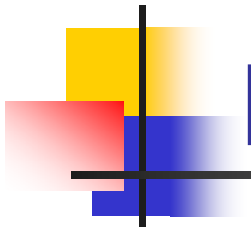
Le contrôle

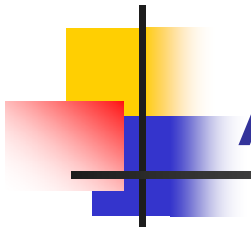
ActionScript

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



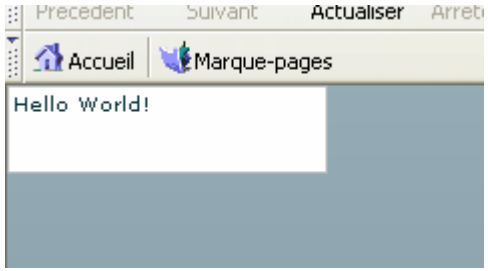
ActionScript 3.0

- 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 (suite 1) : ActionScriptTest0.mxml
  - A l'intérieur de ces balises, on déclare une fonction giveHelloWorld()

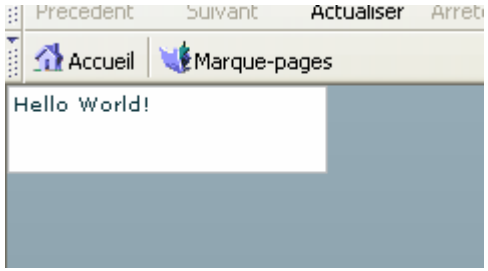



```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
   <mx:Script>
      <![CDATA[
         public function giveHelloWorld():void {
         textAreaId.text="Hello World!";}
          ]]>
   </mx:Script>
   <mx:TextArea id="textAreaId" editable="false"
         text="giveHelloWorld()"/>
</mx:Application>
```

ActionScript 3.0 dans MXML

- Première façon (suite 2) : ActionScriptTest1.mxml
  - Ici, l'appel se fait par creationComplete qui se déclenche automatiquement une fois les composants prêts : appel en cascade

```
<?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/ActionScript Class. Appelez ce fichier HelloWorld
       - ➔ on obtient : HelloWorld.as dans le dossier classes qui est :

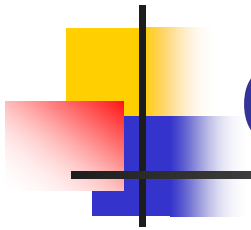
```
package classes{
    public class HelloWorld{
        public static function giveHelloWorld():String{
        return "Hello World!";}
    }
}
```

    4. 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

- Déclaration d'une fonction (listener)

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

  - Cette fonction utilise la classe Alert de Flex qui permet, via la méthode show(), de déclencher un pop-up
- Utilisation de l'événement dans MXML
  - ajouter à l'objet de votre interface MXML, un attribut contenant l'appel : evenementBouton1.mxml

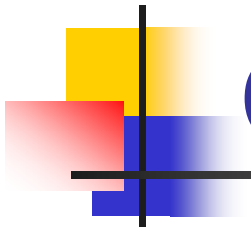
```
<mx:Button id="myButton" label="Cliquer ici"
  click="myListener()"/>
```

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 ou à un identifiant d'un composant MXML

    ```
    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



Gestion des événements

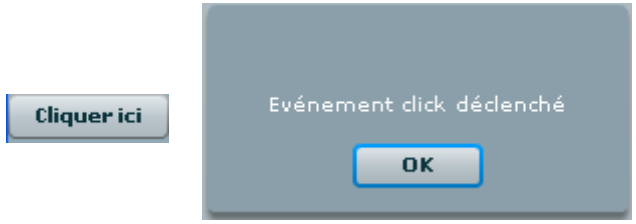
- Exemple complet : eventCode.mxml
  - Il s'agit, en cliquant sur un bouton, de créer un nouveau bouton

```
…<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 : evenementType.mxml
  - On peut accéder aux propriétés de l'événement dans le listener, afin d'en connaître l'origine et le type

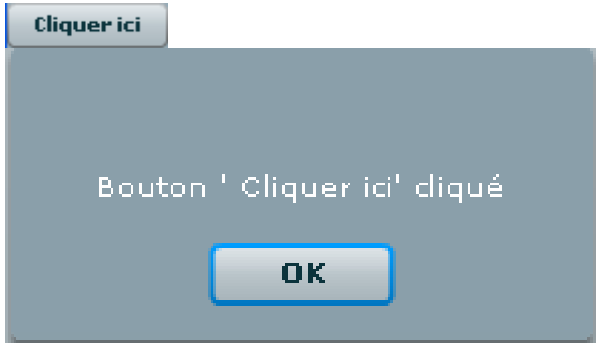
```
function myListener(event:Event):void {
    Alert.show("Evénement " +event.type+" déclenché");
}
```

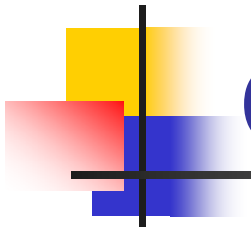


Gestion des événements

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

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



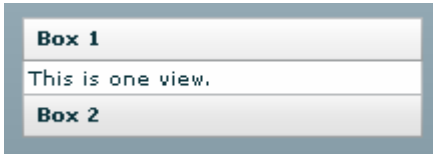


Gestion des événements

- Les événements à connaître
  - Les plus importants sont
    - creationComplete
      - Déclenché lorsqu'un composant et tous ses fils sont crées : on peut alors initialiser les valeurs de certaines variables ou composants
    - Error
      - Permet de récupérer une erreur et de la traiter
    - 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
  - 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



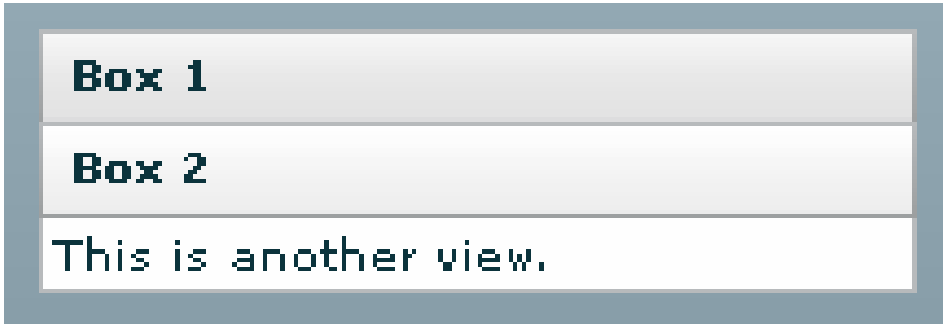
En cliquant sur Box 1

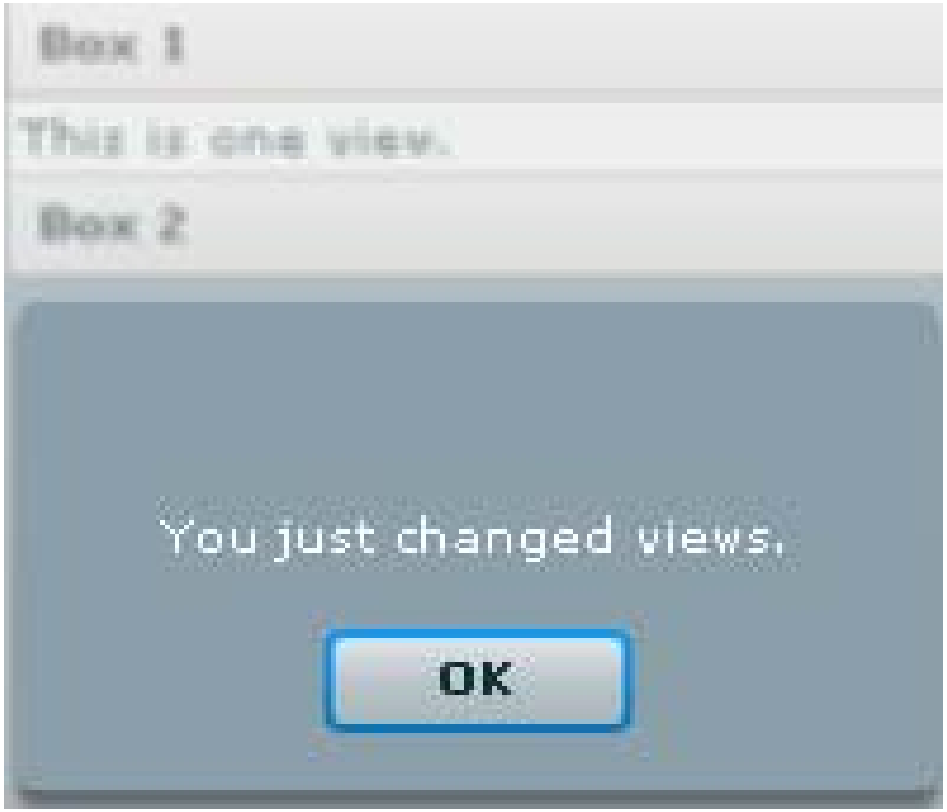
En cliquant sur Box 2

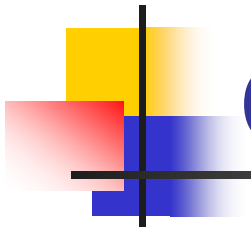
- Exemple pour change : evenementChange.mxml

```
<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







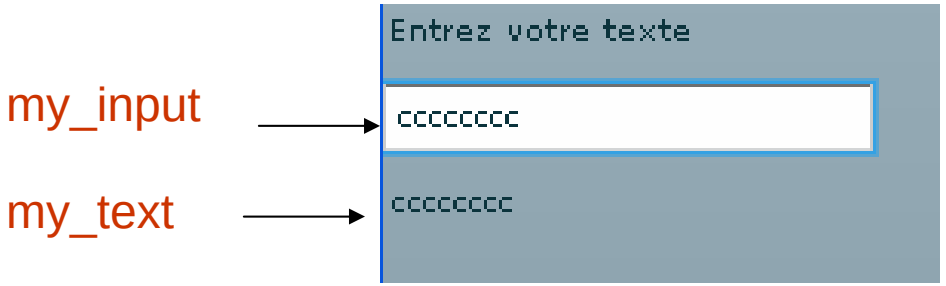
Gestion des événements

- Les événements Clavier
  - Flex propose 2 types d'événements
    - Appui : keyboardEvent.KEY_DOWN
    - Relâche : keyboardEvent.KEY_UP
  - déclenchés dès que l'utilisateur appuie sur une touche clavier (Key_down) ou la relâche (key_up)

Gestion des événements

- Exemple : 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 (KeyUp) sur une touche, le texte écrit est recopié dans la zone "my_text"

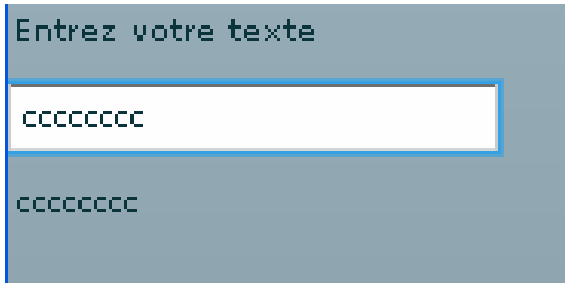
```
<mx:Label  text="Entrez votre texte" id="my_label"/>
<mx:TextInput id="my_input" keyUp="autoFill()"
    y="26" x="0"/>
<mx:Label x="0" y="56" text="le texte écrit"
    id="my_text"/>
```



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>
```

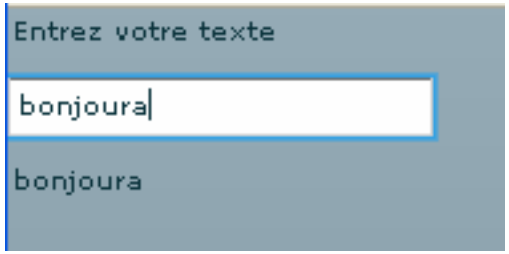


Gestion des événements

- 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

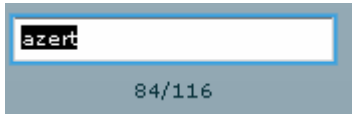
```
function autoFill(event:KeyboardEvent):void {
        if(event.charCode == 97)
        my_text.text=my_input.text
}
```

  - 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

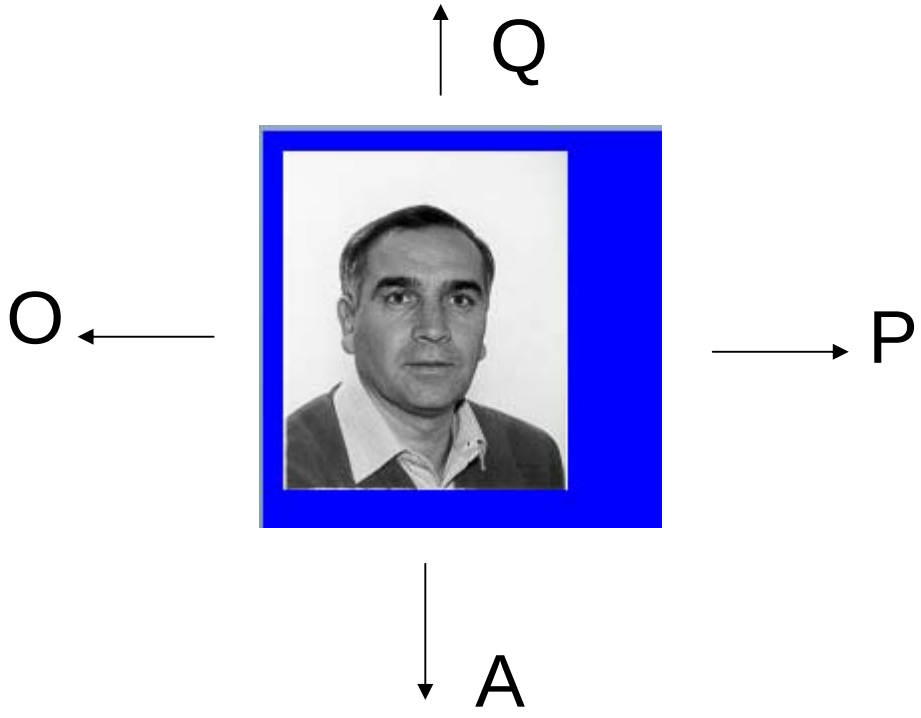


keyCode et charCode de la lettre t

```
… 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>
```

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
altKey	Si la touche Alt est pressée, la valeur de cette propriété est true
ctrlKey	Si la touche ctrl est pressée, la valeur de cette propriété est true
shiftKey	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>
```

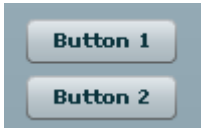
Appuyer sur Ctrl, puis sur le bouton

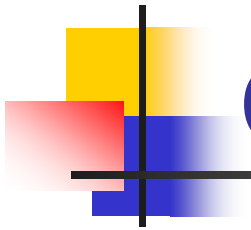
Après saisie et ctrl+clic

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 a 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

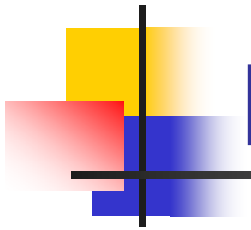
```
<<?xml version="1.0"?>
<!-- events/DispatchEventExampleInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import mx.controls.Alert;

        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>
```

ActionScript 3.0

- Exercice
  - Réaliser une calculette
  - Créer un conteneur avec le montant d'entrée
    - Lire ce montant chiffre par chiffre
    - Cette zone servira également pour l'affiche du résultat
  - Créer un deuxième conteneur pour les boutons des chiffres de 0 à 9
  - Créer un troisième conteneur pour les opérations et le signe =

ActionScript

Éléments du langage



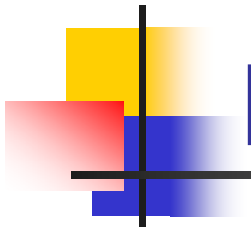
Les packages

- 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 function 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

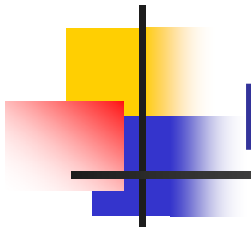


Les packages

- 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.*;



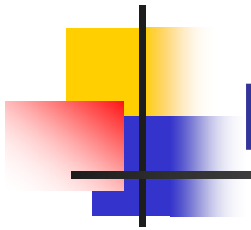
Les types de données et variables

- 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

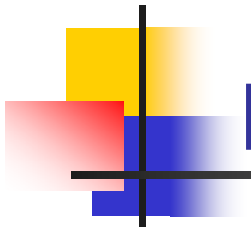
```
var i:int = 42;
if (i is int) {//renverra true
//code exécuté}
if (i is String) {//renverra false
//code non exécuté}
```



Les types de données et variables

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

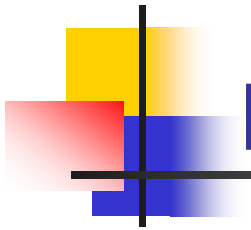
```
var i:int =42;
var res;
res= i as int; //res contiendra 42
res= i as String;  //res contiendra null
```



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

Les types de données et variables

- Les opérateurs

Au niveau du bit		
&	ET au niveau du bit	Convertit `expression1` et `expression2` 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 `expression1` et `shiftCount` en entiers de 32 bits et décale tous les bits de `expression1` vers la gauche en fonction du nombre spécifié par l'entier résultant de la conversion de `shiftCount`.
~	SAUF au niveau du bit	Convertit l'`expression` en un entier signé de 32 bits, puis applique un complément à un au niveau du bit.
|	OU au niveau du bit	Convertit `expression1` et `expression2` en entiers non signés de 32 bits et insère un 1 à chaque position de bit, où les bits correspondant de `expression1` ou `expression2`, sont de 1.
>>	décalage droit au niveau du bit	Convertit `expression` et `shiftCount` en entiers de 32 bits et décale tous les bits de `expression` vers la droite en fonction du nombre spécifié par l'entier résultant de la conversion de `shiftCount`.
>>>	décalage droit non signé au niveau du bit	Identique à l'opérateur de décalage vers la droite au niveau du bit (>>), 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 `expression1` et `expression2` en entiers non signés de 32 bits et insère un 1 à chaque position de bit, où les bits correspondant de `expression1` ou `expression2`, 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 `expression1` la valeur de `expression1 & expression2`.
<<=	décalage gauche au niveau du bit et affectation	Effectue un décalage vers la gauche (<<=) au niveau du bit et stocke ensuite le contenu dans `expression1`.
|=	affectation OU au niveau du bit	Affecte à `expression1` la valeur de `expression1 | expression2`.
>>=	décalage droit au niveau du bit et affectation	Effectue un décalage au niveau du bit vers la droite et stocke le résultat dans `expression`.
>>>=	décalage droit non signé au nivaeu du bit et affectation	Effectue un décalage droite au niveau du bit non signé et stocke le résultat dans `expression`.
^=	affectation XOR au niveau du bit	Affecte à `expression1` la valeur de `expression1 ^ expression2`.
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		
/*..*/	**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.
Comparaison		
==	**égalité**	Vérifie si deux expressions sont égales.
>	**supérieur à**	Compare deux expressions et détermine si `expression1` est supérieur à `expression2`. Dans l'affirmative, le résultat est `true`.
>=	**supérieur ou égal à**	Compare deux expressions et détermines si `expression1` est supérieur ou égal à `expression2` (`true`), ou si `expression1` est inférieur à `expression2` (`false`).
!=	**inégalité**	Recherche l'inverse de l'opérateur d'égalité (==).
<	**inférieur à**	Compare deux expressions et détermine si `expression1` est inférieur ou égal à `expression2`. Dans l'affirmative, le résultat est `true`.
<=	**inférieur ou égal à**	Compare deux expressions et détermine si `expression1` est inférieur ou égal à `expression2`. Dans l'affirmative, le résultat est `true`.
===	**égalité stricte**	Teste l'égalité de deux expressions, mais sans conversion automatique des données.
!==	**inégalité stricte**	Recherche l'inverse exact de l'opérateur d'égalité stricte (===).

Les types de données et variables

- Les opérateurs

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

Les types de données et variables

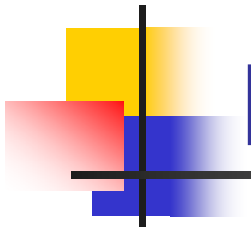
- Les opérateurs

Autre		
[]	accès tableau	Initialise un nouveau tableau, simple ou multidimensionnel, avec les éléments spécifiés (a0, etc.) ou accède aux éléments d'un tableau.
	as	Evalue si une expression spécifiée par le premier opérande est membre du type de données indiqué par le deuxième opérande.
,	virgule	Evalue `expression1`, puis `expression2`, et ainsi de suite.
?:	conditionnel	Evalue `expression1`, et si la valeur de `expression1` est `true`, le résultat reprend la valeur de `expression2`. Sinon, le résultat reprend la valeur de `expression3`.
	delete	Détruit la propriété objet spécifiée par `reference`. Le résultat est `true` si la propriété n'existe pas à la fin de l'opération, et `false` dans tous les autres cas.
.	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.
	in	Evalue si une propriété fait partie d'un objet spécifique.
	instanceof	Evalue si la chaîne de prototype d'une expression comprend l'objet prototype de `function`.
	is	Evalue si un objet est compatible avec un certain type de données ou une certaine classe ou interface.
::	qualificateur de nom	Permet d'identifier l'espace de nom d'une propriété, d'une méthode, d'une propriété ou d'un attribut XML.
	new	Instancie une occurrence de classe.
{}	initialisateur objet	Crée un objet et l'initialise avec les paires de propriétés `name` et `value` spécifiées.
()	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.
/	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.
:	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.
	typeof	Evalue `expression` et renvoie une chaîne spécifiant son type de données.
	void	Evalue une expression, puis supprime sa valeur en renvoyant `undefined`.

Les types de données et variables

- Les opérateurs

Chaîne		
+	**concaténation**	Concatène (combine) des chaînes.
+=	**affectation de concaténation**	Affecte à `expression1` la valeur de `expression1 + expression2`.
"	**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 `expression1`, qui est un objet XMLList, la valeur `expression1 + expression2`.
	delete (XML)	Supprime les éléments ou les attributs XML spécifiés par `reference`.
..	**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.



Eléments du langage

- Les boucles
  - La boucle for

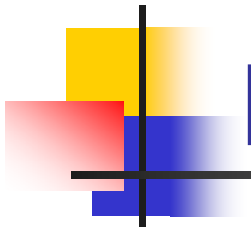
```
var i:int;
for(i=0;i<5;++i)
        //code à exécuter
```

  - 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
```



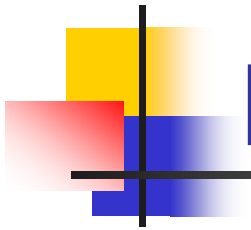
Eléments du langage

- Les boucles
  - La boucle while

```
var i:int;
while (i<5)
        //code à exécuter
++i //incrémentation obligatoire
```

  - La boucle do while

```
var i: int =0;
do {
        //code à exécuter
        ++i; //incrémentation obligatoire
} while (i<5);
```



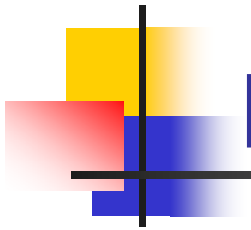
Eléments du langage

- 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

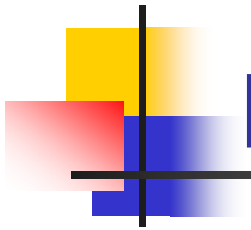
- Les attributs
  - Chaque classe peut avoir des attributs qui la qualifient
  - Exemple :

```
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
            interrnal 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

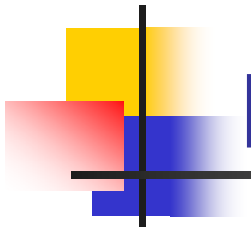
```
<?xml version="1.0" encoding="utf-8"?>
    <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
    layout="vertical" creationComplete="initApp()">
    <mx:Script>
        <![CDATA[
            import myPackage.MyClass
            public var objectTest:MyClass = new MyClass;
            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

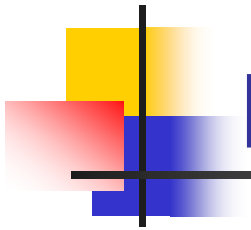
```
package myPackage
{
  public class MyClass
    {
        public var var1:int =1; //visible partout
        //constructeur
        public function MyClass():void
        {
                var1=0;
        }
    }
}
```



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 du 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 débugger 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

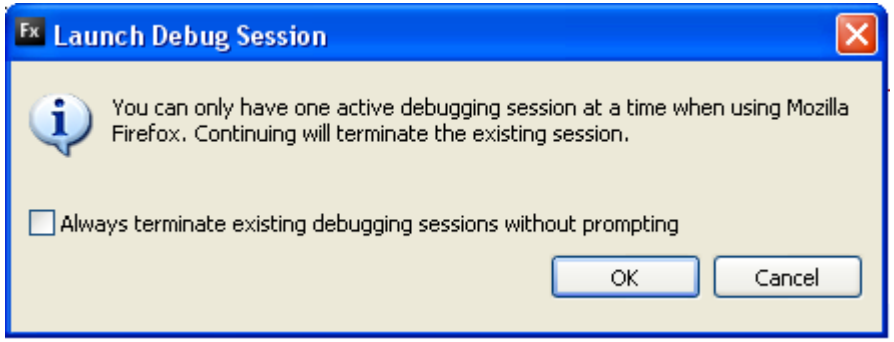
- Étape 3 : Ajoutez un peu d'ActionScript dans votre application

```
<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

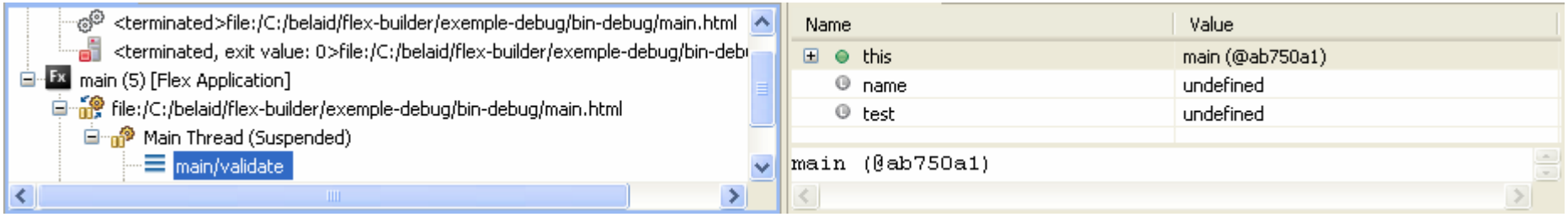
Déboguer de l'ActionScript 3.0

- 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



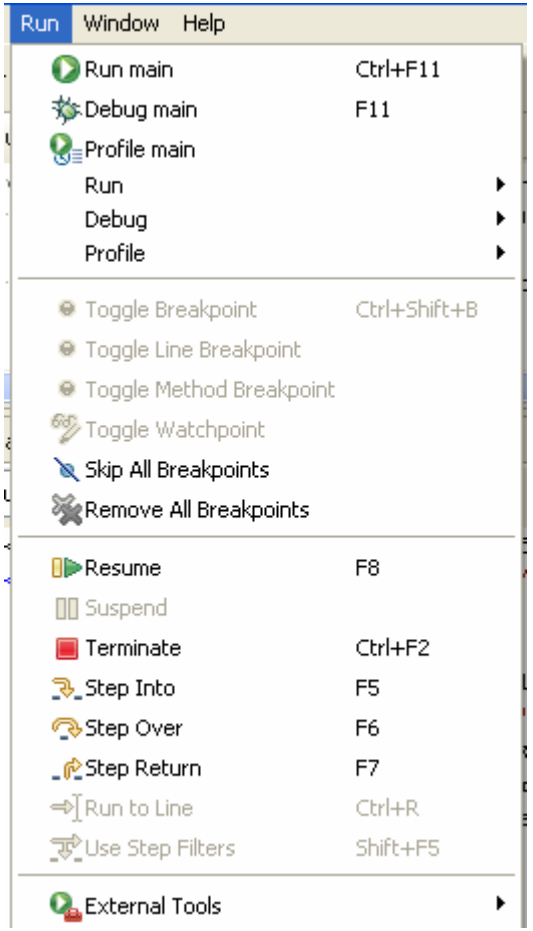
Déboguer de l'ActionScript 3.0

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



Déboguer de l'ActionScript 3.0

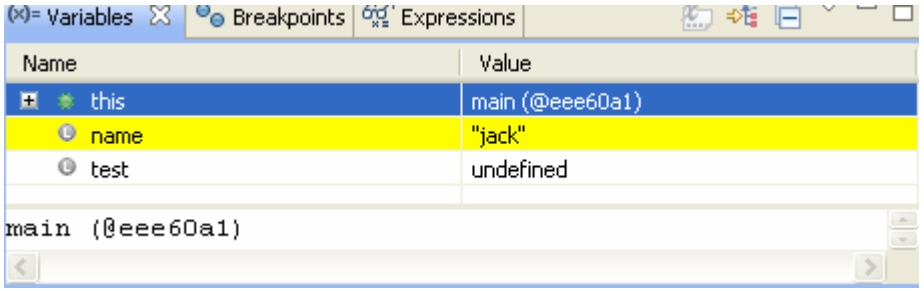
- Étape 5 : Mode pas à pas
  - Dans le menu Run, vous trouvez toutes les possibilités pour avancer pas à pas dans le programme
  - Reprendre l'exécution
  - Utiliser Step Into jusqu'à l'appel de la fonction checkName, encore un et vous rentrez dans cette fonction
  - Une fois dans la fonction, vous devriez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



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      ]]>
```



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)