

Introducción al desarrollo de RIA's con Adobe Flex 3.0 Dia 4

by S. Muñoz-Gutiérrez
stalinmunoz@yahoo.com,
informes@grupolinda.org

Grupo LINDA
Facultad de Ingeniería

UNAM México Octubre-Diciembre 2009



Identificando objetos en mxml

```
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="hola()">
  <mx:Script>
    <![CDATA[
      private function hola():void{
        etiqueta1.text = "Hola";
        etiqueta2.text = "mundo!";
      }
    ]]>
  </mx:Script>
  <mx:HBox>
    <mx:Label id="etiqueta1" />
    <mx:Label id="etiqueta2" />
  </mx:HBox>
</mx:Application>
```

El compilador genera un miembro público para la clase con el mismo nombre

Utilizamos el atributo id

Código AS3 dentro de elementos XML

- Utilizar llaves `{ }` para indicar el código AS3
- Si más de una instrucción, separar con punto y comas
- Para cadenas internas utilizar comillas simples

```
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
    ]]>
  </mx:Script>
  <mx:Button id="bt" label="say hello"
    click="{Alert.show('Hola mundo!');bt.label='saluda'}" />
</mx:Application>
```

Atributos XML en objetos mxml

```
<mx:Application  
  xmlns:mx="http://www.adobe.com/2006/mxml">
```

Espacio de nombres



```
<mx:Button  
  id="bt"  
  label="say hello"  
  fontSize="40"  
  click="{bt.label='Di hola'}" />
```

Atributos del objeto



Estilos



Eventos



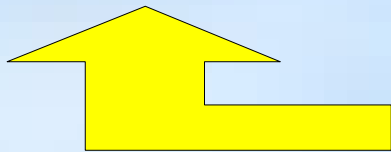
```
</mx:Application>
```

ActionScript 3.0

- Basado en ECMAScript Edición 3, Edición 4*
- Se ejecuta en ActionScript Virtual Machine dentro de Flash Player
- Orientado a Objetos
- Excepciones en tiempo de ejecución
- XML es un tipo nativo
- Soporte para expresiones regulares
- Uso de espacio de nombres
- Modelo de eventos

Declaración de variables y constantes

- `var contador:Number;`
- `var contador:Number = 0;`
- `var mensaje:String;`
- `var mensaje:String = "Hola mundo!";`
- `const TASA_IVA:Number = 0.17; //Ouch!`
- `const PI:Number = 3.1416;`



Buena práctica: usar mayúsculas y guión bajo para nombrar constantes

Tipos de datos (Todos son objetos)

- Básicos:

- **Object**

- String
- Number
- Int, uint
- Boolean

- Complejos:

- Array
- Date
- XML
- RegExp
- Function
- Event

Palabras reservadas

as	break	case	catch
class	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

Comentarios

```
var n:Number = 3; //número de iteraciones
```

```
/* Comentarios que se expanden múltiples líneas  
de código al mismo estilo de lenguaje c*/
```

Operadores primarios

- [] inicializar arreglos
- { x: y } inicializar objetos
- () agrupar expresiones
- f(x) invocar una función
- new crear nueva instancia
- x.y, x[y] acceder a una propiedad
- <></> definir objetos XML
- @ acceder atributos XML
- :: manejo de nombres XML
- .. acceso a descendientes XML

Operadores posfijos

- ++ incremento posterior
- -- decremento posterior

```
var x: Number = 0;
```

```
trace(x++); // 0
```

```
trace(x); // 1
```

```
trace(x--); // 1
```

```
trace(x); // 0
```

Operadores unarios

- ++ (prefijos) incremento
- -- (prefijos) decremento
- + signo positivo
- - signo negativo
- ! negación lógica
- ~ negación bit a bit
- delete elimina una propiedad
- typeof información del tipo
- void sin valor de retorno

Operadores multiplicativos

- Operadores binarios:
 - * multiplicación
 - / división
 - % módulo

Operadores aditivos

- Operadores binarios:
- + Adición
- - Substracción

Operadores de bits

- Operadores binarios
- << corrimiento de bits a la izquierda
- >> corrimiento de bits a la derecha
- >>> corrimiento a la derecha sin signo

Operadores relacionales

- $<$ menor que
- $>$ mayor que
- $<=$ menor igual que
- $>=$ mayor igual que
- `as` casting
- `in` verificación de propiedad
- `instanceof` verificación de de herencia
- `is` verificación de tipo

Operadores de igualdad

- == igualdad de objetos
- != desigualdad de objetos
- === igualdad estricta
- !== desigualdad estricta

Operadores lógicos de bits

- $\&$ Conjunción a nivel de bits
- \wedge OR exclusiva a nivel de bits
- $|$ Disyunción a nivel de bits

Operadores lógicos

- && Conjunción lógica
- || Disyunción lógica

Operador condicional

- Operador ternario:
- ? : Si, en caso contrario

Operadores de asignación

- Asignación:
- =
- Operar con acumulación:
- *=
- /=
- %=
- +=
- -=
- <<=
- >>=
- >>>=
- &=
- ^=
- |=

Condicional if, if-else, if-else-if

```
if(x==3){  
    //condición true  
}
```

```
if(x==3){  
    //condición true  
}else{  
    //condición false  
}
```

```
if(x==3){  
    //condición true  
}else if(x==2){  
    //segunda condición  
    // true  
}else{  
    //ninguna condición  
    //true  
}
```

Condicional Switch

```
switch(x){  
  case 1:  
    //Código si x==1  
    break;  
  case 2:  
    //código si x==2  
    break;  
  default:  
    //código si x no es ninguno de los casos  
    //anteriores  
    break;  
}
```

Ejercicio

Implementé una calculadora con las operaciones básicas de suma, resta, multiplicación y división. El usuario debe poder oprimir los números del 0 al 9 y el punto. Debe existir un botón para cada operación, un botón para limpiar y un botón de resultado. El usuario puede guardar su resultado en una memoria al oprimir un botón y oprimiendo otro botón puede recuperar el número almacenado. Hay un botón de apagado que se habilita cuando el usuario oprime el botón ALT, pero se deshabilita si se oprime cualquier otro botón.