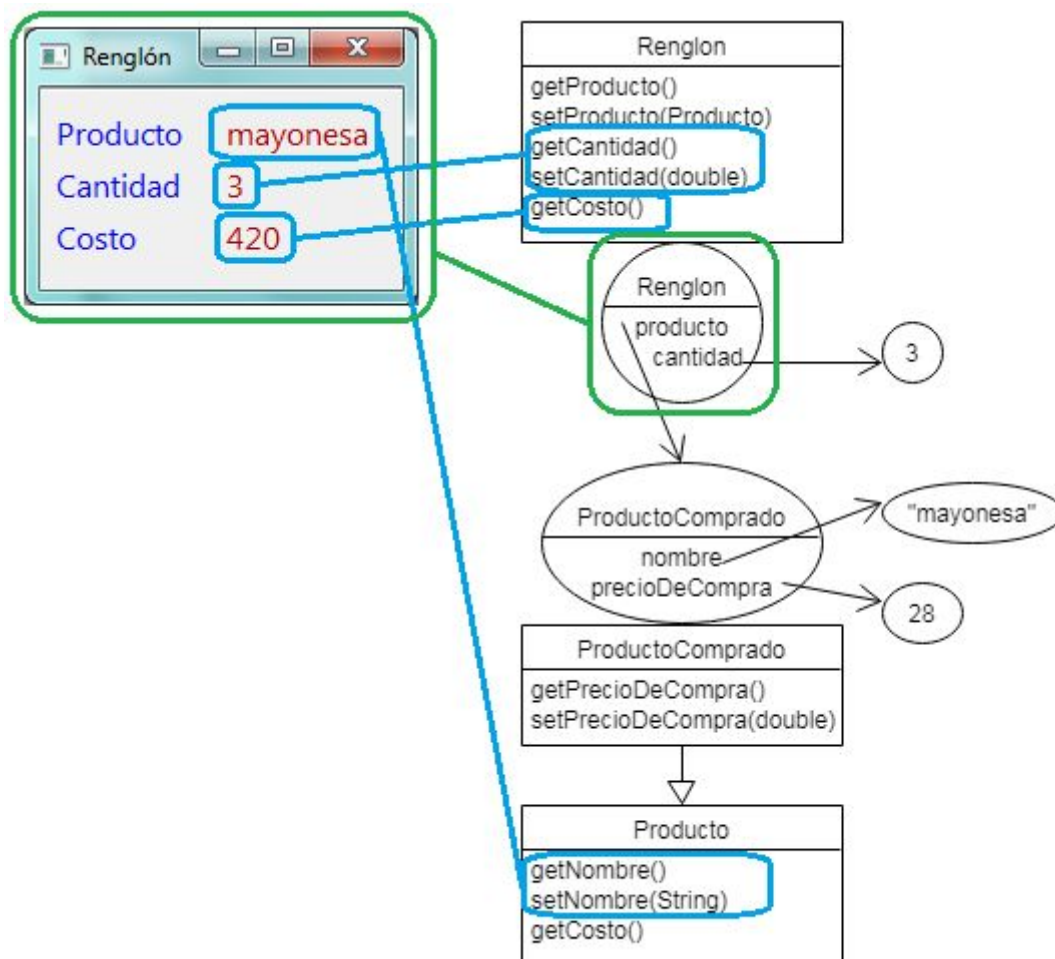
Específicamente, en Arena se establecen estos vínculos.

- Una ventana con un objeto, que es el *modelo de la ventana*. Este objeto se puede obtener enviándole el mensaje `getModelObject()` al objeto ventana.
- Un control con lo que se llama una *propiedad*.

En Java, una propiedad se define por dos métodos que están en una misma clase.

- Uno es el *getter*, el nombre del getter empieza con `get`. Con el getter se obtiene el valor de la propiedad.
- El otro método es el *setter*, el nombre del setter empieza con `set`. Con el setter se modifica el valor de la propiedad.

Una propiedad puede corresponder directamente a un atributo, o bien ser *calculada* (p.ej. el costo en Renglon es una propiedad calculada). Si está el getter pero no el setter, se puede hablar de una propiedad *read-only*. Las propiedades *read-only* son, casi siempre, calculadas. En el ejemplo del gráfico de acá abajo, la clase `Renglon` define tres propiedades: `producto`, `cantidad` y `costo`. Las dos primeras tienen un atributo asociado, la tercera es *read-only*. La clase `Producto` define el atributo `nombre`.



El vínculo, o binding, se establece mediante el nombre de la propiedad. Lo que muestra el control es el valor de la propiedad para el modelo de la vista. Al lado de cada objeto pusimos su clase, y para la

mayonesa también la superclase Producto, para que se vea qué mensajes entiende cada objeto, y por lo tanto, qué propiedades pueden *bindearse* a elementos de la interfaz. En el ejemplo, el control del medio está bindeado a la propiedad “cantidad”. Lo que muestra el control es el valor de la propiedad para el modelo de la ventana, en este caso 3. El control del renglón de arriba se vincula a la propiedad “nombre”, pero no del modelo, sino del valor que tenga la propiedad “producto” del modelo². En el ejemplo, el producto del renglón es la instancia de ProductoComprado que se ve en el gráfico, y su nombre es “mayonesa”.

Eventos

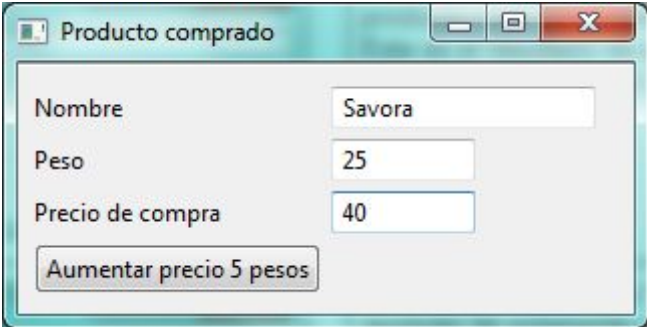
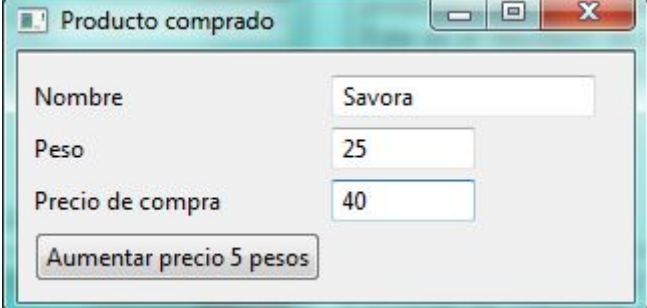
Un **evento** es cualquier acción que toma el usuario: completar el valor de un control que se puede editar (p.ej. un campo en el que se puede ingresar un número o un texto), pulsar un botón, cerrar una ventana, incluso pasar el mouse sobre un control.

Una aplicación maneja algunos de estos eventos, el resto los ignora.

Para cada evento que se decide manejar, hay que indicar cómo es el manejo.

En un control editable, si se establece un binding bidireccional entre ese control y una propiedad de un objeto, eso alcanza para manejar el ingreso de un valor en el campo. Arena (o cualquier tecnología que soporte vínculos automáticos bidireccionales) va a asignar, automáticamente, la propiedad vinculada con el valor que se ingresa.

Ejemplo: un campo de texto bindeado a la propiedad nombre, en una ventana vinculada a un ProductoComprado. Cuando cambiamos el valor en la ventana, automáticamente al producto vinculado se le envía el mensaje `setNombre(String)` con el valor ingresado.

	<p>Se abre una ventana de edición para un producto cuyo nombre es “Savora”. Este es el resultado de enviarle al producto el mensaje <code>getNombre()</code>.</p>
	<p>Se cambia el nombre a “Mostaza”. Si después de solamente cambiar “Savora” por “Mostaza” en la pantalla, se envía el mensaje <code>getNombre()</code> al mismo objeto producto, ahora devuelve “Mostaza”. Se actualizó automáticamente, por el vínculo bidireccional entre el campo editable y la propiedad del producto.</p>

² Varias tecnologías, entre ellas Arena, manejan el concepto de *propiedad anidada*. El nombre del producto se puede ver como una propiedad anidada del renglón, la propiedad `producto.nombre`.

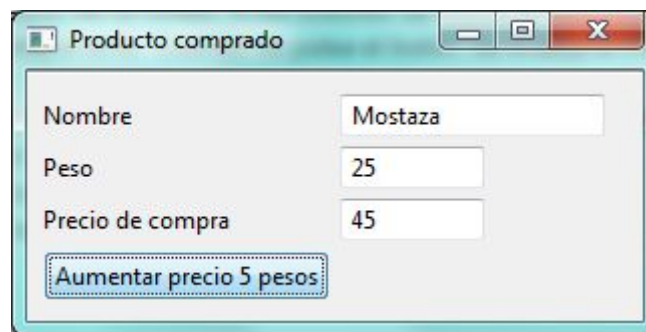
Para otros eventos, hay que indicar explícitamente cómo responder. Para esto, algunos controles entienden mensajes que llevan un parámetro de tipo closure. Cuando ocurre el evento, se evalúa el closure.

Un ejemplo típico es el pulsado de un botón. A los botones de Arena se les puede enviar el mensaje `onClick` con un closure como evento. Cuando se pulsa el botón, se evalúa el closure. El closure no lleva parámetro.

Veamos la configuración del botón incluido en la ventanita de producto. Este código está dentro de un método de la ventana, cuyo modelo es un `ProductoComprado`.

```
Button botonInflacion = new Button(mainPanel)
botonInflacion.setCaption("Aumentar precio 5 pesos")
botonInflacion.onClick(() -> this.getModelObject().aumentarPrecioDeCompra(5));
```

Recordemos que al enviarle `getModelObject()` a la ventana se obtiene el objeto modelo, en este caso el producto. En la clase `ProductoComprado` se agregó el método `aumentarPrecioDeCompra(int)`, que hace lo que uno espera. Si pulsamos el botón, el efecto va a ser el deseado:



se envía el mensaje al producto, que aumenta su precio de 40 a 45. A su vez, como el campo del precio de compra está bindeado con la propiedad `precioDeCompra` del producto, y el vínculo es bidireccional, entonces al cambiar el valor de la propiedad, solita cambia la ventana.

¡Gracias!

A Gisela Decuzzi por la revisada y los comentarios.