



## Tutorial para Desarrollar Módulos: Recibir Eventos del Mouse

(Parte 2)

### ÍNDICE

Índice.....	1
1    Introducción.....	2
2    Como realizar el cambio de figura.....	2
2.1    Cambiando la clase <i>ShapeCellRenderer.java</i> .....	2
2.2    Manejando eventos de entrada en la clase <i>ShapeCell.java</i> .....	3



(Parte 2)

# TUTORIAL PARA DESARROLLAR MÓDULOS: RECIBIR EVENTOS DEL MOUSE (PARTE 2)

## 1 INTRODUCCIÓN

En este tutorial usted aprenderá como hacer que el cell (módulo) que creo en la parte 1, responda a los eventos del Mouse usando el API creado para Wonderland. Se modificará el Shape Cell para que cambie de figura, entre un cubo y una esfera, cada vez que el usuario de clic en la figura. La figura cambiará de acuerdo a su forma actual, si es un cubo, al darle clic se convertirá en una esfera, y si es una esfera, se convertirá en cubo.

Para poder llevar a cabo este tutorial, es necesario haber completado la primera parte de esta serie de tutoriales, pues se explicará como complementar el código creado en la parte 1 para que acepte los eventos del Mouse.

## 2 COMO REALIZAR EL CAMBIO DE FIGURA

En este tutorial solo será necesario modificar dos clases que fueron creadas en el tutorial anterior, y son: *ShapeCell.java* y *ShapeCellRenderer.java*. Se recomienda ampliamente usar el código escrito en el tutorial anterior, a pesar de que se harán varias modificaciones a los métodos que están en ellas. Las modificaciones que se harán son, a grandes rasgos, modificar la clase *ShapeCell.java* para que acepte eventos del teclado, y la clase *ShapeCellRenderer.java* para que muestre un cubo o una esfera, según sea el caso.

### 2.1 CAMBIANDO LA CLASE *ShapeCellRenderer.java*

Primero se modificará la clase *ShapeCellrenderer.java* para que al mandar llamar el método *updateShape()* se actualice la figura actual de acuerdo al nombre que esté guardado en la clase *ShapeCell*. El método *updateShape()* obtendrá el nombre de la figura de la clase *ShapeCell*, y

## Tutorial para Desarrollar Módulos: Recibir Eventos del Mouse

(Parte 2)

creará la figura de acuerdo al tipo recibido, quitará la figura anterior y le agregará la nueva figura al Nodo. Antes que nada, hay que importar la siguiente clase:

```
import org.jdesktop.wonderland.client.jme.ClientContextJME;
```

Ahora, usted puede implementar el método como sigue:

```
public void updateShape() {
    String name = cell.getCellID().toString();
    String shapeType = ((ShapeCell) cell).getShapeType();

    node.detachAllChildren();
    node.attachChild(this.getShapeMesh(name, shapeType));
    node.setModelBound(new BoundingBox());
    node.updateModelBound();

    ClientContextJME.getWorldManager().addToUpdateList(node);
}
```

Note que, en cualquier momento que cambie la escena gráfica, debe de indicarle al manejador del mundo de Wonderland el nodo que ha sido actualizado, esto se hace con el método *addToUpdateList()*, este método le indica al subsistema de gráficos 3D que lo re-dibuje, en este caso para actualizar su figura.

### 2.2 MANEJANDO EVENTOS DE ENTRADA EN LA CLASE *ShapeCell.java*

En la clase *ShapeCell.java* se debe de modificar el código para que recibir notificaciones en cualquier momento en que alguien haga clic con el Mouse sobre la figura. En wonderland 0.5, los cell's del lado del cliente registran los eventos del Mouse ocurridos sobre los objetos gráficos (entidades), en este caso, el cubo.

Para llevar a cabo lo anterior, es necesario importar las siguientes clases justo debajo de la declaración del paquete:

```
import org.jdesktop.wonderland.common.cell.CellStatus;
import org.jdesktop.wonderland.client.input.EventClassListener;
import org.jdesktop.wonderland.client.jme.input.MouseButtonEvent3D;
import org.jdesktop.wonderland.client.jme.input.MouseEvent3D.ButtonId;
import org.jdesktop.wonderland.client.input.Event;
```

Después, debe de sobrescribir otro método declarado en la clase Cell: *setStatus()*. Este método es llamado cuando ha cambiado el estatus del cell en el cliente. El estatus del cell del lado del cliente es un conjunto de estados que muestran cuando el cell es cargado en memoria, cuando está "cerca" o visible para el avatar. Existen los siguientes estatus para el cell, definidos en la clase **CellStatus**:



## Tutorial para Desarrollar Módulos: Recibir Eventos del Mouse

(Parte 2)

- **DISK:** El cell está en disco, sin estar en memoria
- **BOUNDS:** El cell y sus límites están en memoria
- **INACTIVE:** La geometría del cell está en memoria, pero no está siendo mostrada
- **ACTIVE:** El cell está “cerca” del avatar
- **VISIBLE:** El cell está en pantalla

En su implementación del método *setCellStatus()* usted agregará un “listener” para cuando el cell esté en el estado **ACTIVE** (esto es, que el avatar esté relativamente cerca del cell) y quitará el “listener” cuando el cell regrese al estatus de **BOUNDS**. El sistema garantiza que en cualquier momento que el estatus del cell cambie, pase por todos los estatus intermedios.

Primero, defina una propiedad en la clase *ShapeCell* para manipular el “listener” (que será definido mas abajo):

```
private MouseEventListener listener = null;
```

Ahora, defina el método *setCellStatus()* como sigue:

```
@Override
public boolean setStatus(CellStatus status) {
    super.setStatus(status);
    switch (status) {
        case DISK:
            if (listener != null) {
                listener.removeFromEntity(renderer.getEntity());
                listener = null;
            }
            break;

        case ACTIVE:
            if (listener == null) {
                listener = new MouseEventListener();
                listener.addToEntity(renderer.getEntity());
            }
            break;

        default:
            break;
    }
    return true;
}
```

Cuando el cell esté en el estatus de **ACTIVE**, se creará una nueva instancia (un nuevo objeto) de tipo *MouseListener* y lo agrega a una entidad (Nótese que se agregan “listeners” a

## Tutorial para Desarrollar Módulos: Recibir Eventos del Mouse

(Parte 2)

entidades, no precisamente a cells). De este modo, usted puede simplemente agregar un "listener" a la entidad raíz que se crea en el cell renderer. Cuando el estatus del cell cae en **BOUNDS** (esto es, cuando tu avatar esta realmente lejos del cell), se quitará el "listener" de la entidad.

La clase *MouseEventListener* es una clase interna de la clase *ShapeCell*. Usted la puede definir como sigue:

```
class MouseEventListener extends EventClassListener {
    @Override
    public Class[] eventClassesToConsume() {
        return new Class[] { MouseButtonEvent3D.class };
    }

    // Note: No sobrescribimos el método computeEvent por que no hacemos
    // ningun cálculo en este "listener"

    @Override
    public void commitEvent(Event event) {
        MouseButtonEvent3D mbe = (MouseButtonEvent3D)event;
        if (mbe.isClicked() == false || mbe.getButton() !=
        ButtonId.BUTTON1) {
            return;
        }
        shapeType = (shapeType.equals("BOX") == true) ? "SPHERE" : "BOX";
        renderer.updateShape();
    }
}
```

La clase de *MouseEventListener* hereda de la clase *EventClassListener* que es una base abstracta que le ayudará a implementar "listeners" de eventos. También implementa la interfase *EventListener*. Es raro que usted necesite implementar la interfase de *EventListener* en sus clases para Wonderland, la clase abstracta base provee la mayoría de las funcionalidades necesarias.

El método de *eventClassesToConsume()* regresa un arreglo de objetos de clases de java que el "listener" maneja. En este caso, el listener maneja solo eventos de botones del mouse (*MouseButtonEvent3D.class*). Todos estos eventos deben de heredar de la clase de Wonderland *Event* (en el paquete **org.jdesktop.wonderland.client.input**). Las siguientes clases manejadoras de eventos están definidas en el paquete de **org.jdesktop.wonderland.client.jme.input**:

Clases Manejadoras de eventos de entrada	<u>Descripcion</u>
--	--------------------

## Tutorial para Desarrollar Módulos: Recibir Eventos del Mouse

(Parte 2)

MouseButtonEvent3D	Representa la presión, soltura y clics de los botones del Mouse, incluyendo el número de clics
MouseEvent3D	Representa a los movimientos del mouse, sin incluir presión de un botón y movimiento del mouse al mismo tiempo
MouseEvent3D	Representa los movimientos del mouse mientras se mantiene presionado algún botón
MouseEvent3D	Representa cualquier momento en que el mouse entra o sale de un cell
MouseEvent3D	Representa el movimiento de la rueda del mouse (mouse scroll), incluyendo la cantidad de veces que se da un "clic" en el movimiento.
KeyEvent3D	Representa todas presiones, soltadas y tecladas de las teclas (valga la redundancia). Una teclada es presionar y soltar una tecla inmediatamente

*El contenido de este tutorial está basado en el tutorial de "Extending Wonderland: Accepting Mouse Input in the Shape Cell (Part 2)", de la página <http://wiki.java.net/bin/view/Javadesktop/ProjectWonderlandModuleTutorialPart2Dev3>*



## Tutorial para Desarrollar Módulos: Recibir Eventos del Mouse

(Parte 2)