



Le contrôle

ActionScript
Gestion des événements



Découvrir ActionScript

■ Le langage

- ActionScript est un langage de script orienté objet
- Basé sur l'ECMAScript, un langage développé à partir des premières versions de JavaScript
- Apporte des possibilités supplémentaires à MXML
 - Gestion des événements
 - Interaction avec le contenu



ActionScript 3.0

- Deux façons d'introduire ce code

- Soit à l'intérieur du fichier MXML, soit dans des fichiers séparés
- Première façon : insérer du code entre les balises `<mx:Script>` et `</mx:Script>`

- Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">
  <mx:Script>
    <![CDATA[
      // code ActionScript 3.0
    ]]>
  </mx:Script>
  <mx:TextArea id="textArea" editable="false"/>
</mx:Application>
```



ActionScript 3.0 dans MXML

- Première façon : ATest1.mxml

- L'appel se fait en cliquant sur un bouton

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">
  <mx:Script>
  <![CDATA[
public function sayHello():void {
  lblResults.text = "Hello ";
}
]]>
</mx:Script>
<mx:Panel x="10" y="10" width="306" height="200" layout="absolute"
  title="Say Hello">
  <mx:Button x="178" y="36" label="Dis Bonjour!" click="sayHello()"/>
  <mx:Label x="10" y="66" id="lblResults"/>
</mx:Panel>
</mx:Application>
```

ActionScript 3.0 dans MXML

- Une autre première façon : ActionScriptTest1.xml

- L'appel se fait à l'ouverture de l'application par **creationComplete** qui se déclenche automatiquement une fois les composants prêts

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute" creationComplete="printHello()">
```

```
<mx:Script>
```

```
<![CDATA[
```

```
public function printHello():void{
    textAreaId.text=giveHelloWorld();}
```

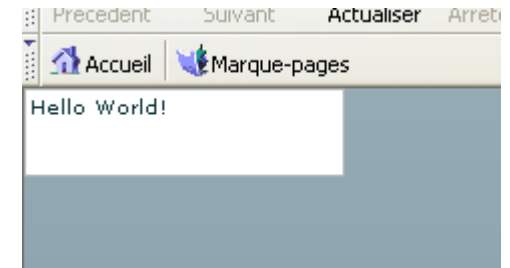
```
public function giveHelloWorld():String{
    return "Hello World!";}
```

```
]]>
```

```
</mx:Script>
```

```
<mx:TextArea id="textAreaId" editable="false" />
```

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





ActionScript 3.0 dans MXML

- Deuxième façon : le code .as dans un fichier séparé

- Etapes :

1. Créer un nouveau dossier : File >> New >> Folder
2. Nommez le **classes** (au niveau du src)
3. Sélectionnez le projet puis allez dans le menu File/New/Classe ActionScript
4. Appelez ce fichier HelloWorld
 - on obtient : **HelloWorld.as** dans le dossier **classes** qui est :

```
package classes{  
    public class HelloWorld{  
        public static function giveHelloWorld():String{  
            return "Hello World!";}  
        }  
    }  
}
```

5. On retire maintenant la fonction giveHelloWorld du fichier MXML pour la placer dans la classe HelloWorld :



ActionScript 3.0 dans MXML

- L'appeler à la création : `ActionScriptTest2.MXML`

```
<?xml version="1.0" encoding="utf-8"?>
  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" creationComplete="printHello()">
    <mx:Script>
      <![CDATA[
        import classes.HelloWorld;
        public function printHello():void
        {
            textAreaId.text = classes.HelloWorld.giveHelloWorld();
        }
      ]]>
    </mx:Script>
    <mx:TextArea id="textAreaId" editable="false"/>
  </mx:Application>
```



Gestion des événements

■ Utiliser les événements

- La mise en place d'un événement s'effectue en deux temps
 - **Déclaration** d'une fonction (listener) qui déclenche une réaction
 - Cette fonction utilise l'objet **Event** pour accéder aux propriétés qui définissent l'événement
 - **Liaison** du listener à un objet en spécifiant le type d'événement attendu (un clic de souris par exemple)

Gestion des événements

■ Utilisation dans MXML

1. On crée une fonction listener : `myListener(...)` comme suit :

```
<mx:Script>
  <![CDATA[
    import mx.controls.alert;
    private function myListener(event:Event):void {
      Alert.show("Événement déclenché");}
  ]]>
</mx:Script>
```

❖ Cette fonction utilise la classe `Alert` de Flex qui permet, via la méthode `show()`, de déclencher un pop-up

2. On utilise ensuite l'événement dans MXML

```
<mx:Button id="myButton" label="Cliquer ici"
  click="myListener()"/>
```

`evenementBouton1.mxml`

Cliquer ici





Gestion des événements

http://livedocs.adobe.com/flex/3/html/help.html?content=tooltips_3.html

■ Utilisation de l'événement dans AS

- On associe la méthode `addEventListener` à un nom de variable

```
myButton.addEventListener(MouseEvent.CLICK,  
myListener);
```

- `addEventListener` prend deux arguments :
 - Le type de l'événement attendu (ici un click de souris)
 - et la méthode à associer à cet événement, ici `myListener`

■ Exemple complet : eventCode.mxml

- Il s'agit, en cliquant sur un bouton, de créer un nouveau bouton

```
...<mx:Script><![CDATA[
```

```
    public function createNewButton(event:MouseEvent):void {  
        //on crée un autre bouton, avec une variable Button à qui on  
        associe un événement
```

```
        var myButton:Button = new Button();
```

```
        myButton.label = "Create Another Button";
```

```
        myButton.addEventListener(MouseEvent.CLICK,  
            createNewButton);
```

```
        addChild(myButton);
```

```
    }
```

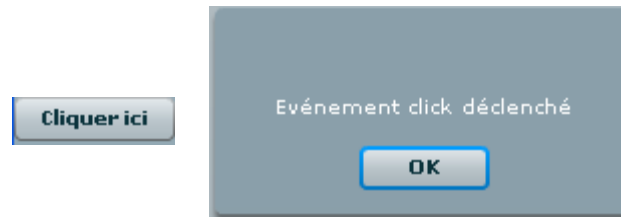
```
]]></mx:Script>
```

```
<mx:Button id="b1" label="Create Another Button"  
    click="createNewButton(event);"/>
```

Gestion des événements

- Manipuler l'objet Event : `eventType.mxml`
 - On peut accéder aux propriétés de l'événement dans le listener, afin d'en connaître l'origine et le type

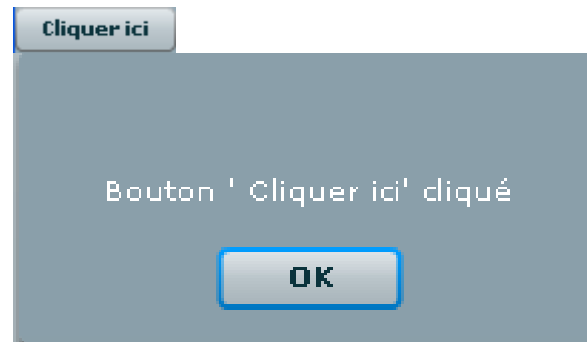
```
function myListener(event:Event):void {  
    Alert.show("Événement " +event.type+" déclenché");  
}
```



Gestion des événements

- On peut également connaître le bouton à l'origine de l'événement : `eventTarget.mxml`

```
function myListener(event:Event):void {  
    Alert.show("Bouton ' " +event.currentTarget.label+"  
    cliqué");  
}
```



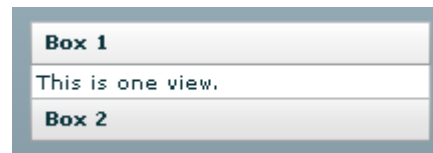


Gestion des événements

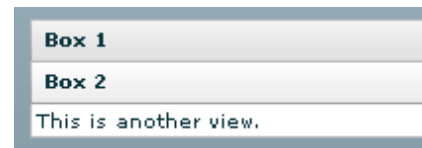
- Les événements à connaître
 - Les plus importants sont
 - `creationComplete`
 - ❖ Déclenché lorsqu'un composant et tous ses fils sont créés : on peut alors initialiser les valeurs de certaines variables ou composants
 - `Error`
 - ❖ Déclenché à la production d'une erreur
 - `Scroll`
 - ❖ Déclenché à chaque fois que l'utilisateur défile avec sa souris dans l'application
 - `Change`
 - ❖ Déclenché dès qu'un composant subit une modification
 - ❖ Très utile lors de l'utilisation d'une `ComboBox` ou d'un composant personnalisé

Gestion des événements

- Exemple de l'événement : change
 - Le container <mx:Accordion>
 - Contient plusieurs éléments mais seul le contenu d'un seul est visible
 - En cliquant sur un bouton, son contenu devient visible, et le contenu de l'autre bouton sera caché
 - Ce phénomène peut être géré par l'attribut **change**



En cliquant sur Box 1



En cliquant sur Box 2

- Exemple pour change : `evenementChange.mxml`

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
width="300" height="280">
```

```
<mx:Script>
```

```
  <![CDATA[
```

```
    import mx.controls.Alert;
```

```
    private function handleAccChange():void {
```

```
      Alert.show("You just changed views.");}
```

```
  ]]></mx:Script>
```

```
<mx:Accordion id="myAcc" height="60" width="200"
  change="handleAccChange();">
```

```
  //Les éléments avec leurs valeurs
```

```
  <mx:HBox label="Box 1">
```

```
    <mx:Label text="This is one view."/>
```

```
  </mx:HBox>
```

```
  <mx:HBox label="Box 2">
```

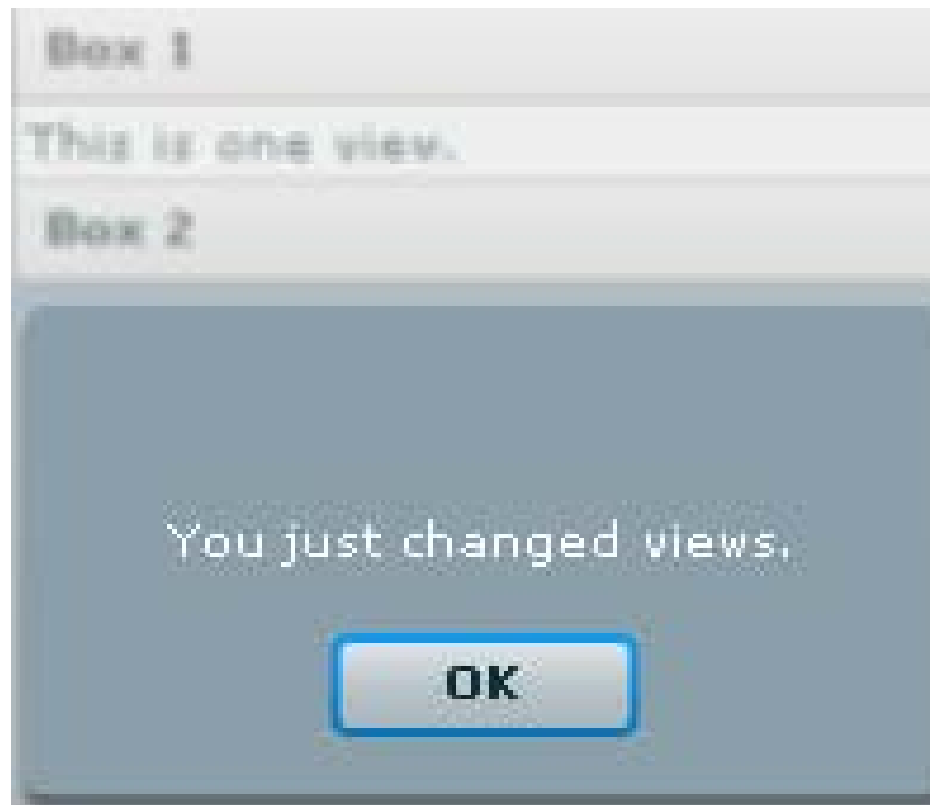
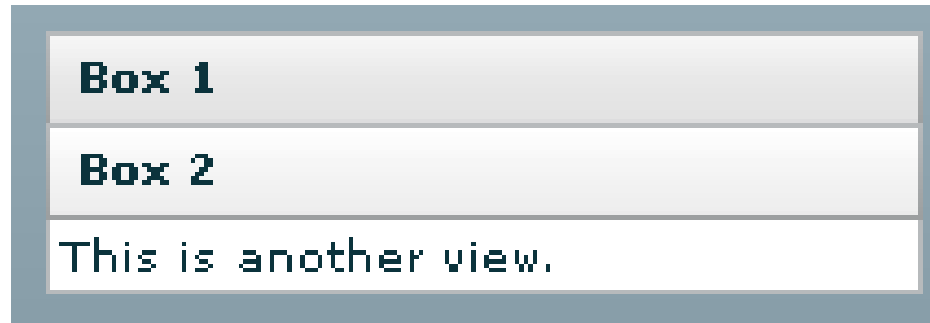
```
    <mx:Label text="This is another view."/>
```

```
  </mx:HBox>
```

```
</mx:Accordion>
```

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


Réagit dès qu'on touche à un composant





Gestion des événements

- Les événements Clavier
 - Flex propose 2 types d'événements
 - ❖ Appui : `keyboardEvent.KEY_DOWN`
 - ❖ Relâche : `keyboardEvent.KEY_UP`

Gestion des événements

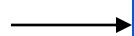
- Exemple du keyUp : **EvenementText.mxml**
 - Déclare un TextInput et deux Labels, un pour écrire et l'autre pour récupérer ce qui a été écrit
 - Dès qu'on relâche l'appui sur une touche, le texte écrit est recopié dans la zone "my_text"

```
<mx:Label text="Entrez votre texte" id="my_label"/>
```

```
<mx:TextInput id="my_input" keyUp="autoFill()"  
y="26" x="0"/>
```

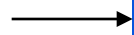
```
<mx:Label x="0" y="56" text="le texte écrit"  
id="my_text"/>
```

my_input



Entrez votre texte

my_text

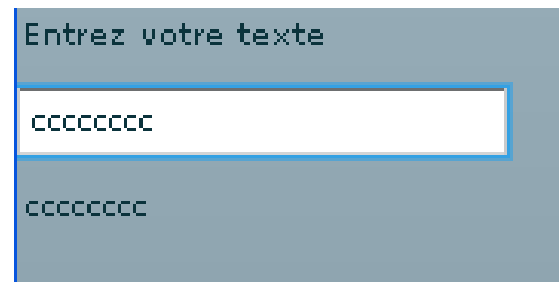


cccccccc

Gestion des événements

- La fonction autoFill()

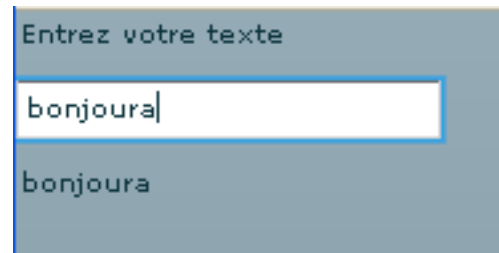
```
</mx:Script>  
  <![CDATA[  
    import mx.controls.Alert;  
    import mx.events.CloseEvent  
    private function autoFill():void {  
      my_text.text=my_input.text  
    }  
  ]]  
</mx:Script>
```



Gestion des événements

- **Récupérer les codes clavier : evenementText2.mxml**
 - On peut récupérer le code de la touche pressée
 - Pour cela, on utilise le **keyCode** (spécifique à la touche pressée) ou le **charCode** (spécifique au caractère correspondant, suivant le code ASCII)
 - Il suffit de modifier la fonction autoFill précédente

```
function autoFill(event:KeyboardEvent):void {  
    if(event.charCode == 97)  
        my_text.text=my_input.text  
}
```
 - Désormais, le texte ne sera recopié que si on appuie sur la lettre a (=97 en ASCII)



Gestion des événements

■ Autre exemple : evenementText3.mxml

- Ici, on écrit le code ASCII du dernier caractère tapé et son code clé

```
... creationComplete="initApp();">
```

```
<mx:Script><![CDATA[
```

```
private function initApp():void {
```

```
Application.application.addEventListener(KeyboardEvent.  
KEY_UP, keyHandler);
```

```
//application est une propriété de <mx:Application>
```

```
}
```

```
private function keyHandler(event:KeyboardEvent):void {
```

```
t1.text =event.keyCode + "/" + event.charCode;
```

```
}
```

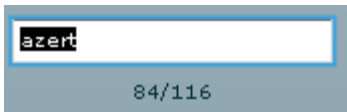
```
]]>
```

```
</mx:Script>
```

```
<mx:TextInput id="myTextInput"/>
```

```
<mx:Text id="t1"/>
```

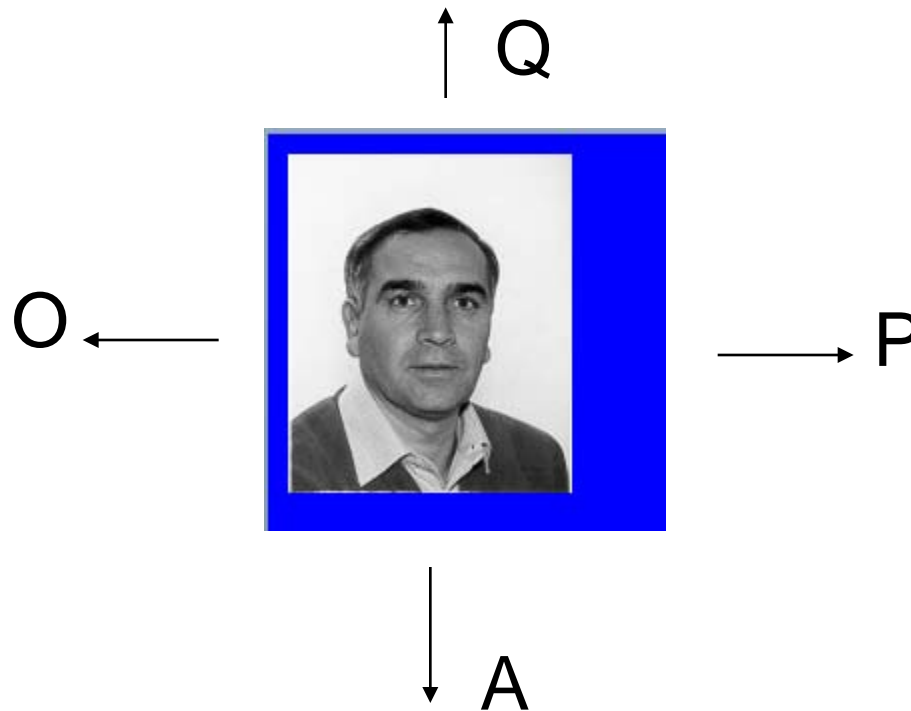
```
</mx:Application>
```



keyCode et charCode
de la lettre t

Gestion des événements

- Usage du keycode : `keycode.mxml`
 - Il s'agit d'afficher une image dans un canvas et de la déplacer en appuyant sur les touches : A, Q, P, O



```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  creationComplete="initApp();">
  <mx:Script>
  <![CDATA[
    private function initApp():void {
      img1.setFocus(); //comme au départ le focus n'est pas placé sur une des
        touches de déplacement de l'image, on l'utilise pour mettre le focus sur l'image
      application.addEventListener(KeyboardEvent.KEY_DOWN, keyHandler);
    }

    private function keyHandler(event:KeyboardEvent):void {
      if (event.keyCode == 81) {img1.y -= 10;} //P ?
      if (event.keyCode == 65) {img1.y += 10;} //O ?
      if (event.keyCode == 79) {img1.x -= 10;} //A ?
      if (event.keyCode == 80) {img1.x += 10;} //Q ?
    }
  ]]>
</mx:Script>

<mx:Canvas height="200" width="200" backgroundColor="blue">
  <mx:Image id="img1" x="10" y="10" source="photo.jpg" />
</mx:Canvas>

</mx:Application>

```




Gestion des événements

■ Les événements Clavier-Souris

- Nous allons voir maintenant comment combiner le clavier et la souris
- Lors d'un clic de souris, Flex développe un événement de type `MouseEvent`, cet objet dispose de plusieurs propriétés permettant de vérifier si une touche a été pressée

Attribut	Description
<code>altKey</code>	Si la touche Alt est pressée, la valeur de cette propriété est true
<code>ctrlKey</code>	Si la touche ctrl est pressée, la valeur de cette propriété est true
<code>shiftKey</code>	Si la touche shift est pressée, la valeur de cette propriété est true

■ Exemple : gestion du Ctrl+Click : evenementText5.mxml

```
<?xml version="1.0" encoding="utf-8"?>
```

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

```
<mx:Script>
```

```
<![CDATA[
```

```
  private function autoFill(event:MouseEvent):void{
```

```
    if(event.ctrlKey)my_label.text=my_input.text;}}]>
```

```
</mx:Script>
```

```
<mx:Label x="10" y="67" text="Entrez votre texte" id="my_label"/>
```

```
<mx:TextInput x="10" y="87" id="my_input"/>
```

```
<mx:Button id="myButton" label="Ctrl+click" click="autoFill(event)"
  x="10" y="107"/>
```

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



Appuyer sur Ctrl, puis sur le bouton



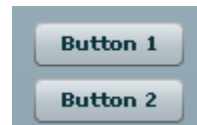
Après saisie et ctrl+clik

■ Autre exemple : shiftClik.mxml

- Ajoute des boutons en cliquant sur Reset



- Supprime le dernier bouton en faisant shift+click





Gestion des événements

- Propagation d'un événement
- Dans certains cas, vous pouvez avoir envie de faire réagir un composant de votre application à partir d'un événement particulier
- Il faut dans ce cas définir l'événement d'origine
- Lier cet événement à une fonction qui va créer un autre événement plus spécifique et le propager au même objet ou un autre objet
- **On utilise la méthode `dispatchEvent()`**
 - Cette méthode permet de propager un événement affecté à un objet à un autre objet



Gestion des événements

- Exemple avec le même objet :
`dispatch_manu3.mxml`
 - Ici, l'événement « survol de souris » déclenche automatiquement l'événement click de souris sur le même objet b1

```

<<?xml version="1.0"?>
<!-- events/DispatchEventExampleInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script><![CDATA[
    import mx.controls.Alert;

    private function dispatch(e:Event):void {
      //fait l'appel à la fonction au click de la souris
      b1.addEventListener(MouseEvent.CLICK, myClickHandler);
      //lie l'événement survol à l'événement click pour l'objet b1
      b1.dispatchEvent(new MouseEvent(MouseEvent.CLICK, true, false));
    }

    private function myClickHandler(e:Event):void {
      Alert.show("The event dispatched by the MOUSE_OVER was of type " +
        e.type + ".");
    }
  ]]></mx:Script>

  // l'objet déclencheur
  <mx:Button id="b1" label="Click Me" mouseOver="dispatch(event)"
  />

</mx:Application>

```



TD 4

- Question 1

- Voir Fiche



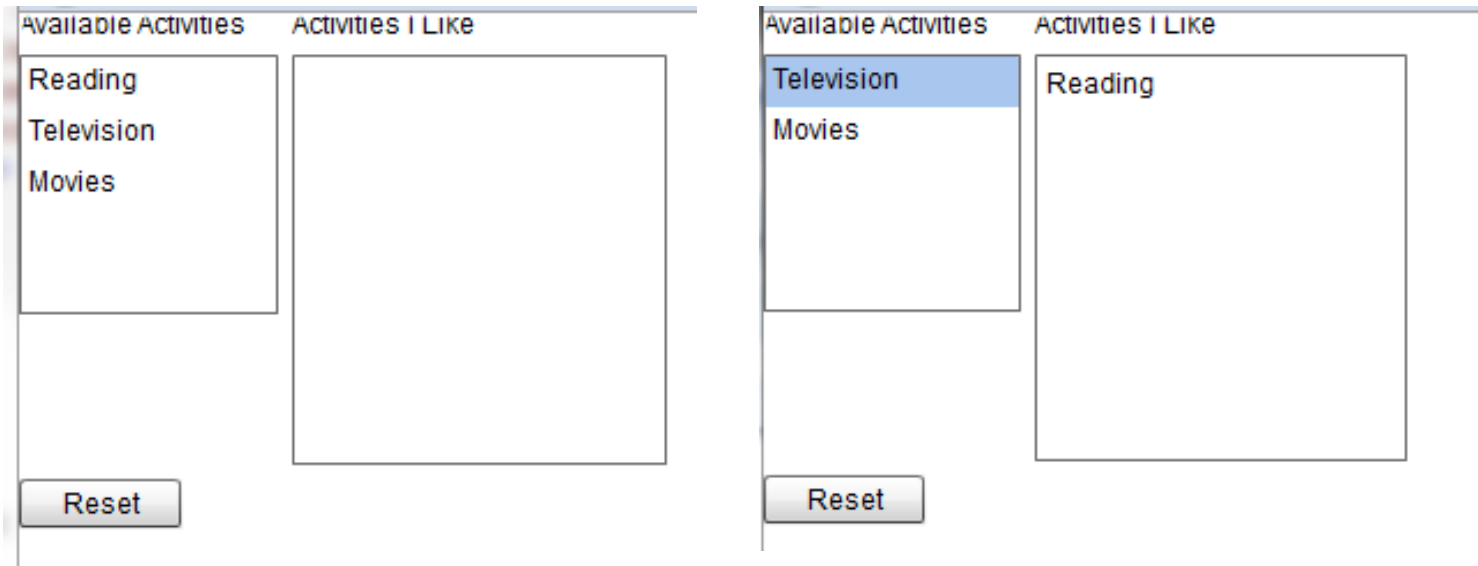
Application : le Drag and Drop

■ Le Drag and Drop simple

- i.e. sans gestion d'événements
- Les conteneurs suivants à base de listes de contrôle comprennent un support intégré pour l'opération du glisser-déposer:
 - `DataGrid`, `HorizontalList` Liste, `PrintDataGrid`, `TileList`,
Arbre
- Ils disposent de propriétés :
 - ❖ `dragEnabled` , `dropEnabled` et `dragMoveEnabled`
- `dragEnabled = true` : laisse glisser ses éléments
- `dropEnabled = true` : laisse déposer
- `dragMoveEnabled = true` : laisse déplacer

http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf69084-7cfd.html

- **Exemple : dragdrop_SimpleListToListMove.mxml**
 - Déplacer un élément d'une liste vers un autre conteneur



```
<s:Application ... creationComplete="initApp();">
```

```
<fx:Script>
```

```
<![CDATA[
```

```
import mx.collections.ArrayCollection;
```

```
private function initApp():void {
```

```
    //la source est remplie au départ
```

```
    srcList.dataProvider =
```

```
    new ArrayCollection(['Reading', 'Television',  
    'Movies']);
```

```
    //la destination est vide au départ
```

```
    destList.dataProvider = new ArrayCollection([]);
```

```
}]>
```

```
</fx:Script>
```

```
<s:HGroup>
<s:VGroup>
  <s:Label text="Available Activities"/>
  <s:List id="srcList" allowMultipleSelection="true"
    dragEnabled="true" dragMoveEnabled="true"/>
</s:VGroup>
<s:VGroup>
<s:Label text="Activities I Like"/>
  <s:List id="destList" dropEnabled="true"/>
</s:VGroup>
</s:HGroup>
<s:Button id="b1" label="Reset" click="initApp();"/>
```

■ Remarques

- En réglant les propriétés **dragEnabled** et **dropEnabled** à vrai, cela permet de glisser des éléments de la première liste à la seconde sans se soucier de l'un des traitements des événements sous-jacents
- La seule contrainte est la compatibilité entre les deux conteneurs
- Ici :
 - le data provider pour **srclist** est un Array de Strings,
 - Le data provider pour la liste de destination est Array vide
 - Si le data provider de destlist est un Array d'un autre type que liste, destlist peut ne pas recevoir correctement la donnée glissée

- Un autre exemple : `dragdrop_SimpleDGToDG.mxml`
 - Ici, on peut glisser-déposer des lignes d'un DataGrid à un autre

Available Albums			Buy These Albums		
Artist	Album	Price	Artist	Album	Price
Paul Simon	Graceland	10.99	Carole King	Tapestry	11.99
Original Cast	Camelot	12.99			
The Beatles	The White Album	11.99			

Reset

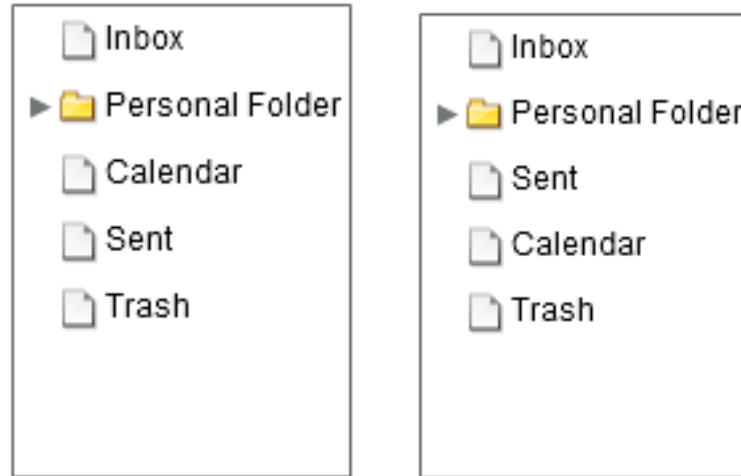
■ La première dataGrid

```
<mx:DataGrid id="srcgrid"  
  allowMultipleSelection="true"  
  dragEnabled="true"  
  dropEnabled="true"  
  dragMoveEnabled="true">  
  <mx:columns>  
    <mx:DataGridColumn dataField="Artist"/>  
    <mx:DataGridColumn dataField="Album"/>  
    <mx:DataGridColumn dataField="Price"/>  
  </mx:columns>  
</mx:DataGrid>
```

■ La deuxième dataGrid

```
<mx:DataGrid id="destgrid"  
  allowMultipleSelection="true"  
  dragEnabled="true"  
  dropEnabled="true"  
  dragMoveEnabled="true">  
<mx:columns>  
  <mx:DataGridColumn dataField="Artist"/>  
  <mx:DataGridColumn dataField="Album"/>  
  <mx:DataGridColumn dataField="Price"/>  
</mx:columns>  
</mx:DataGrid>
```

- **Un autre exemple : dragdrop_SimpleTreeSelf.mxml**
 - Ici, on peut glisser-déposer des éléments à l'intérieur du même contrôle : Tree



■ Déclaration de l'arbre

```
<mx:Tree id="firstList"  
  showRoot="false"  
  labelField="@label"  
  dragEnabled="true"  
  dropEnabled="true"  
  allowMultipleSelection="true"  
  creationComplete="initApp();"/>
```



Application : le Drag and Drop

- En utilisant les événements :
 - Deux classes sont utilisées
 - DragManager
 - DragEvent



Application : le Drag and Drop

■ La classe DragManager

- gère les opérations de glisser-déposer, pour déplacer des données d'un endroit à un autre dans une application Flex
- Toutes les méthodes et les propriétés de DragManager sont statiques, donc pas besoin de créer une instance



Application : le Drag and Drop

- Lorsque l'utilisateur sélectionne un élément avec la souris, le composant sélectionné est appelé l'initiateur du déplacement
- L'image qui s'affiche pendant l'opération de glisser est appelée le **proxy** glisser
- Lorsque l'utilisateur déplace le proxy glisser sur un autre composant, l'événement **dragEnter** est envoyé à ce composant
- Si le composant accepte le glisser, il devient la cible de dépôt et reçoit les événements **dragOver**, **dragExit** et **dragDrop**
- Lorsque le déplacement est terminé, un événement **dragComplete** est envoyé à l'initiateur du déplacement



Application : le Drag and Drop

■ Propriétés de DragManager

- **isDragging** : booléen renvoie vrai si un glissement est en cours.
- **prototype** : Object : référence à l'objet prototype d'un objet de classe ou fonction



Application : le Drag and Drop

■ Méthodes de DragManager

- **acceptDragDrop** (cible: IUIComponent): vide :
 - à appeler à partir de votre gestionnaire d'événements dragEnter si vous acceptez les glisser / déposer des données
- **doDrag** (dragInitiator: IUIComponent , DragSource: DragSource , mouseEvent: MouseEvent , dragImage: IFlexDisplayObject = null, xOffset: Nombre = 0, yOffset: Nombre = 0, imageAlpha: Nombre = 0,5, allowmove: booléen = true): vide
 - Lance une opération de glisser-déposer.



Application : le Drag and Drop

- **getFeedback** (): cordes
 - Renvoie le glisser-déposer des commentaires en cours
- **showFeedback** (feedback: cordes): vide
 - Définit l'indicateur de réaction pour l'opération de glisser-déposer



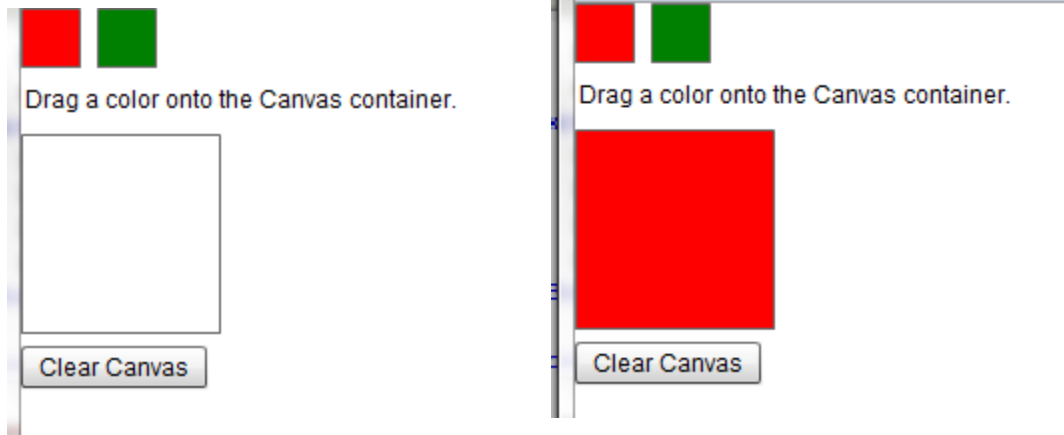
Application : le Drag and Drop

■ La classe `DragEvent`

- représente les objets event qui sont envoyés dans le cadre d'une opération de glisser-déposer
- Quelques propriétés
 - **`dragInitiator`** : `IUIComponent` : La composante qui a initié le glisser
 - **`draggedItem`** : `Object` : Si la propriété `dragInitiator` contient un objet `IAutomationObject`, cette propriété contient l'objet `IAutomationObject` enfant près du curseur de la souris
 - **`dragSource`** : `DragSource` : L'objet `DragSource` contenant les données à déplacer

■ Exemple : drag_drop1.mxml

- On peut glisser déposer une couleur dans le récipient vide



■ Exemple : drag_drop1.mxml

- Les canvas de couleurs : vert et rouge qu'on peut glisser-déposer

```
<mx:HBox>
```

```
  <mx:Canvas width="30" height="30"  
    backgroundColor="red" borderStyle="solid"  
    mouseMove="mouseMoveHandler(event);"/>
```

```
  <mx:Canvas width="30" height="30"  
    backgroundColor="green" borderStyle="solid"  
    mouseMove="mouseMoveHandler(event);"/>
```

```
</mx:HBox>
```

```
<mx:Label text="Drag a color onto the Canvas container."/>
```

■ Exemple : drag_drop1.mxml

<!-- Utiliser les événements dragEnter et dragDrop pour permettre les déplacements vers le récipient -->

```
<mx:Canvas id="myCanvas"
  width="100" height="100"
  backgroundColor="#FFFFFF"
  borderStyle="solid"
  dragEnter="dragEnterHandler(event);"
  dragDrop="dragDropHandler(event);"/>
```

```
<mx:Button id="b1" label="Clear Canvas"
  click="myCanvas.setStyle('backgroundColor',
    '0xFFFFFFFF');"/>
```

```
// Initialise les opérations de drag and drop  
private function mouseMoveHandler(event:MouseEvent):void {  
  
// Récupère le composant drag initiator de l'événement  
var dragInitiator:Canvas=Canvas(event.currentTarget);  
  
// Prend la couleur du composant drag initiator  
var dragColor:int = dragInitiator.getStyle('backgroundColor');  
  
// Crée un objet DragSource  
var ds:DragSource = new DragSource();  
  
// Ajoute la donnée à l'objet  
ds.addData(dragColor, 'color');  
  
// Appelle la méthode doDrag() pour commencer le drag  
DragManager.doDrag(dragInitiator, ds, event);  
}
```

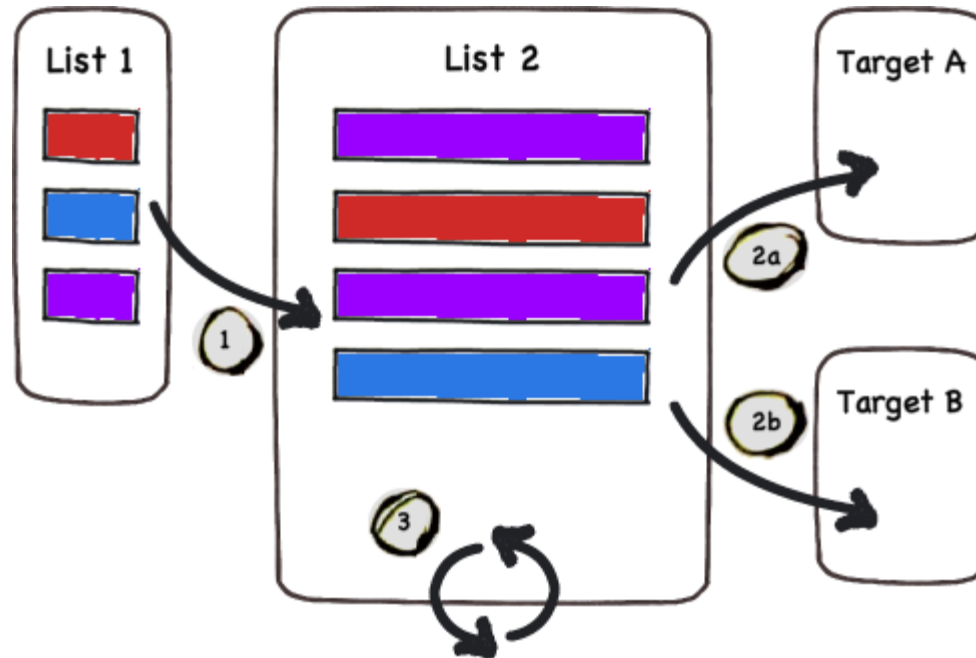
```
// Appelée quand l'utilisateur déplace le drag indicator sur la  
cible drop  
private function dragEnterHandler(event:DragEvent):void {  
  
// Accepte le drag seulement si l'utilisateur est entrain de  
draguer une donnée  
// identifié la valeur du format 'color'  
if (event.dragSource.hasFormat('color')) {  
  
// Récupère la cible drop à partir de l'objet événement  
var dropTarget:Canvas=Canvas(event.currentTarget);  
// Accepte le drop  
DragManager.acceptDragDrop(dropTarget);  
}  
}
```

```
// Appelé si la cible accepte l'objet glissé et l'utilisateur
// relâche le bouton de la souris au dessus du conteneur
Canvas private function
dragDropHandler(event:DragEvent):void {

// Récupère les données identifiées par le format de couleur de
// la source de glissement
var data:Object = event.dragSource.dataForFormat('color');
// Affecte la couleur au canvas
myCanvas.setStyle("background-color", data);
}
```

■ En résumé

- On peut résumer les actions du drag and drop dans ce schéma



- (1) faire glisser de la palette dans une liste
- (2a) et (2b) faire glisser des éléments de la liste dans une boîte cible
- (3) faire glisser des éléments à l'intérieur de la liste pour les réorganiser. Je vais les briser un par un.

■ #1 Glisser une liste d'Item dans une liste d'Item

```
<s:List id="sourceList" dragEnabled="true">  
  <s:ArrayList source="['A','B','C','D']" />  
</s:List>
```

```
<s:List id="targetList" dropEnabled="true">  
  <s:ArrayList>  
  </s:ArrayList>  
</s:List>
```

- On met dragEnabled à vrai dans la source et dropEnabled à vrai pour la destination
- Astuce 1 :
 - quand la liste cible est vide, lui donner un dataprovider vide (comme l'ArrayList)
- Astuce 2 :
 - La liste cible et la liste source doivent avoir des conteneurs compatibles

■ #2 — Glisser une List Item dans une cible UIComponent

```
<s:List id="sourceList" dragEnabled="true">  
<s:ArrayList source="['A','B','C','D']" />  
</s:List>
```

```
<s:Panel id="targetA"  
  dragEnter="dragEnterHandler(event)"  
  dragDrop="dragDropHandler(event)">  
</s:Panel>
```

- Quand la cible n'est pas une liste, comme c'est le cas ici, alors, on associe à **dragEnter** et **dragDrop** des gestionnaires d'événements

- Voici les gestionnaires d'événements

```
private function dragEnterHandler(e:DragEvent):void {  
    DragManager.acceptDragDrop(e.currentTarget as  
        UIComponent);  
}
```

```
private function dragDropHandler(e:DragEvent):void {  
    //do something with dropped item here  
}
```

- On accepte le glisser dans le gestionnaire dragEnter handler via la méthode acceptDragDrop() et une fois l'objet est glissé, on peut en faire ce qu'on en veut dans le gestionnaire dragDrop

■ #3 — Glisser une List Item pour les réorganiser en interne

```
<s:List id="reorderList"  
  dragEnabled="true"  
  dropEnabled="true"  
  dragMoveEnabled="true">  
  <s:ArrayList source="['A','B','C','D']" />  
</s:List>
```

- La Liste Spark fait tout le travail pour nous
- On a juste mis `dragEnable = true` et `dropEnable = true` sur la liste que nous voulons réorganiser
- On doit également positionner `dragMoveEnabled = true` qui contrôle les mouvements internes : déplacé, copié...

■ Création d'effet lors du déposer

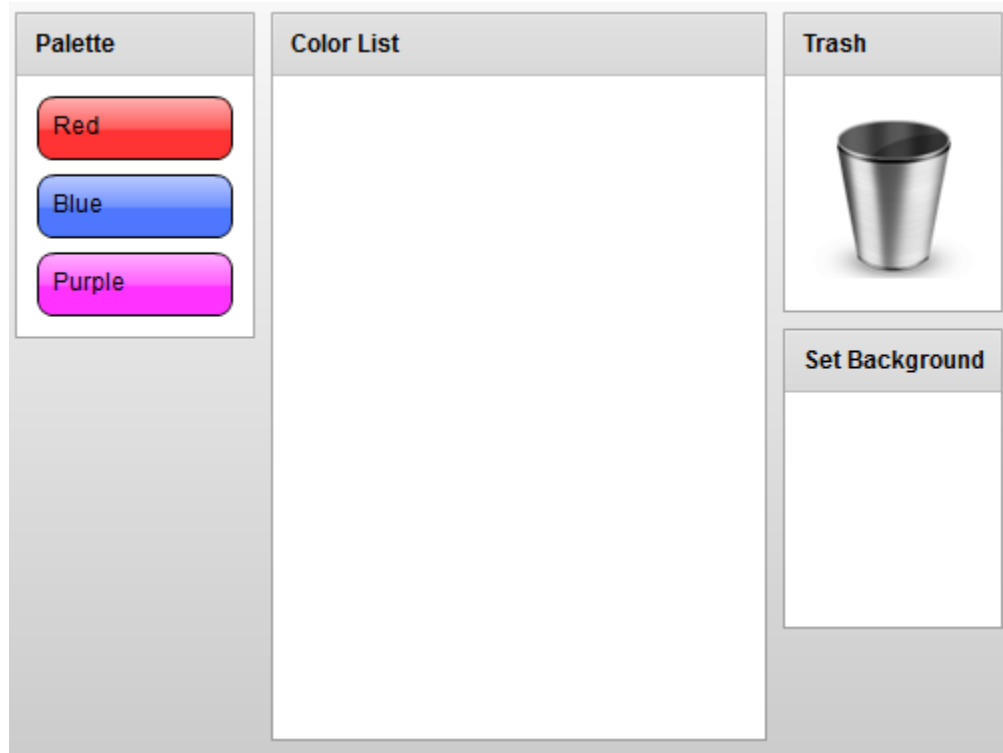
- Quand on glisse de List à List, le contrôleur List crée un effet lors du dépôt
- Quand on utilise des objets de type non List UIComponent, on doit créer nous même cet effet : appelé **glow**

```
private function dragEnterHandler(e:DragEvent):void {
    myGlow.alpha = 1;
    DragManager.acceptDragDrop(e.currentTarget as
        UIComponent);
}
private function dragExitHandler(e:DragEvent):void {
    myGlow.alpha = 0;
}
private function dragDropHandler(e:DragEvent):void {
    myGlow.alpha = 0;
    //do something with dropped item here
}
```

- Dans le gestionnaire dragEnter, on active le glow, et on le désactive soit sur l'événement dragExit ou dragDrop
- Et on a juste besoin d'ajouter un simple GlowFilter au composant cible :

```
<s:Panel id="targetA"  
  dragEnter="dragEnterHandler(event)"  
  dragDrop="dragDropHandler(event)">  
<s:filters>  
  <s:GlowFilter id="myGlow" blurX="16" blurY="16"  
    color="#3399ff" alpha="0" />  
</s:filters>  
</s:Panel>
```

■ Exemple : DragDropColors.mxml





TD 4

- Question 2

- Voir Fiche



ActionScript

Éléments du langage



Les packages

■ Utilité

- Organiser le code à l'intérieur de votre application
- On peut les assimiler à des dossiers

■ Déclaration

Package nom-package

```
{  
    Public class nom-de-classe  
    {  
        public static fonction nom-de-fonction():type  
        {  
            ... return...  
        }  
    }  
}
```

■ Emplacement

- Un package doit correspondre à un répertoire dans le système de fichiers
- On peut également définir un sous-package (défini dans un sous répertoire de package). La déclaration sera :

`Package nom-de-package.nom-de-sous-package`



Les packages

■ Contenu

- Les packages sont conçus pour regrouper des classes
- Le nom du fichier ActionScript doit correspondre au nom de la classe qu'il contient
- Cette classe doit être déclarée comme public afin qu'elle soit accessible en dehors du package, de même pour les fonctions qu'elle contient

■ Utilisation

- Pour utiliser une classe, il faut l'importer
- Exemple
 - `Import nom-de-package.nom-de-classe;`
- On peut aussi importer toutes les classes
 - `Import nom-de-package.*;`



Les types de données et variables

- Types de données de base
 - Boolean, int, Number, String, uint, void, Null
- Déclaration de variables
 - var myVar; // on ne peut pas l'utiliser
 - Var myVar:int=42; // on peut l'utiliser
- Opérateur is : permet de vérifier le type
 - Exemple

```
var i:int = 42;
if (i is int) {//renverra true
//code exécuté}
if (i is String) {//renverra false
//code non exécuté}
```



Les types de données et variables

- Opérateur as :

- vérifie le type et retourne la valeur de l'expression si le type est vérifié
- Exemple

```
var i:int =42;
```

```
var res;
```

```
res= i as int; //res contiendra 42
```

```
res= i as String; //res contiendra null
```



Les types de données et variables

- Les conversions

- Exemple

```
var n:Number =3.14;  
var i:int =2;  
var str:String;  
str= String(n); //str contiendra "3.14"  
str= String(i); //str contiendra "2"  
str= "21";  
i=int(str); //i contiendra la valeur 21  
str= "37.42";  
n= Number(str); //n contiendra "37.42"
```



Les types de données et variables

■ Déclaration de fonctions

- Simplement : nom (paramètres) {code}
- Exemple

```
Function add(a:int, b:int)
{
    return a+b;
    // le code qui suit n'est plus exécuté
}
```

- Tous les paramètres sont passés par référence : on travaille sur le même objet

Les types de données et variables

■ Les opérateurs

Arithmétique		
+	addition	Ajoute des expressions numériques.
--	décrément	Soustrait 1 de l'opérande.
/	division	Divise <code>expression1</code> par <code>expression2</code> .
++	incrément	Ajoute 1 à une expression.
%	modulo	Calcule le reste de <code>expression1</code> divisé par <code>expression2</code> .
*	multiplication	Multiplie deux expressions numériques.
-	soustraction	Utilisé pour la négation ou la soustraction.
Affectation composée arithmétique		
+=	affectation addition	Affecte à <code>expression1</code> la valeur de <code>expression1 + expression2</code> .
/=	affectation division	Affecte à <code>expression1</code> la valeur de <code>expression1 / expression2</code> .
%=	affectation modulo	Affecte à <code>expression1</code> la valeur de <code>expression1 % expression2</code> .
*=	affectation multiplication	Affecte à <code>expression1</code> la valeur de <code>expression1 * expression2</code> .
-=	affectation soustraction	Affecte à <code>expression1</code> la valeur de <code>expression1 - expression2</code> .
Affectation		
=	affectation	Affecte la valeur <code>expression2</code> (opérande à droite) à la variable, l'élément de tableau ou une propriété dans <code>expression1</code> .

Les types de données et variables

■ Les opérateurs

Au niveau du bit		
&	ET au niveau du bit	Convertit <code>expression1</code> et <code>expression2</code> en entiers non signés de 32 bits, et exécute une opération Boolean ET sur chaque bit des paramètres de l'entier.
<<	décalage gauche au niveau du bit	Convertit <code>expression1</code> et <code>shiftCount</code> en entiers de 32 bits et décale tous les bits de <code>expression1</code> vers la gauche en fonction du nombre spécifié par l'entier résultant de la conversion de <code>shiftCount</code> .
~	SAUF au niveau du bit	Convertit l' <code>expression</code> en un entier signé de 32 bits, puis applique un complément à un au niveau du bit.
 	OU au niveau du bit	Convertit <code>expression1</code> et <code>expression2</code> en entiers non signés de 32 bits et insère un 1 à chaque position de bit, où les bits correspondant de <code>expression1</code> ou <code>expression2</code> , sont de 1.
>>	décalage droit au niveau du bit	Convertit <code>expression</code> et <code>shiftCount</code> en entiers de 32 bits et décale tous les bits de <code>expression</code> vers la droite en fonction du nombre spécifié par l'entier résultant de la conversion de <code>shiftCount</code> .
>>>	décalage droit non signé au niveau du bit	Identique à l'opérateur de décalage vers la droite au niveau du bit (<code>>></code>), à la différence qu'il ne conserve pas le signe de l'expression d'origine car les bits de gauche sont toujours remplacés par des 0.
^	XOR au niveau du bit	Convertit <code>expression1</code> et <code>expression2</code> en entiers non signés de 32 bits et insère un 1 à chaque position de bit, où les bits correspondant de <code>expression1</code> ou <code>expression2</code> , mais pas des deux, sont de 1.

Les types de données et variables

■ Les opérateurs

Affectation composée au niveau du bit		
&=	affectation ET au niveau du bit	Affecte à <code>expression1</code> la valeur de <code>expression1 & expression2</code> .
<<=	décalage gauche au niveau du bit et affectation	Effectue un décalage vers la gauche (<code><<=</code>) au niveau du bit et stocke ensuite le contenu dans <code>expression1</code> .
 =	affectation OU au niveau du bit	Affecte à <code>expression1</code> la valeur de <code>expression1 expression2</code> .
>>=	décalage droit au niveau du bit et affectation	Effectue un décalage au niveau du bit vers la droite et stocke le résultat dans <code>expression</code> .
>>>=	décalage droit non signé au niveau du bit et affectation	Effectue un décalage droite au niveau du bit non signé et stocke le résultat dans <code>expression</code> .
^=	affectation XOR au niveau du bit	Affecte à <code>expression1</code> la valeur de <code>expression1 ^ expression2</code> .
Commentaire		
/*..*/	séparateur bloc de commentaires	Délimite une ou plusieurs lignes de commentaires de script.
//	séparateur ligne de commentaire	Signale le début d'un commentaire de script.

Les types de données et variables

■ Les opérateurs

Commentaire		
<code>/*..*/</code>	séparateur bloc de commentaires	Délimite une ou plusieurs lignes de commentaires de script.
<code>//</code>	séparateur ligne de commentaire	Signale le début d'un commentaire de script.
Comparaison		
<code>==</code>	égalité	Vérifie si deux expressions sont égales.
<code>></code>	supérieur à	Compare deux expressions et détermine si <code>expression1</code> est supérieur à <code>expression2</code> . Dans l'affirmative, le résultat est <code>true</code> .
<code>>=</code>	supérieur ou égal à	Compare deux expressions et détermine si <code>expression1</code> est supérieur ou égal à <code>expression2</code> (<code>true</code>), ou si <code>expression1</code> est inférieur à <code>expression2</code> (<code>false</code>).
<code>!=</code>	inégalité	Recherche l'inverse de l'opérateur d'égalité (<code>==</code>).
<code><</code>	inférieur à	Compare deux expressions et détermine si <code>expression1</code> est inférieur ou égal à <code>expression2</code> . Dans l'affirmative, le résultat est <code>true</code> .
<code><=</code>	inférieur ou égal à	Compare deux expressions et détermine si <code>expression1</code> est inférieur ou égal à <code>expression2</code> . Dans l'affirmative, le résultat est <code>true</code> .
<code>===</code>	égalité stricte	Teste l'égalité de deux expressions, mais sans conversion automatique des données.
<code>!==</code>	inégalité stricte	Recherche l'inverse exact de l'opérateur d'égalité stricte (<code>===</code>).



Les types de données et variables

■ Les opérateurs

Logique		
&&	ET logique	Renvoie <code>expression1</code> s'il a la valeur <code>false</code> ou peut être converti en <code>false</code> , <code>expression2</code> dans tous les autres cas.
&&=	affectation ET logique	Affecte à <code>expression1</code> la valeur de <code>expression1 && expression2</code> .
!	SAUF logique	Inverse la valeur Boolean d'une variable ou d'une expression.
 	OU logique	Renvoie <code>expression1</code> s'il a la valeur <code>true</code> ou peut être converti en <code>true</code> , <code>expression2</code> dans tous les autres cas.
 =	affectation OU logique	Affecte à <code>expression1</code> la valeur de <code>expression1 expression2</code> .

Les types de données et variables

■ Les opérateurs

Autre		
<code>[]</code>	accès tableau	Initialise un nouveau tableau, simple ou multidimensionnel, avec les éléments spécifiés (a0, etc.) ou accède aux éléments d'un tableau.
<code>as</code>		Evalue si une expression spécifiée par le premier opérande est membre du type de données indiqué par le deuxième opérande.
<code>,</code>	virgule	Evalue <code>expression1</code> , puis <code>expression2</code> , et ainsi de suite.
<code>?:</code>	conditionnel	Evalue <code>expression1</code> , et si la valeur de <code>expression1</code> est <code>true</code> , le résultat reprend la valeur de <code>expression2</code> . Sinon, le résultat reprend la valeur de <code>expression3</code> .
<code>delete</code>		Détruit la propriété objet spécifiée par <code>reference</code> . Le résultat est <code>true</code> si la propriété n'existe pas à la fin de l'opération, et <code>false</code> dans tous les autres cas.
<code>.</code>	point	Accède aux méthodes et aux variables de la classe, récupère et définit les propriétés de l'objet et délimite les packages ou les classes importé(e)s.
<code>in</code>		Evalue si une propriété fait partie d'un objet spécifique.
<code>instanceof</code>		Evalue si la chaîne de prototype d'une expression comprend l'objet prototype de <code>fonction</code> .
<code>is</code>		Evalue si un objet est compatible avec un certain type de données ou une certaine classe ou interface.
<code>::</code>	qualificateur de nom	Permet d'identifier l'espace de nom d'une propriété, d'une méthode, d'une propriété ou d'un attribut XML.
<code>new</code>		Instancie une occurrence de classe.
<code>{}</code>	initialisateur objet	Crée un objet et l'initialise avec les paires de propriétés <code>name</code> et <code>value</code> spécifiées.
<code>()</code>	parenthèses	Regroupe un ou plusieurs paramètres, évalue les expressions de façon séquentielle ou entoure un ou plusieurs paramètres et les transmet en tant qu'arguments à une fonction placée avant les parenthèses.
<code>/</code>	séparateur RegExp	Lorsqu'ils entourent des caractères, indiquent que ceux-ci ont une valeur littérale et doivent être traités en tant qu'expression régulière (RegExp) et non pas en tant que variable, chaîne ou tout autre élément ActionScript.
<code>:</code>	type	Utilisé pour affecter un type aux données ; cet opérateur spécifie le type de variable, le type de renvoi de la fonction ou le type de paramètre de la fonction.
<code>typeof</code>		Evalue <code>expression</code> et renvoie une chaîne spécifiant son type de données.
<code>void</code>		Evalue une expression, puis supprime sa valeur en renvoyant <code>undefined</code> .

Les types de données et variables

■ Les opérateurs

Chaîne		
+	concaténation	Concatène (combine) des chaînes.
+=	affectation de concaténation	Affecte à <code>expression1</code> la valeur de <code>expression1 + expression2</code> .
"	séparateur de chaînes	Lorsqu'ils entourent des caractères, indiquent que ces caractères ont une valeur littérale et sont considérés comme une chaîne, et non pas en tant que variable, valeur numérique ou tout autre élément ActionScript.
XML		
@	identificateur attributs	Identifie les attributs d'un objet XML ou XMLList.
{ }	accolades (XML)	Evalue une expression utilisée dans un initialiseur XML ou XMLList.
[]	crochets (XML)	Accède à une propriété ou un attribut d'un objet XML ou XMLList.
+	concaténation (XMLList)	Concatène (combine) des valeurs XML ou XMLList dans un objet XMLList.
+=	affectation de concaténation (XMLList)	Affecte à <code>expression1</code> , qui est un objet XMLList, la valeur <code>expression1 + expression2</code> .
	delete (XML)	Supprime les éléments ou les attributs XML spécifiés par <code>reference</code> .
..	accesseur descendant	Navigue jusqu'aux éléments descendants d'un objet XML ou XMLList ou (combiné avec l'opérateur @) détecte les attributs correspondants des descendants.
.	point (XML)	Accède aux éléments enfant d'un objet XML ou XMLList, ou (combiné avec l'opérateur @) renvoie les attributs d'un objet XML ou XMLList.
()	parenthèses (XML)	Evalue une expression dans une construction E4X XML.
< >	séparateur balise de littéral XML	Définit une balise XML dans un littéral XML.



Éléments du langage

■ Les boucles

- La boucle for

```
var i:int;  
for(i=0;i<5;++i)  
    //code à exécuter
```

- La boucle for in

```
var tab:Array=["a", "b", "c"];  
for (var i:int in tab)  
    // i contient l'indice de l'élément dans le tableau et non sa  
    valeur
```

- La boucle for each in

```
var tab:Array=["a", "b", "c"];  
for (var i:String in tab)  
    // i la valeur de l'élément courant
```



Éléments du langage

■ Les boucles

- La boucle while

```
var i:int;
```

```
while (i<5)
```

```
    //code à exécuter
```

```
    ++i //incréméntation obligatoire
```

- La boucle do while

```
var i: int =0;
```

```
do {
```

```
    //code à exécuter
```

```
    ++i; //incréméntation obligatoire
```

```
} while (i<5);
```



Éléments du langage

- Les conditions

- If else

```
if (i<10) {  
    //code à exécuter  
} else {  
    //code sinon  
}
```

- La condition switch

```
var i: int =1;  
switch (i) {  
    case 0:  
        //code à exécuter  
        break;  
    case 1:  
        //code à exécuter  
        break;  
    default :  
        //code si aucune condition n'est vérifiée  
}
```




La POO dans ActionScript 3.0

■ Les classes : définition

- La définition se fait grâce au mot-clé **class** dans un fichier .as
- Exemple :

```
package myPackage
{
    public class MyClass
    {
    }
}
```

- **public** :
 - permet d'accéder à la classe depuis toute votre application
- **dynamic** :
 - permet d'ajouter des propriétés à la classe pendant l'exécution
- **final** :
 - la classe ne doit pas être héritée d'une autre classe
- **internal** :
 - c'est le comportement par défaut, la classe est visible dans le package



La POO dans ActionScript 3.0

■ Les attributs

- Chaque classe peut avoir des attributs qui la qualifient
- Exemple :

```
Package myPackage
{
    public class MyClass
    {
        public var var1:int =1; //visible partout
        private var var2:int = 2; //visible dans la classe
        protected var var3:int=3; //visible dans les classes
                                héritées
        internal var var4:int=4; //visible dans le package
        public const const1:int=0;
    }
}
```



La POO dans ActionScript 3.0

- Utilisation dans MXML : seul var1 est lisible à l'extérieur

```
<?xml version="1.0" encoding="utf-8"?>
  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
    layout="vertical" creationComplete="initApp()">
    <mx:Script>
      <![CDATA[
        import myPackage.MyClass
        public var objectTest:MyClass = new MyClass;
        public function initApp():void
        {
          objectValue.text=String(objectTest.var1)
        }
        public function addVar():void
        {
          objectTest.var1 +=1;
          objectValue.text = String(objectTest.var1);
        }
      ]]>
    </mx:Script>
    <mx:Button click="addVar()" label="Click me"/>
    <mx:Label id="objectValue"/>
  </mx:Application>
```



La POO dans ActionScript 3.0

■ Les méthodes

- Les protections sont les mêmes que pour les attributs
- Exemple

```
package myPackage
{
    public class MyClass
    {
        public var var1:int =1; //visible partout
        //constructeur
        public function MyClass():void
        {
            var1=0;
        }
    }
}
```



La POO dans ActionScript 3.0

■ Les interfaces : déclaration

- Ce sont des collections de déclarations de méthodes qui permettent à des objets indépendants de communiquer entre eux
- Déclaration

```
package myPackage{  
    public interface myInterface{  
        function addVar(toAdd:int):void;  
    }  
}
```



La POO dans ActionScript 3.0

- Les interfaces : implémentation

```
package myPackage{
    public class MyClass implements MyInterface{
        private var var1:int;
        public function addVar(toAdd:int):void{
            var1 +=toAdd;
        }
    }
}
```



La POO dans ActionScript 3.0

■ L'héritage

- Permet de réutiliser du code, d'organiser son application, et de profiter des propriétés comme le polymorphisme
- Exemple
 - Créer une classe de base : Shape dont les classes Circle et Square hériteront

```
package myPackage{  
    public class Shape{  
        public function area(): Number{  
            return NaN;  
        }  
    }  
}
```



La POO dans ActionScript 3.0

- Créons maintenant la classe Circle

```
package myPackage{  
    public class Circle extends Shape{  
        private var radius:Number = 1;  
        override public function area(): Number{  
            return Math.PI*(radius * radius);  
        }  
    }  
}
```

- Utilisation possible

```
public var forme:Shpae = new Circle;  
public function getArea(forme:Shape):Number{  
    return forme.area();  
}
```




Déboguer de l'ActionScript 3.0

■ Étapes

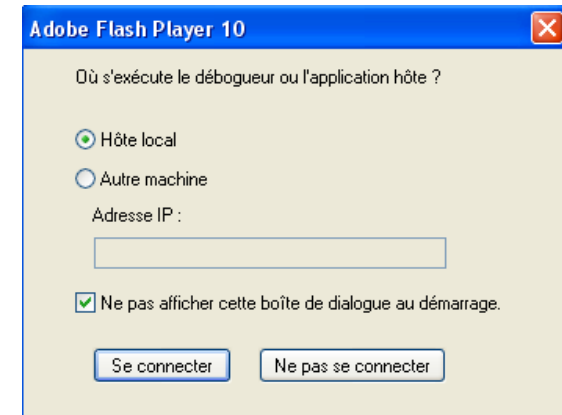
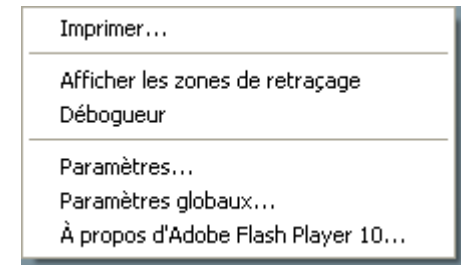
- Étape 0
 - Télécharger et exécuter un débogueur pour votre Flash player
 - ❖ [Download the Windows Flash Player 10.1 Plugin content debugger \(for Netscape-compatible browsers\) \(EXE, 2.69 MB\)](#)
- Étape 1 : création d'un nouveau projet : exemple_debug

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">
  <mx:VBox horizontalAlign="center" verticalAlign="middle"
    horizontalCenter="0" verticalCenter="0">
    <mx:Label id="infoLbl" text="Entrez un Login"/>
    <mx:TextInput id="loginTxtIn"/>
    <mx:Button id="validateBtn" label="valider" />
  </mx:VBox>
</mx:Application>
```
- Pour l'instant, elle ne sert à rien car il n'y pas de code ActionScript, mais assurons nous qu'elle compile en la lançant dans le mode Debug

Déboguer de l'ActionScript 3.0

- Étape 2 : mode Debug
 - Lancer l'application non pas avec Run mais avec le bouton à droite : Debug
 - A première vue, on ne voit rien
 - Cliquer sur le bouton droit au niveau du résultat (navigateur) et un menu Debugger apparaît

- En cliquant sur Debugger, une fenêtre apparaît qui va vous permettre de choisir la machine sur laquelle se trouve l'instance de Flex Builder qui a lancé l'application
- Normalement, vous devriez laisser Localhost et répondre Ok





Déboguer de l'ActionScript 3.0

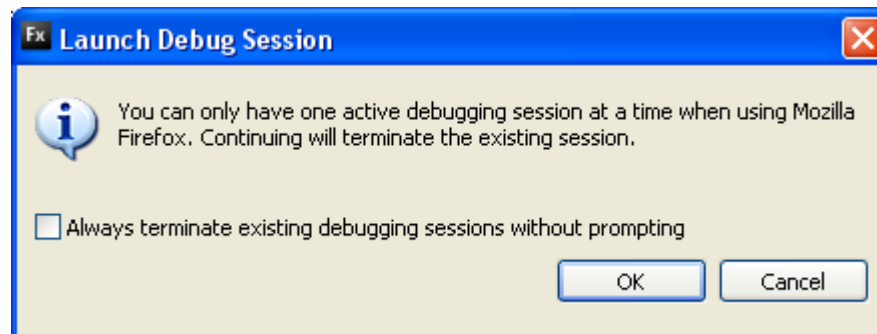
- Étape 3 : Ajoutez un peu d'ActionScript dans votre application

```
<mx:Script>
<![CDATA[
import mx.controls.Alert;
var nameList:Array=["Joe", "William", "Jack", "Averell"];
public function checkName(name:String):Boolean {
    for each(var item:String in nameList) {
        if (name==item) return true;}
    return false;
}
public function validate():void {
    var name:String=loginTxtIn.text;
    var test:Boolean=checkName(name);
    if(test)
        Alert.show ("Bienvenue " + name + ", vous êtes bien sur la
liste!");
    else Alert.show("Désolé " + name + " mais vous n'êtes pas sur la
liste...");
}
]]>
</mx:Script>
```

- Ajouter : click='validate()' dans votre bouton

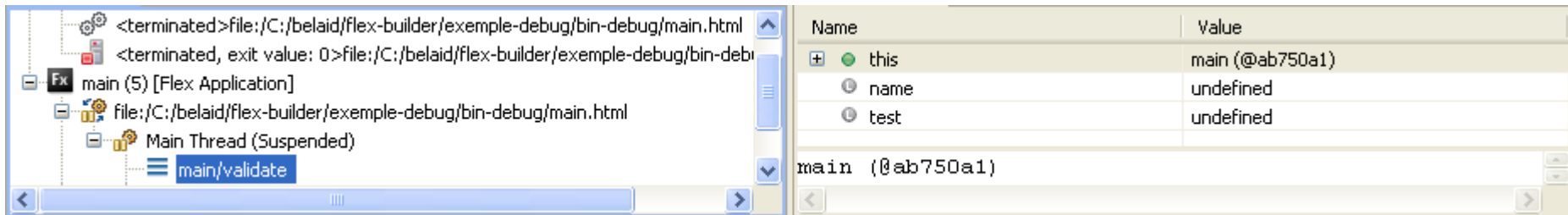
Déboguer de l'ActionScript 3.0

- Étape 4 : point d'arrêt
 - En essayant avec jack en minuscule, un échec va se produire
 - Ajoutons maintenant un point d'arrêt en double cliquant sur la fonction validate, à gauche → un point bleu apparaît
 - Relancez l'application en mode Debug, entrez à nouveau jack en minuscule et constatez le comportement de Flex Builder lorsque vous cliquez sur le bouton Valider
 - Une fenêtre apparaît alors que Flex ouvre la perspective Debug, cliquez sur Ok



Déboguer de l'ActionScript 3.0

- Étape 4 : point d'arrêt
 - Comme vous pouvez le constater, deux nouvelles fenêtres s'ouvrent



The screenshot shows a debugger window with two panes. The left pane displays a call stack with the following entries:

- <terminated>file:/C:/belaid/flex-builder/exemple-debug/bin-debug/main.html
- <terminated, exit value: 0>file:/C:/belaid/flex-builder/exemple-debug/bin-debr
- main (5) [Flex Application]
- file:/C:/belaid/flex-builder/exemple-debug/bin-debug/main.html
- Main Thread (Suspended)
- main/validate

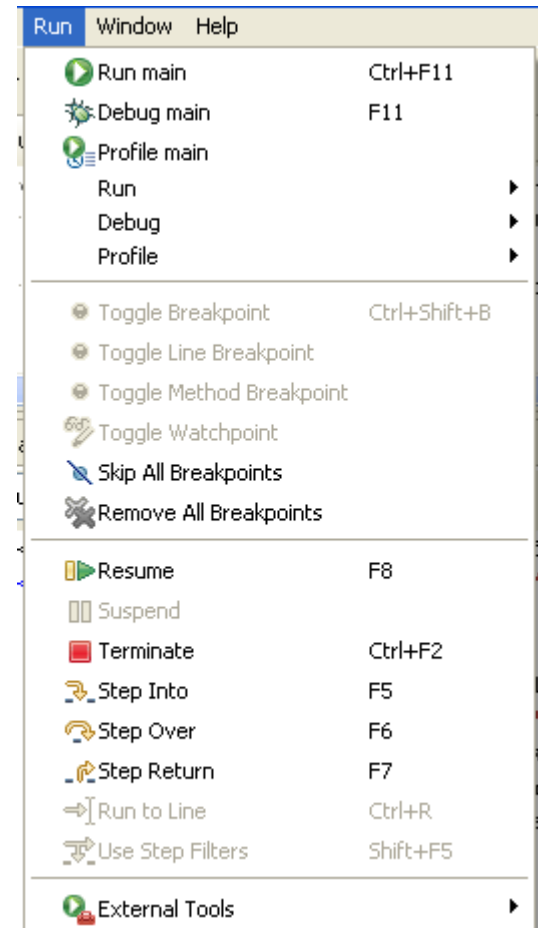
The right pane shows a variable inspector with the following table:

Name	Value	
+	this	main (@ab750a1)
⊖	name	undefined
⊖	test	undefined

Below the table, the variable `main (@ab750a1)` is visible.

Déboguer de l'ActionScript 3.0

- Étape 5 : Mode pas à pas
 - Dans le menu Run, vous trouvez toutes les possibilités pour avancer pas à pas dans le programme
 - Reprendre l'exécution
 - Utiliser Step Into jusqu'à l'appel de la fonction checkName, encore un et vous rentrez dans cette fonction
 - Une fois dans la fonction, vous devriez exécuter la boucle étape par étape à moins de faire un step return pour retourner à la suite de la fonction valide



Déboguer de l'ActionScript 3.0

- Étape 6 : Afficher le contenu d'une variable
 - Passez le curseur au dessus de la variable désirée, sa valeur s'affiche dans la fenêtre correspondante

```
13     public function validate():void {  
14         var name:String=loginTxtIn.text;  
15         var test:Boolean=checkName(name);  
16         if(test)  
17             Alert.show("Bienvenue " + name + ", v  
18         else Alert.show("Désolé " + name + " mais '  
19     }  
20     11>
```

Name	Value
this	main (@eee60a1)
name	"jack"
test	undefined

main (@eee60a1)

Déboguer de l'ActionScript 3.0

- Étape 6 : Afficher le contenu d'une variable
- Ce nouveau panneau permet également de rechercher une variable
- Appuyer en premier lieu sur la croix à droite de la variable this pour visualiser toutes variables du programme
- On effectue maintenant une recherche en appuyant sur Ctrl + f → une nouvelle fenêtre s'affiche
- Taper le nom de la variable recherchée (ici nameList)

