Flash et ActionScript

ActionScript

Mickaël Martin Nevot

V2.3.0

Flash et ActionScript

I. Présentation du cours
II. ActionScript
III. Adobe Flash

ActionScript / ActionScript 3.0

- Historique :
  - 1998 : Gary Grossman (Flash)
  - 2006 : ActionScript 3.0
- Principales caractéristiques :
  - Extension de fichier : `.as`
  - Interprété **côté client** (parfois côté serveur)
  - Langage de programmation de **script**
  - Standard **ECMAScript** (JavaScript, JSON, etc.)
  - Langage **orienté objet typé**
  - Machine virtuelle 2 (AVM2) pour ActionScript 3.0 :
    - Optimisations de performances

Mise en forme / commentaires

- Mise en forme :
  - **Indentation** à chaque niveau de bloc
  - **Convention de nommage** :
    - `mypackage`
    - `MyClass`
    - `myMethod`
    - `myVar`
    - `MY_CONST`
- **Commentaires** de type C/C++ :

```
// Commentaire ActionScript (une seule ligne).
/* Autre commentaire ActionScript (une ou plusieurs lignes). */
/*
    Commentaire ActionScript
    (sur plusieurs lignes).
*/
```

Variable et constante

- Définition de type :
  - **Déclaration explicite** du type d'une variable
  - Vérification des types à la **compilation** et à l'**exécution**
- **Variable** : `var`

```
var a:Number;
var name:String = "John";
```

- **Constante** (variable non modifiable) : `const`

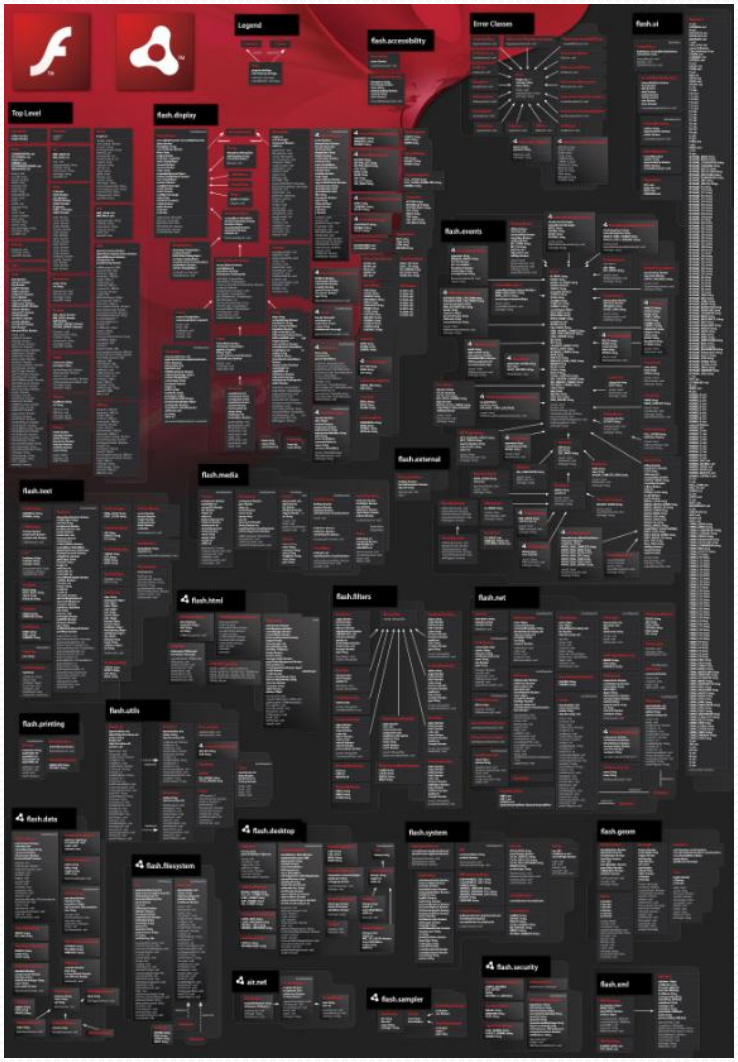
```
const B:Number = 3;
B = 2;          // Erreur !
const B:Number; // Erreur !
C = 4;          // Erreur !
```

- **Portée** : lexicale
  - Globale : durée de vie dans tout le script
  - Locale : durée de vie limitée au bloc de sa déclaration

Déconseillé !

Types

- Primitifs :
  - `Boolean`
  - `int`
  - `uint`
  - `Number`
  - `String`
  - `*`
  - `Null`
  - `void`
- Complexes :
  - `Object`
  - `Array`
  - `Error`
  - `Date`
  - `Math`
  - `RegExp`
  - `Function`
  - Etc.



En ligne sur le site de l'enseignant

Types primitifs

- `Boolean` : peut prendre comme valeur `true` ou `false`
- Entiers :
  - `int` : entier
  - `uint` : entier non signé
- `Number` : nombre décimal
  - Valeurs spéciales :
    - « Pas un nombre » (*not a number*) : `NaN`
    - Infinité : `-Infinity`, `Infinity`

```
var a:Number = -1 / 0; // a = -Infinity.
var b:Number = 1 / 0;  // b = Infinity.
```

- `String` : séquence de caractères spécifiée par `""` ou `''`

Tableau

Un tableau peut être indicé ou associatif et peut être multidimensionnel

- Construction :

```
var arr1:Array = new Array(); // Créer un tableau vide.
var arr2:Array = new Array("monday", "tuesday", "wednesday");
arr1[0] = "dog";
```

- Méthodes :
  - `length:uint` : taille du tableau (nombre d'éléments)
  - `push(... args):uint` : ajoute un/plusieurs éléments à la fin
  - `pop():*` : retire et renvoie le dernier élément
  - `concat(... args)` : renvoie un tableau avec les éléments ajoutés
  - `slice(start, end):Array` : renvoie un sous-tableau
  - `sort(... args):uint` : tri (fonction de comparaison optionnelle)
  - `unshift(... args):uint` : insère des éléments en tête

Fonction

```
function foo(a, b, c = 2):int {
    // Rappel : a, b et c sont des PARAMÈTRES.
    // = 2 : valeur par défaut (uniquement de type scalaire).
    // Un paramètre par défaut est optionnel.
    ++a;
    --b;
    // Termine la fonction en retournant la valeur calculée.
    return (a * b) / c;
    // Le code éventuellement placé ici n'est pas exécuté.
}

var a:uint = 5;
var b:int = foo(3, a);  // Rappel : 3 et a sont des ARGUMENTS.
// a = 5.
// b = 8.
```

Classe

- Un fichier source contient une **classe**
- Une classe contient des **attributs** et des **méthodes**
- Objet courant : `this`
- Définition :

```
class MyClass { // Déclaration de classe.
    var att1:int; // Déclaration d'attribut.
    …
    function meth1(i:int):String { //Déclaration de méthode.
        this.att1 = i;
        …
    }
}
```

- Utilisation :

```
var obj:MyClass = new MyClass();
obj.meth1(5);
```

Constructeur

- **Même nom que la classe**
- Méthode spéciale appelée à l'instanciation d'un objet pour initialiser son état
- **Pas de valeur de retour** : `void`
- Si aucun constructeur n'existe, un **constructeur par défaut** (sans paramètre et qui ne fait rien) est défini
- Mot clé `new` (création et allocation mémoire)

```
class MyClass { // Déclaration de classe.
    function MyClass(p:int):void { … }
…
// Déclaration.
var obj:MyClass;
// Création et allocation mémoire : instanciation.
obj = new MyClass(5);
```

Paquetage

- **Groupe de classes** associé à une fonctionnalité
- À chaque paquetage correspond un **répertoire**
- Un paquetage peut contenir un autre paquetage
- Déclaration (première instruction d'une classe) : `package`

```
package package1.package2;
// MyClass appartient au paquetage package2 qui appartient lui-même à package1.
class MyClass {
    …
}
```

- Utilisation : `import`

```
// Importe la classe MyClass.
import package1.package2.MyClass;
// Importe toutes les éléments du paquetage display qui appartient lui-même à flash.
import flash.display.*;
```

Mot clefs spéciaux

- Types primitifs :
  - `*` : non typé (peut prendre la valeur : `undefined`) :
  - `void` : n'existe pas ou non typé (seule valeur : `undefined`)
  - `Null` : typé mais non initialisé (seule valeur : `null`)
- Paramètres :
  - ... : nombre indéterminé de paramètres

Fonction de traces : `trace()`

Opérateurs

- Unaires :
  - Signe : +, –
  - Négation : !
  - Incrémentation / décrémentation (pré et post) : ++, ––
  - Type : `typeof`

```
var a:Number = -0.5;
++a;
--a;
var c:Boolean = !b;
var d:String = typeof c;
```

Opérateurs

- Binaires :
  - Arithmétiques : +, –, *, /, %
  - Concaténation de chaîne : +
  - Affectation : =, +=, –=, *=, /=, %=, etc.
  - Comparaison : ==, >, <, >=, <=, !=, ===, !==, etc.
  - Logiques : &&, ||, etc.
  - Bit à bit : &, |, ^, etc.
  - Vérification de type : `is`, `as`

Comparaison sur les valeurs et les types

Ou exclusif

```
var a:uint = 5;
var b:uint = 10;
var c:String = "a : " + a;
var d:Boolean = (a == b) && ((b % a != 0) || (b < 20));
trace(c is String); // true.
trace(c as Number);
```

Structures conditionnelles

Structure conditionnelle :

```
if (a == 0) {
    ++a;
} else if (a == 1) {
    --a;
} else {
    a = 0;
}
```

Branchement conditionnel :

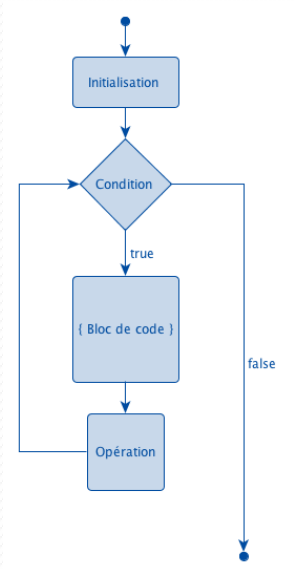
```
switch(i) {
  case "jambon":
       trace("salé");
       break;
  case "tarte":
  case "bonbon":
  case "biscuit":
      trace("sucré");
      break;
  default:
      trace("autre");
  }
```

Identique à JavaScript

Boucles

- Boucle `for` :

```
var res:String = "";
for (var i:uint = 0 ; i < 10 ; ++i) {
   res += i + ", ";
}
trace(res);
// 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,.
```



- Boucle `while` :

De 0 à n fois

```
var i:uint = 0;
while(i < 0) {
   ++i;
}
trace(i);
// 0.
```

- Boucle `do ... while` :

De 1 à n fois

```
var i:uint = 0;
do {
   ++i;
} while (i < 0);
trace(i);
// 1.
```

Boucles

Boucle for … in :

- Tableau :

```
var res:String = "";
var arr:Array = [1, 2, 3, 4, 5];
for (var val:uint in arr) {
    res += (arr[val] * 2) + ", ";
}
trace(res);
// 2, 4, 6, 8, 10, .
```

- Objet :

```
var obj:MyClass = new MyClass();
for (var i:String in obj) {
    trace(i + " : " + obj[i]);
}
// x : 20.
// y : 30.
```

Boucle for each … in :

- Tableau :

```
var res:String = "";
var arr:Array = [1, 2, 3, 4, 5];
for each (var val:uint in arr) {
    res += (val * 2) + ", ";
}
trace(res);
// 2, 4, 6, 8, 10, .
```

- Objet :

```
var obj:MyClass = new MyClass();
for each (var i:String in obj) {
    trace(i);
}
// 20.
// 30.
```

Break/continue

- Sortir d'une boucle ou d'un `switch ... case : break`

```
while(1) {
   if (i == 10) {
      break;
   }
   ++i;
}
// Exemple de break dans un switch sur la diapositive de structures conditionnelles.
```

- « Sauter » à l'itération suivante d'une boucle : `continue`

```
var a:uint = 0;
for  (var i:uint = 0 ; i < 10 ; ++i) {
   if (i % 2 == 0) {
      continue;
   }
   a += i;
}
// ?
```

Instruction

- Se termine par un `;`
- Type d'instruction :
  - Déclaration : `var a:int;`
  - Affectation : `a = new MyClass();`
  - Appel de fonction : `myFunction();`
  - Instruction conditionnelle : `if (a == b) … ;`
  - Instruction vide : `;`
  - Bloc (d'instruction) :

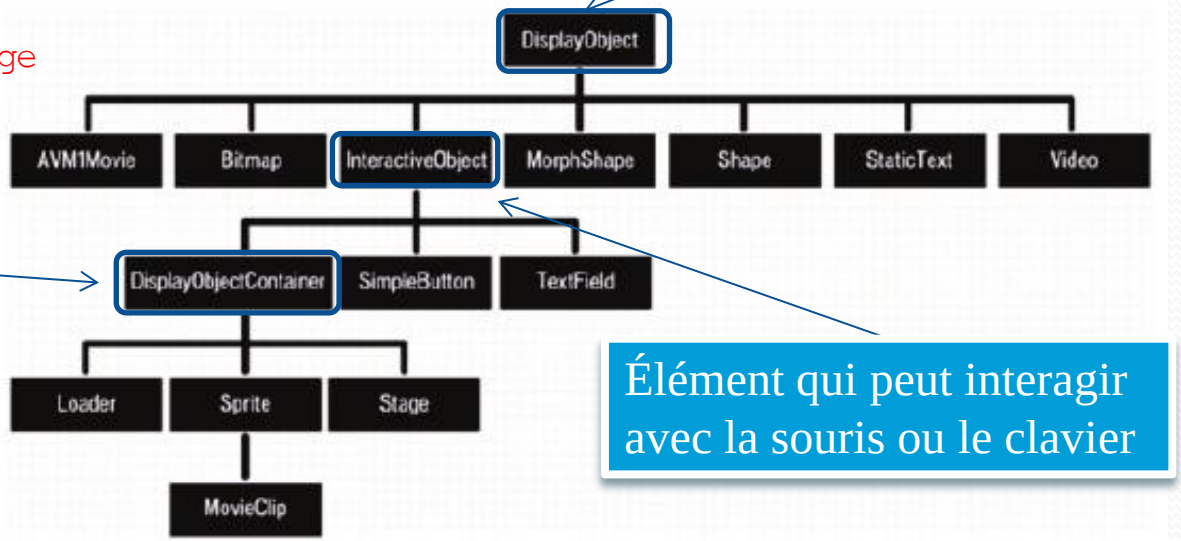
```
{
    instruction_1
    instruction_2
    …
}
```

Liste d'affichage

- Paquetages de classes graphiques :
  - `flash.display`
  - `flash.media`
  - `flash.text`
- Instanciation des classes graphiques :

```
var a:TextField = new TextField();
a.text = "Hello";
// ajout à la liste d'affichage
addChild(a);
```

Élément graphique de base



Élément pouvant contenir d'autres objets graphiques

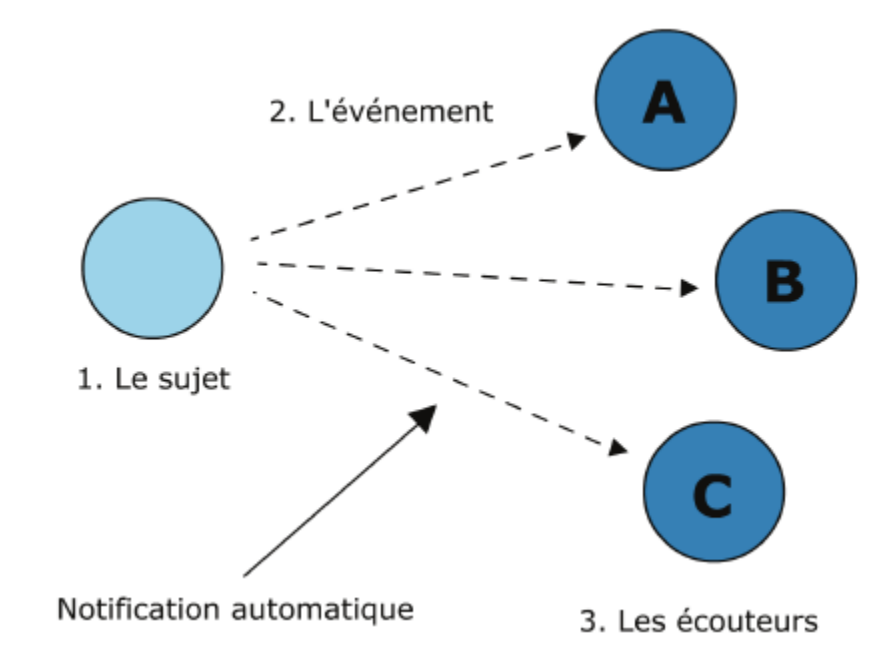
Élément qui peut interagir avec la souris ou le clavier

Les évènements

- **Sujet** : source de l'évènement (bouton, clip, etc.)
- **Évènement** : changement d'état du sujet
- **Écouteur** : méthode (ou fonction) qui écoute les changements d'état du sujet (appelé automatiquement)
- Utilisation :

```
var a:Sprite = new Sprite();
a.addEventListener(Event.ADDED, added);
function added(e:Event):void {
    …
}
```

La propagation évènementielle est la même que celle du DOM



Aller plus loin

- Encapsulation
- Héritage
- Ramasse-miettes
- Date
- Expression régulière
- Exceptions
- Utilisation de XML
- Librairies 3D
- Flex et MXML
- Air

Liens

- Documents électroniques :
  - Manuels :
    - http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3
    - http://help.adobe.com/fr_FR/ActionScript/3.0_ProgrammingAS3
- Documents classiques :
  - Gary Rosenzweig. *ActionScript 3.0 pour les jeux*.
  - Colin Moock. *Le meilleur d'ActionScript 3*.
  - Thibault Imbert. *Pratique d'ActionScript 3*.
  - Vianney Baron, Jessy Bernal, Adrien Montoille, Edouard Ruiz, Nicolas Yuen. *Flex & Air*.
  - Raphaël Mamède. *Gestion de projet d'un jeu en Flash*.

Crédits

Auteur

Mickaël Martin Nevot
mmartin.nevot@gmail.com

Carte de visite électronique

Relecteur

Cours en ligne sur : mickael.martin.nevot.free.fr