

# Introducción al desarrollo de RIA's con Adobe Flex 3.0 Dia 6

by S. Muñoz-Gutiérrez  
[stalinmunoz@yahoo.com](mailto:stalinmunoz@yahoo.com),  
[informes@grupolinda.org](mailto:informes@grupolinda.org)

Grupo LINDA  
Facultad de Ingeniería

UNAM México Octubre-Diciembre 2009



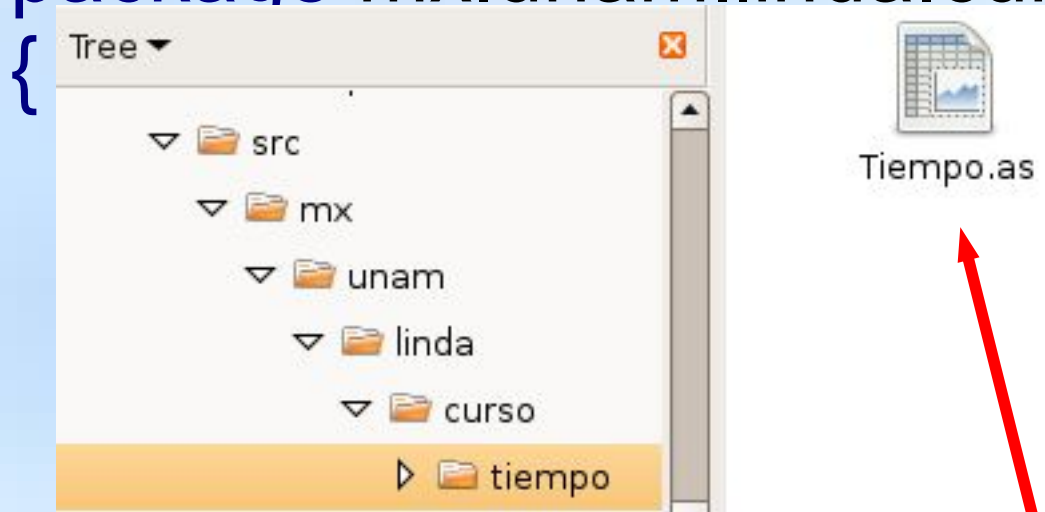
# Definición de clases

```
//Definición del paquete
package mx.unam.linda.curso
{
    //Clases a importar
    import mx.collections.ArrayCollection;
    //Definición de la clase
    public class Tiempo{
        //Atributos y métodos
        //...
    }
}
```

# Paquetes

//Definición del paquete

```
package mx.unam.linda.curso.tiempo
```



La estructura de organización de paquetes debe corresponder con la estructura de directorios donde se encuentra el archivo de la clase

El nombre de la clase corresponde con el nombre del archivo con extensión "as"

```
public class Tiempo{  
    //Atributos y métodos  
}
```

Por convención los nombres de clases comienzan con una letra mayúscula

# Modificadores de clases

- **dynamic**  
Permite agregar propiedades de manera dinámica en tiempo de ejecución
- **final**  
No permite que existan clases hijas
- **internal** (default)  
El acceso a la clase se permite solo a otras clases dentro del mismo paquete
- **public**  
El acceso se permite a clases de cualquier paquete

# Miembros de clase

- Propiedades: variables y constantes
- Métodos

```
public class Tiempo{  
    //Atributo  
    private var horas:int;  
    //Método  
    public function isAM():Boolean{  
        return horas <12;  
    }  
}
```

# Modificadores de miembros

- **internal**      visibilidad dentro del paquete
- **private**      visibilidad dentro de la clase
- **protected**    visibilidad misma clase e hijas
- **public**        visibilidad desde cualquier clase
- **static**        propiedad de clase no de instancia
- **<namespace>**    defnido por usuario

# Getters y setters

- Permiten el acceso a atributos privados

```
public class Tiempo{  
    //Atributo  
    private var horas:int;  
    //obtiene las horas, getter  
    public function get horas():int{  
        return horas;  
    }  
    //establece las horas, setter  
    public function set horas(horas:int):void{  
        this.horas = horas;  
    }  
}
```

# Atributos estáticos

- Se asocian a la clase no a la instancia

```
public class Tiempo{  
    public static const SEC_X_HOUR:int = 3600;  
}
```

- No se requiere una instancia para accederlos

```
x = Tiempo.SEC_X_HOUR * 27;
```

# Constructores de clase

- Métodos constructores comparten el nombre de la clase y son implícitamente publicos

```
public class FechaHora extends Fecha{  
    private var _date:Date;
```

```
//método constructor
```

```
public function FechaHora(date:Date) {  
    super(date);  
    _date = date;  
}
```

Pueden invocar el constructor de la clase padre usando super()

```
}
```

# Métodos estáticos

- Están asociados a la clase, no a la instancia

```
public class Tiempo{  
    public static function time(tiempo:Tiempo):int{  
        //implementación del método  
        //no se tiene acceso a miembros no estáticos  
    }  
}
```

```
tn = Tiempo.time(tf)-Tiempo.time(ti);
```

# Interfaces

- Permiten desacoplar clases
- Proveen con una colección de definiciones de métodos

//Definición de interface

```
package mx.unam.linda.curso.interfaces
{
    public interface IComparable{
        function equals(objeto:IComparable):Boolean;
    }
}
```

# Implementación de interfaces

- Se utiliza la palabra reservada `implements`
- Se puede implementar más de una interface

```
public class LapsoTiempo implements
                                Comparable{
    //implementación de método definido en
    //interface
    public function equals(
        objeto:Comparable):Boolean{
        //implementación
    }
}
```

# Herencia

- Para indicar que una clase es hija de otra se utiliza la palabra reservada **extends**

```
public class A{
```

```
public class B extends A{
```

```
public class C extends B{
```

- Atributos y métodos definidos como **protected** o **public** son visibles desde las clases hijas
- No se puede tener más de una clase padre
- Miembros estáticos no son heredados

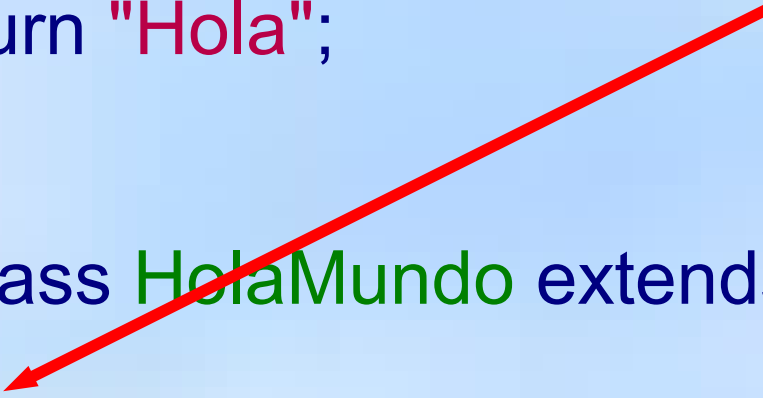
# Sobreescritura de métodos

- Las clases hijas cambian el comportamiento de la clase padre al sobreescribir los métodos de la clase padre o bien agregar nuevos

```
public class Hola {  
    public function say():String  
    {  
        return "Hola";  
    }  
}
```

Usamos override para  
sobreescribir el método

```
public class HolaMundo extends Hola  
{  
    override public function say():String  
    {  
        return super.say() + " mundo";  
    }  
}
```



# Ejercicio

Implemente clases para representar el horario de un alumno de la FI. El alumno esta inscrito a 4 materias. El **horario** es una asignación de **grupos** a **salones**. Cada **materia** puede impartirse en 1, 2, 3 o 4 sesiones. Cada **sesión** corresponde con la asignación de un grupo a un salón en un **lapso** de tiempo. Los lapsos estan definidos por una hora inicio y hora fin con resolución de 15 minutos.

Considere un catálogo con 3 salones y 6 grupos para 4 materias.

Utilice ComboBox para mostrar las opciones de los catálogos. CheckBox para los dias de la semana y NumericStepper para los lapsos.

Tras capturar la información y oprimir un botón de "aceptar horario" verifique que los horarios no se traslapen y las materias no se repitan.