11

Classe du document

Intérêts

Depuis l'introduction d'ActionScript 2, les développeurs avaient pour habitude de créer une classe servant de point d'entrée à leur application. Celle-ci instanciait d'autres objets ayant généralement besoin d'accéder au scénario principal. Il fallait donc passer une référence au scénario principal.

Il n'était donc pas rare de trouver sur une image du scénario de l'application le code suivant :

```
var monApplication:Application = new Application ( this );
```

L'application pouvait aussi être initialisée à l'aide d'une simple méthode statique :

```
Application.initialise ( this );
```

D'autres développeurs utilisaient une autre technique, consistant à changer le contexte d'exécution de la classe principale avec le code suivant :

```
this.__proto__= Application.prototype;
Application['apply']( this , null );
```

En utilisant le mot-clé `this` au sein de l'instance de la classe `Application` nous faisions alors référence au scénario principal.

ActionScript 3 intègre une nouvelle fonctionnalité appelée *Classe du document* permettant d'éviter d'avoir recours à ces différentes astuces.

La classe MainTimeline

Comme nous l'avons vu lors du chapitre 4 intitulé *La liste d'affichage* le lecteur Flash est constitué d'un objet `Stage` situé au sommet de la liste d'affichage. Lorsqu'un SWF est lu dans le lecteur, celui-ci ajoute le scénario principal du SWF chargé en tant qu'enfant de l'objet `Stage` :

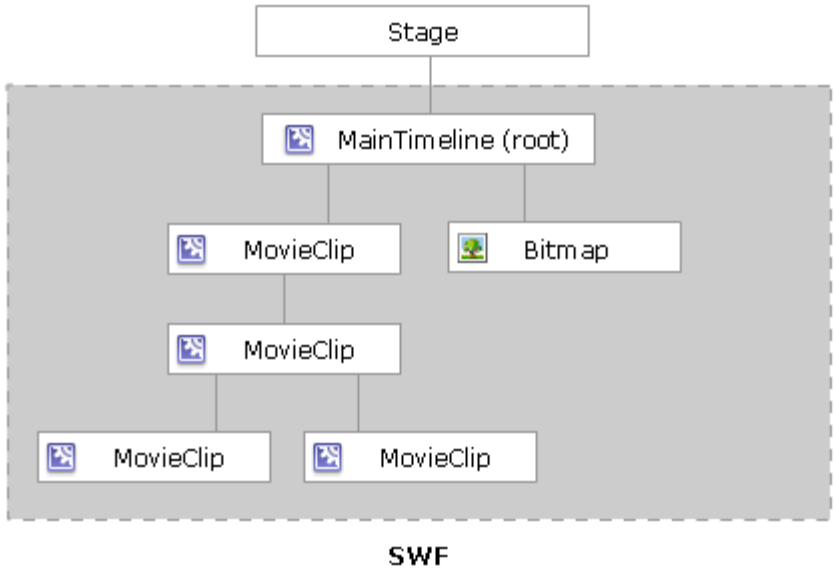

Figure 11-1. Classe du document `MainTimeline`.

Par défaut le scénario d'un SWF est représenté par la classe `MainTimeline`. Nous pouvons nous en rendre compte très facilement. Dans un nouveau document Flash CS3, testez le code suivant sur le scénario principal :

```
// affiche : [object MainTimeline]
trace( this );
```

Nous allons voir que nous ne sommes pas limités à cette classe.

A retenir

- Par défaut le scénario principal est représenté par la classe `MainTimeline`.

Classe du document

Flash CS3 intègre une nouvelle fonctionnalité permettant d'utiliser une instance de sous classe graphique comme scénario principal. Attention, celle-ci doit hériter obligatoirement d'une classe graphique telle `flash.display.Sprite` ou `flash.display.MovieClip`.

> Il est techniquement possible d'utiliser une sous classe de `flash.display.Shape` mais il serait impossible d'ajouter un objet graphique sur le scénario principal, la classe `Shape` n'étant pas une sous classe de `flash.display.DisplayObjectContainer`.

Si nous utilisons comme classe du document une sous-classe graphique nommée `Application`, Flash ajoute aussitôt une instance de celle-ci comme scénario principal.

La figure 11-2 illustre l'idée :

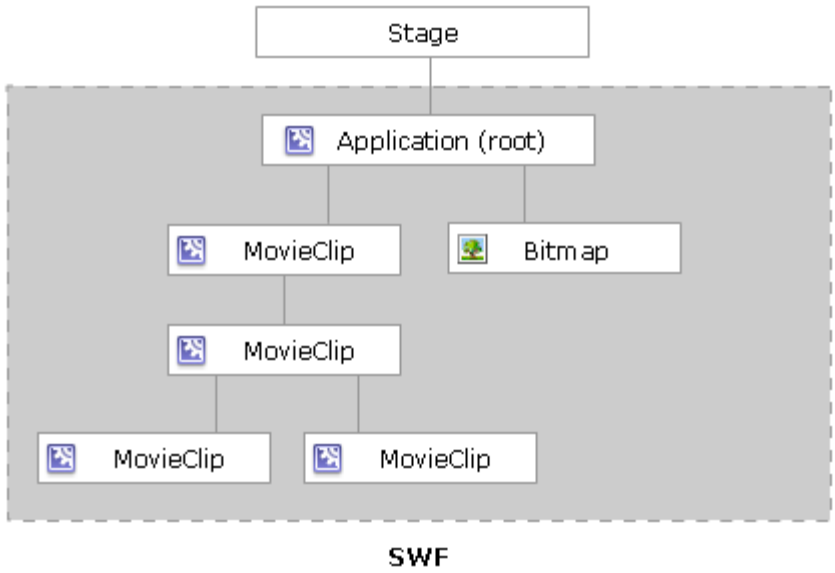

Figure 11-2. Classe du document `Application`.

Si nous ne spécifions aucune classe, le lecteur crée une instance de `MainTimeline`.

Nous allons créer un nouveau document Flash CS3 puis créer a coté un répertoire `org` dans lequel nous créons un répertoire `document`.

Au sein du répertoire `document` nous définissons une classe `Application` dont voici le code :

```
package org.bytearray.document

{

    import flash.display.Sprite;

    public class Application extends Sprite

    {

         public function Application ()

         {

              // affiche : [object Application]
              trace( this );

         }

    }

}
```

Grâce à la notion de classe du document, le mot-clé `this` fait ici référence à l'instance de la classe `Application`, donc au scénario principal.

Afin de lier cette classe du document à notre document Flash nous utilisons le champ *Classe du document* du panneau *Inspecteur de propriétés* illustré en figure 11-3 :

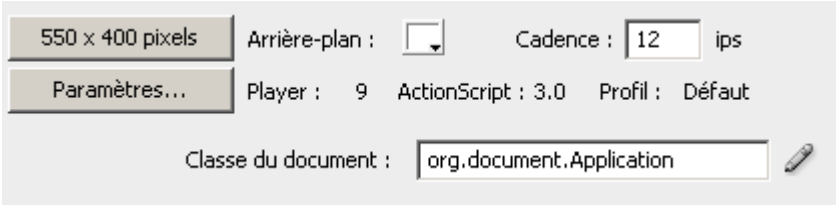

Figure 11-3. Champ Classe du document.

Nous spécifions le paquetage complet de la classe `Application`.

Il est intéressant de noter que l'instanciation de la classe `Application` est transparente. Lorsque l'animation est chargée, le lecteur Flash crée automatiquement une instance de la classe `Application` et l'ajoute en tant qu'enfant de l'objet `Stage` :

```
package org.bytearray.document

{
```

```
    import flash.display.Sprite;

    public class Application extends Sprite

    {

        public function Application ()

        {

            // affiche : [object Stage]
            trace( parent );

            // affiche : [object Stage]
            trace( stage );

            // affiche : [object Application]
            trace( this );


        }

    }
}
```

Si nous écoutons l'événement `Event.ADDED_TO_STAGE` nous remarquons que celui-ci est bien diffusé :

```
package org.bytearray.document

{

    import flash.display.Sprite;
    import flash.events.Event;

    public class Application extends Sprite

    {

        public function Application ()

        {

            // écoute de l'événement Event.ADDED_TO_STAGE
            // est diffusé automatiquement car le lecteur
            // ajoute à la liste d'affichage une instance
            // de la classe Application
            addEventListener ( Event.ADDED_TO_STAGE, ajoutAffichage );

        }

        private function ajoutAffichage ( pEvt:Event ):void

        {

            // affiche : [Event type="addedToStage" bubbles=true
cancelable=false eventPhase=2]
            trace( pEvt );

        }
```

```
    }
}
```

Il faut donc considérer la classe du document comme le point d'entrée de notre projet. Nous avons dans notre exemple utilisé une sous-classe de `Sprite` comme classe du document. Nous allons voir dans quelle mesure son utilisation est limitée dans ce contexte.

A retenir

- Pour affecter une classe du document, nous utilisons le champ *Classe du document* de l'inspecteur de propriétés.
- L'instanciation de classe du document est transparente, le lecteur s'en charge automatiquement.
- L'instance de la classe du document devient le scénario principal.

Limitations de la classe Sprite

Rappelez-vous, lors du chapitre 4 intitulé *La liste d'affichage* nous avons découvert que la classe `Sprite` ne disposait pas de scénario.

Ainsi, lorsque la classe du document est une sous-classe de `Sprite` il est impossible de faire appel aux différentes méthodes de manipulation du scénario telles `gotoAndPlay`, `gotoAndStop` ou autres.

Le code suivant ne peut être compilé :

```
package org.bytearray.document
{
    import flash.display.Sprite;
    public class Application extends Sprite
    {
         public function Application ()
         {
              gotoAndStop ( "intro" );
         }
    }
}
```

L'erreur suivante est générée à la compilation :

```
1180: Appel à une méthode qui ne semble pas définie, gotoAndStop.
```

Nous utiliserons donc une sous-classe de `Sprite` comme classe du document lorsque nous n'aurons pas besoin de scénario.

Nous pourrions penser qu'il s'agit de la seule limitation de la classe `Sprite`, mais il est aussi impossible d'ajouter du code sur les images du scénario. Les lignes suivantes sont posées sur l'image 1 de notre animation :

```
var monClip:MovieClip = new MovieClip();
```

L'erreur suivante est générée :

```
1180: Appel à une méthode qui ne semble pas définie, addFrameScript.
```

Une simple ligne de code commentée empêche la compilation :

```
//var monClip:MovieClip = new MovieClip();
```

La puissance de la classe du document réside dans l'interaction entre le code des images et les fonctionnalités apportées par la classe du document. L'utilisation d'une sous-classe de `Sprite` comme classe du document est donc généralement déconseillée.

En étendant la classe `MovieClip`, nous pouvons ajouter des actions d'images et manipuler le scénario.

Nous modifions la classe `Application` afin d'obtenir le code suivant :

```
package org.bytearray.document
{
    import flash.display.MovieClip;
    public class Application extends MovieClip
    {
         public function Application ()
         {


         }
    }
}
```

Dans le cas d'une application nécessitant un minimum d'animations et de code sur les images nous préférerons étendre la classe `MovieClip`.

Actions d'images

Toute action d'image s'exécute dans le contexte de l'instance de la classe du document. Prenons un exemple simple, nous ajoutons une méthode `afficheMenu` au sein de la classe `Application` :

```
package org.bytearray.document

{

    import flash.display.MovieClip;

    public class Application extends MovieClip

    {

          public function Application ()

          {


          }

          // méthode de création du menu
          public function afficheMenu ( ):void

          {

               trace("création du menu");

          }

    }

}
```

La méthode `afficheMenu` devient donc une méthode du scénario principal, sur n'importe quelle image nous pouvons l'exécuter en plaçant le code suivant :

```
// stop le scénario principal
stop();

// appelle la méthode afficheMenu de la classe du document
// affiche : affiche menu
afficheMenu();
```

Nous stoppons le scénario principal puis appelons la méthode `afficheMenu` définie au sein de la classe du document.

Cette capacité à pouvoir déclencher différentes fonctionnalités de la classe du document depuis les images rend le développement plus souple. Lorsqu'un simple effet ou un ensemble de mécanismes complexes doivent être déclenchés nous l'appelons directement depuis le scénario.

A retenir

- L'utilisation d'une sous classe de `Sprite` comme classe du document est généralement déconseillée.
- Nous préférons généralement utiliser une sous-classe de `MovieClip`.
- La puissance réside dans l'interaction entre les actions d'images et les fonctionnalités de la classe du document.

Déclaration automatique des occurrences

Afin d'accéder aux objets graphiques depuis la classe du document nous utilisons deux techniques.

Comme nous l'avons vu au cours du chapitre 9 intitulé *Etendre les classes natives*, l'option *Déclarer automatiquement les occurrences de scène* est activée par défaut dans Flash CS3.

Dans l'exemple suivant nous créons un symbole clip auquel nous associons une classe `Personnage`, puis nous posons une occurrence sur la scène, nommée `monPersonnage` :

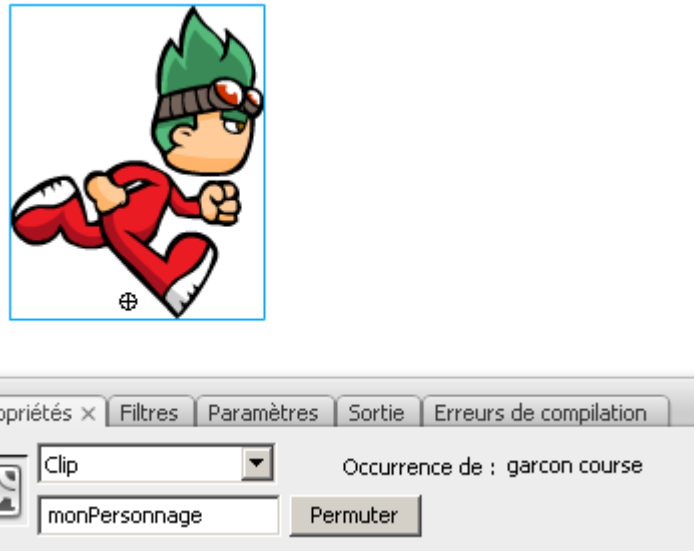

Figure 11-4. Occurrence `monPersonnage`.

A la compilation, une propriété `monPersonnage` est automatiquement ajoutée et référence l'occurrence :

```
package org.bytearray.document

{

    import flash.display.MovieClip;

    public class Application extends MovieClip
```

```
    {
        public function Application ()
        {
            // affiche : [object Personnage]
            trace( monPersonnage );
        }
    }
}
```

On pourrait également recourir à la méthode `getChildByName` :

```
package org.bytearray.document
{
    import flash.display.MovieClip;
    public class Application extends MovieClip
    {
        public function Application ()
        {
            // suppression du symbole monPersonnage
            removeChild ( getChildByName ( "monPersonnage" ) );
        }
    }
}
```

Le code suivant supprime l'occurrence `monPersonnage` posée sur le scénario principal :

```
package org.bytearray.document
{
    import flash.display.MovieClip;
    public class Application extends MovieClip
    {
        public function Application ()
        {
            // suppression du symbole monPersonnage
            removeChild ( monPersonnage );
        }
```

```
        }
    }
```

Lorsque l'option *Déclarer automatiquement les occurrences de scène* est activée, le compilateur ajoute automatiquement au sein de la classe conteneur des propriétés pointant vers les occurrences crées depuis l'environnement auteur.

Cette déclaration automatique des occurrences empêche le développeur de voir quels sont les objets utilisés au sein de la classe et ne facilite pas le déboguage de l'application. C'est pour cette raison que nous préférons généralement désactiver la déclaration automatique dans un projet géré par des développeurs seulement.

A l'inverse, dans un contexte d'interactions développeurs-designers le graphiste peut être amené à poser un nouvel objet sur la scène et lui donner un nom d'occurrence afin d'enrichir graphiquement l'application.

Si l'option *Déclarer automatiquement les occurrences de scène* est désactivée, le graphiste, ne pourra plus compiler le projet sans que le développeur n'ajoute une propriété au sein de la classe du document correspondant à l'occurrence ajoutée. Cela risque donc de gêner le flux de travail au sein d'une équipe.

Déclaration manuelle des occurrences

En désactivant la déclaration automatique des occurrences, nous devons définir manuellement au sein de la classe des propriétés du même nom que les occurrences :

```
package org.bytearray.document

{

    import flash.display.MovieClip;

    public class Application extends MovieClip

    {

        // propriété liée à l'occurrence
        public var monPersonnage:MovieClip

        public function Application ()

        {

            // suppression du symbole monPersonnage
            removeChild ( monPersonnage );

        }
```

```
    }

}
```

La propriété définie manuellement doit obligatoirement être publique. Si nous la rendons privée à l'aide de l'attribut `private` le compilateur génère une erreur :

```
ReferenceError: Error #1056: Impossible de créer la propriété monPersonnage sur
org.bytearray.document.Application.
```

Lorsque le compilateur tente d'affecter la propriété `monPersonnage` afin de la faire pointer vers l'occurrence, celle-ci est privée et n'est donc pas accessible depuis l'extérieur de la classe. Ainsi, les propriétés liées aux occurrences doivent **toujours** être publique.

Si nous définissons la propriété et que l'option *Déclarer les occurrences de scène* est activée, une erreur à la compilation s'affiche :

```
1151: Conflit dans la définition monPersonnage dans l'espace de nom internal.
```

Le compilateur tente de définir une propriété pointant vers l'occurrence posée sur la scène, mais celui ci rencontre une propriété du même nom déjà définie. Un conflit de propriétés est généré.

A retenir

- De manière générale, il est préférable de désactiver la déclaration automatique des occurrences de scène.
- Dans un contexte d'interactions designers-développeurs, il est préférable d'activer la déclaration automatique des occurrences.
- Les propriétés liées aux occurrences doivent toujours être publique.

Ajouter des fonctionnalités

Il faut bien comprendre que lorsqu'une classe du document est définie, son instance représente le scénario principal.

Nous avons découvert lors du chapitre 4 intitulé *La liste d'affichage* différents comportements liés à l'accès aux objets graphiques. Souvenez vous en ActionScript 2 nous pouvions écrire le code suivant :

```
gotoAndStop (20);
monPersonnage._alpha = 100;
```

Même si l'occurrence `monPersonnage` n'était présente qu'à l'image 20, nous pouvions depuis n'importe quelle image y accéder. Cela n'est plus vrai en ActionScript 3.

Afin d'accéder correctement à l'objet `monPersonnage` nous pouvons utiliser l'événement `Event.RENDER`. Cet événement est déclenché lorsque le lecteur a fini de rendre les données vectorielles. Afin qu'il soit diffusé nous devons appeler la méthode `invalidate` de la classe `Stage`.

Ainsi, nous pourrions accéder à l'objet `monPersonnage` avec le code suivant :

```
// écoute de l'événement Event.RENDER
addEventListener ( Event.RENDER, miseAJour );

function miseAJour ( pEvt:Event ):void

{

    monPersonnage.alpha = 1;

    // supprime l'écoute
    removeEventListener( Event.RENDER, miseAJour );

}

// déplace le tête de lecture
gotoAndStop (20);

// force le rafraîchissement
stage.invalidate();
```

La fonction `miseAJour` est déclenchée lorsque le lecteur a terminé d'afficher les objets graphiques présents à l'image 20. Nous pouvons donc accéder sans problème à l'occurrence `monPersonnage`.

Grâce à la classe du document, nous allons mettre en place une fonctionnalité rendant tout ce traitement transparent.

Au sein de la classe `Application` nous définissons une méthode `myGotoAndStop` :

```
package org.bytearray.document

{

    import flash.display.MovieClip;
    import flash.events.Event;

    public class Application extends MovieClip

    {

         // propriétés liées à l'occurrence
         public var monPersonnage:MovieClip
         // propriété permettant l'exécution de la fonction de rappel
         private var rappel:Function;

         public function Application ()
```

```
        {

        }

        // méthode de déplacement de la tête de lecture personnalisé
        public function myGotoAndStop ( pImage:int, pFonction:Function ):void

        {

            // écoute de l'événement Event.RENDER
            addEventListener ( Event.RENDER, miseAJour );

            // déplacement de la tête de lecture
            gotoAndStop ( pImage );
            // retourne un objet permettant
            rappel = pFonction;

            // force la diffusion de l'événement Event.RENDER
            stage.invalidate();

        }

        private function miseAJour ( pEvt:Event ):void

        {

            // nous tentons d'appeler la fonction de rappel
            try
            {

                rappel();

            // si cela échoue, nous affichons un message d'erreur
            } catch ( pErreur:Error )
            {

                trace("Erreur : La méthode de rappel n'a pas été définie");

            // dans tout les cas, nous supprimons l'écoute de l'événement
Event.RENDER
            } finally
            {

                removeEventListener ( Event.RENDER, miseAJour );

            }

        }

    }

}
```

Ainsi lorsque nous souhaitons retrouver le comportement de la méthode `gotoAndStop` des anciennes versions d'ActionScript nous écrivons le code suivant :

```
// déplace la tête de lecture en image 10
// lorsque tout les objets sont disponibles
// la fonction actifAccessible est déclenchée
myGotoAndStop ( 20, actifAccessible );
```

```
function actifAccessible ():void

{

    // affiche : [object Personnage]
    trace(monPersonnage);

    // affecte la rotation de l'occurrence
    monPersonnage.rotation = 90;

}

stop();
```

La fonction `actifAccessible` est déclenchée automatiquement lorsque les occurrences de l'image 20 sont disponibles.

Grâce au bloc `try`, `catch`, `finally` nous gérons les erreurs à l'exécution. Au cas où l'utilisateur oublierait de définir la fonction associée :

```
// déplace la tête de lecture en image 10
// lorsque tout les objets sont disponibles
// la fonction actifAccessible est déclenchée
// affiche : Erreur : La méthode de rappel n'a pas été définie
myGotoAndStop ( 10, actifAccessible );

// fonction non définie
var actifAccessible:Function;

stop();
```

L'erreur à l'exécution est gérée et nous affichons le message indiquant l'oublie de définition.

Grâce à la classe du document nous pouvons ajouter des fonctionnalités au scénario principal ou modifier certains comportements. Nous étendons les capacités du scénario en définissant de nouvelles méthodes au sein de la classe du document.

A retenir

- Il est préférable de désactiver la déclaration automatique des occurrences.
- Les propriétés liées aux occurrences doivent toujours être publique.

Initialisation de l'application

Nous allons supprimer l'occurrence nommée `monPersonnage` sur la scène, puis initialiser l'application en instanciant par programmation l'instance de `Personnage`.

Afin d'initialiser notre application, nous utilisons simplement le constructeur de la classe du document. Dans le code suivant nous instancions puis affichons le symbole `Personnage` :

```
package org.bytearray.document

{

    import flash.display.MovieClip;
    import flash.events.Event;

    public class Application extends MovieClip

    {

         // propriété permettant l'exécution de la fonction de rappel
         private var rappel:Function;

         public function Application ()

         {

              // création du symbole Personnage
              var autrePersonnage:Personnage = new Personnage();

              // ajout à la liste d'affichage
              addChild ( autrePersonnage );

              // l'occurrence est centrée
              autrePersonnage.x = (stage.stageWidth - autrePersonnage.width)/2;
              autrePersonnage.y = (stage.stageHeight -
autrePersonnage.height)/2;

         }

         // méthode de déplacement de la tête de lecture personnalisé
         public function myGotoAndStop ( pImage:int, pFonction:Function ):void

         {

              // écoute de l'événement Event.RENDER
              addEventListener ( Event.RENDER, miseAJour );

              // déplacement de la tête de lecture
              gotoAndStop ( pImage );

              // retourne un objet permettant
              rappel = pFonction;

              // force la diffusion de l'événement Event.RENDER
              stage.invalidate();

         }

         private function miseAJour ( pEvt:Event ):void

         {

              // nous tentons d'appeler la fonction de rappel
              try
              {
```

```
                    rappel();

                // si cela échoue, nous affichons un message d'erreur
                } catch ( pErreur:Error )
                {

                    trace("Erreur : La méthode de rappel n'a pas été définie");

                // dans tout les cas, nous supprimons
                // l'écoute de l'événement Event.RENDER
                } finally
                {

                    removeEventListener ( Event.RENDER, miseAJour );

                }

            }

        }

    }
```

La figure 11-6 illustre le résultat :

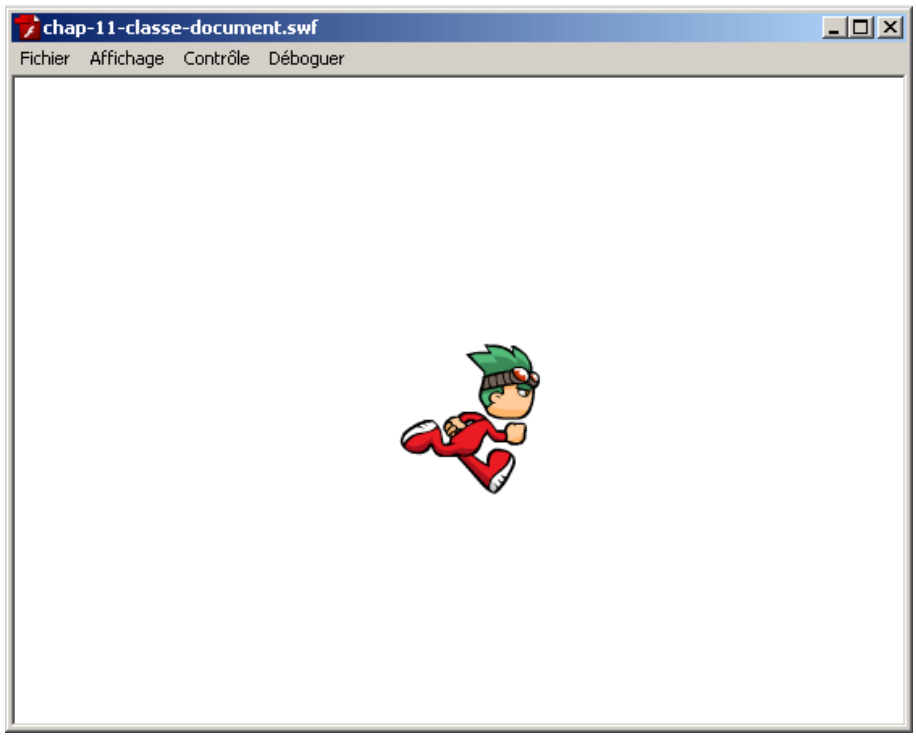

Figure 11-6. Symbole `Personnage` affiché.

Ne remarquez vous pas quelque chose de particulier ?

Nous devrions normalement écouter l'événement `Event.ADDED_TO_STAGE` pour pouvoir accéder à l'objet `Stage` de manière sécurisée. Dans le cas de la classe du document, le code est

exécuté une fois l'instance de la classe du document ajoutée à la liste d'affichage, ce qui est automatique et assuré par le lecteur. L'objet `Stage` est donc directement accessible par la propriété `stage` depuis le constructeur.

Accès global à l'objet Stage

Il serait intéressant d'utiliser la classe du document comme point d'accès global à l'objet `Stage`. Pour cela, nous définissons une propriété statique `GLOBAL_STAGE` au sein de la classe `Application` :

```
// point d'accès à l'objet Stage
public static var GLOBAL_STAGE:Stage;
```

Puis nous modifions le constructeur :

```
public function Application ()

{

    // affecte une référence à l'objet Stage
    Application.GLOBAL_STAGE = stage;


}
```

En ciblant la propriété `Application.GLOBAL_STAGE`, n'importe quel objet obtiendra une référence à l'objet `Stage`.

Afin d'illustrer cette fonctionnalité, nous allons définir au sein d'un répertoire `ui` une classe `Fenetre` :

```
package org.bytearray.ui

{

    import flash.display.Shape;

    public class Fenetre extends Shape

    {

         public function Fenetre ( pLargeur:Number, pHauteur:Number,
pRayon:Number, pCouleur:Number )

         {

              graphics.beginFill ( pCouleur );
              graphics.drawRoundRect ( 0, 0, pLargeur, pHauteur, pRayon );

         }

    }

}
```

Cette classe crée une forme rectangulaire illustrant une simple fenêtre.

En pointant vers la propriété `Application.GLOBAL_STAGE` nous obtenons un point d'accès global à l'objet `Stage` depuis n'importe quelle instance de la classe `Fenetre` :

```
package org.bytearray.ui

{

    import flash.display.Shape;
    import org.bytearray.document.Application;

    public class Fenetre extends Shape

    {

         public function Fenetre ( pLargeur:Number, pHauteur:Number,
pRayon:Number, pCouleur:Number )

         {

              graphics.beginFill ( pCouleur );
              graphics.drawRoundRect ( 0, 0, pLargeur, pHauteur, pRayon );

              // nous centrons la fenêtre
              // en utilisant la propriété statique Application.GLOBAL_STAGE
nous bénéficions d'un point d'accès global à l'objet Stage
              x = (Application.GLOBAL_STAGE.stageWidth - width)/2;
              y = (Application.GLOBAL_STAGE.stageHeight - height)/2;

         }

    }

}
```

Puis nous affichons la fenêtre :

```
package org.bytearray.document

{

    import flash.display.MovieClip;
    import flash.events.Event;
    import org.bytearray.ui.Fenetre;

    public class Application extends MovieClip

    {

         // point d'accès à l'objet Stage
         public static var GLOBAL_STAGE:Stage;
         // propriété permettant l'exécution de la fonction de rappel
         private var rappel:Function;

         public function Application ()

         {

              addEventListener ( Event.ADDED_TO_STAGE, activation );

         }
```

```
        private function activation ( pEvt:Event ):void

        {

            // affecte une référence à l'objet Stage
            Application.GLOBAL_STAGE = stage;

            // création de la fenêtre
            var maFenetre:Fenetre = new Fenetre( 250, 250, 8, 0x88CCAA );

            // ajout à la liste d'affichage
            addChild ( maFenetre );

        }

        // méthode de déplacement de la tête de lecture personnalisé
        public function myGotoAndStop ( pImage:int, pFonction:Function ):void

        {

            // écoute de l'événement Event.RENDER
            addEventListener ( Event.RENDER, miseAJour );
            // déplacement de la tête de lecture
            gotoAndStop ( pImage );
            // retourne un objet permettant
            rappel = pFonction;
            // force la diffusion de l'événement Event.RENDER
            stage.invalidate();

        }

        private function miseAJour ( pEvt:Event ):void

        {

            // nous tentons d'appeler la fonction de rappel
            try
            {

                rappel();

            // si cela échoue, nous affichons un message d'erreur
            } catch ( pErreur:Error )
            {

                trace("Erreur : La méthode de rappel n'a pas été définie");

            // dans tout les cas, nous supprimons l'écoute de l'événement
Event.RENDER
            } finally
            {

                removeEventListener ( Event.RENDER, miseAJour );

            }

        }

    }

}
```

La figure 11-7 illustre le résultat :

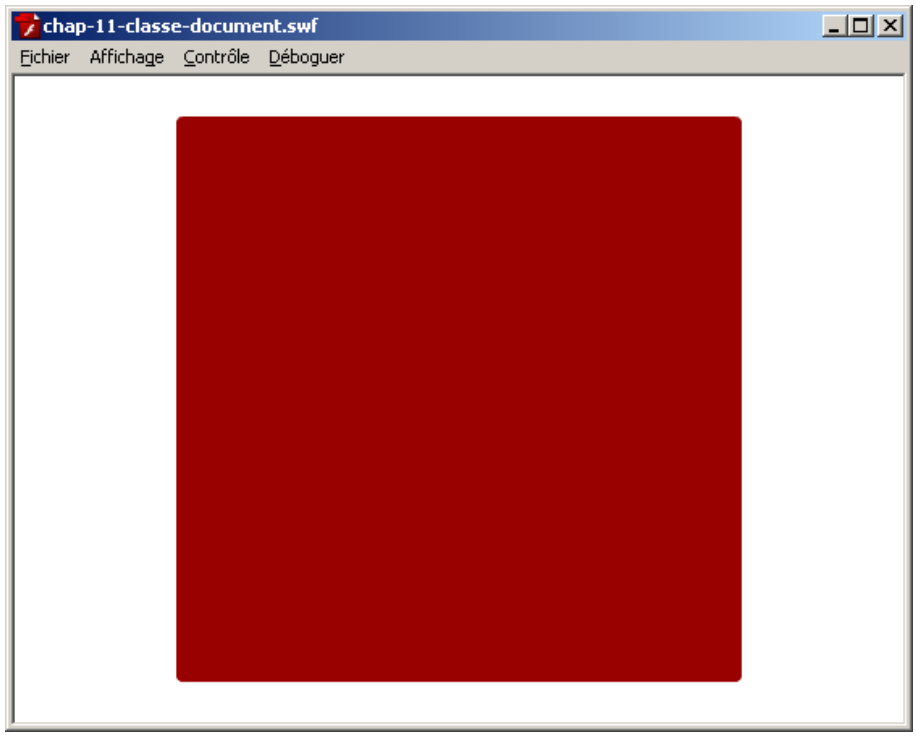

Figure 11-7. Instance de la classe `Fenetre` affichée.

Grâce à cette technique nous n'avons pas besoin d'écouter l'événement `Event.ADDED_TO_STAGE` afin de pouvoir cibler l'objet `Stage`. Mais comme nous l'avons vu précédemment, il est toujours recommandé d'écouter en interne l'événement `Event.ADDED_TO_STAGE` afin d'accéder à l'objet `Stage`. Cette technique permet donc aux objets non graphiques d'accéder à l'objet `Stage` de manière simplifiée.

En revanche, si nous devons réutiliser la classe `Fenetre` dans un autre projet, nous serons obligé de définir une classe du document nommée `Application` ainsi qu'une propriété statique `GLOBAL_STAGE` référençant l'objet `Stage`.

Afin d'automatiser ce processus, nous allons utiliser l'héritage.

Automatiser l'accès global à l'objet Stage

Pour pouvoir bénéficier d'un accès global à l'objet `Stage` automatiquement dans tous nos projets, nous avons plusieurs possibilités.

La première, comme nous venons de voir à l'instant consiste à définir dans la classe du document de chaque projet, une propriété statique

accessible de n'importe classe. Permettant ainsi un accès simplifié à l'objet `Stage`. Il serait dommage de devoir recopier ce code dans classe du document, nous préférerons donc utiliser la solution suivante.

Nous savons que la classe `Application` contient des fonctionnalités pouvant être nécessaires à chaque projet :

- Un accès global à l'objet `Stage`
- Une méthode `myGotoAndStop`

Afin d'hériter de ces fonctionnalités, nous allons utiliser pour chaque projet une classe du document héritant de la classe `Application`.

Il n'est pas nécessaire de dupliquer celle-ci dans le répertoire de classes de chaque projet nous allons l'isoler en la plaçant dans un répertoire spécifique global à tous nos projets.

A la racine de notre disque dur nous créons un répertoire nommé `classes_as3`. Puis nous créons répertoire `org` contenant un répertoire `abstrait`.

Au sein de ce dernier nous stockons la classe `ApplicationDefaut` dont voici le code complet final :

```
package org.bytearray.abstrait

{

    import flash.display.MovieClip;
    import flash.events.Event;
    import flash.display.Stage;

    public class ApplicationDefaut extends MovieClip

    {
          // point d'accès à l'objet Stage
          public static var GLOBAL_STAGE:Stage;
          // propriété permettant l'exécution de la fonction de rappel
          private var rappel:Function;

          public function ApplicationDefaut ()

          {

               // affecte une référence à l'objet Stage
               ApplicationDefaut.GLOBAL_STAGE = stage;

          }

          // méthode de déplacement de la tête de lecture personnalisé
          public function myGotoAndStop ( pImage:int, pFonction:Function ):void

          {

               // écoute de l'événement Event.RENDER
```

```
                addEventListener ( Event.RENDER, miseAJour );

                // déplacement de la tête de lecture
                gotoAndStop ( pImage );

                // retourne un objet permettant
                rappel = pFonction;

                // force la diffusion de l'événement Event.RENDER
                stage.invalidate();

            }

            private function miseAJour ( pEvt:Event ):void

            {

                // nous tentons d'appeler la fonction de rappel
                try
                {

                    rappel();

                // si cela échoue, nous affichons un message d'erreur
                } catch ( pErreur:Error )
                {

                    trace("Erreur : La méthode de rappel n'a pas été définie");

                // dans tout les cas, nous supprimons l'écoute de l'événement
Event.RENDER
                } finally
                {

                    removeEventListener ( Event.RENDER, miseAJour );

                }

            }

        }

    }
```

Afin de spécifier un chemin d'accès de classes global à tous les projets au sein de Flash CS3, nous cliquons sur *Modifier* puis *Préférences* puis dans la liste nous sélectionnons *ActionScript*.

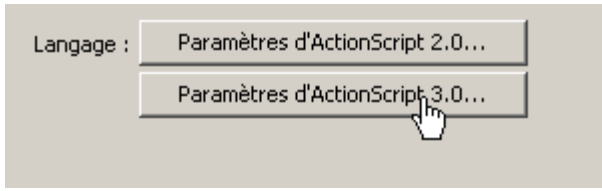

Figure 11-8. Paramètres d'ActionScript 3.

Une fois cliqué sur le bouton *Paramètres d'ActionScript 3* le panneau indiquant le chemin d'accès aux classes s'affiche. Comme l'illustre la figure 11-9.

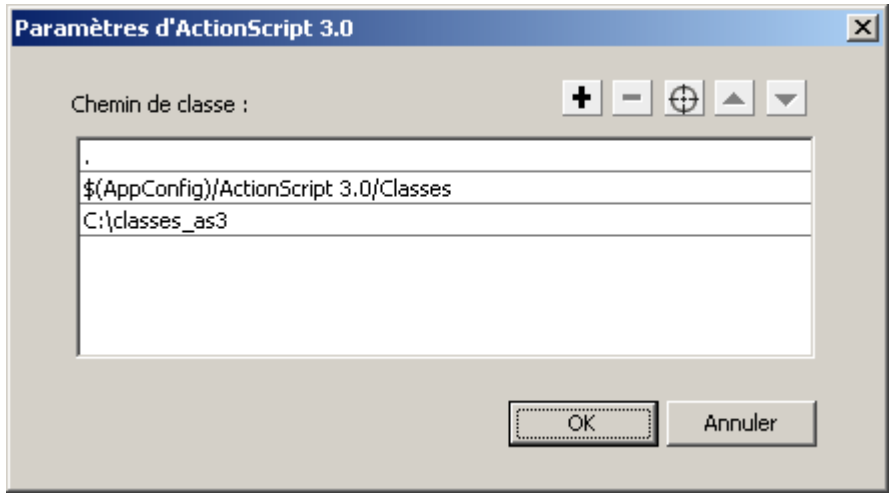

Figure 11-9. Chemin d'accès aux classes.

En cliquant sur l'icône + nous ajoutons un champ afin de spécifier le chemin d'accès au répertoire contenant les classes globales. Dans notre exemple nous ajoutons une entrée contenant le chemin `C:\classes_as3`.

Deux autres lignes sont déjà présentes, la première indique que le compilateur doit regarder à coté du document en cours. Puis la deuxième indique le chemin d'accès aux classes natives de Flash.

En cliquant sur OK, les classes stockées au sein du répertoire `classes_as3` seront disponibles depuis n'importe quel projet.

Nous allons modifier notre application en affectant une classe du document héritant de la classe `ApplicationDefaut`.

Au sein du répertoire document nous créons une classe `Document` :

```
package org.bytearray.document

{

    import org.bytearray.abstrait.ApplicationDefaut;

    public class Document extends ApplicationDefaut

    {

         public function Document ()

         {

              trace( this );

         }

    }
```

```
}
```

Puis nous l'affectons comme classe du document en cours, comme l'indique la figure 11-10 :

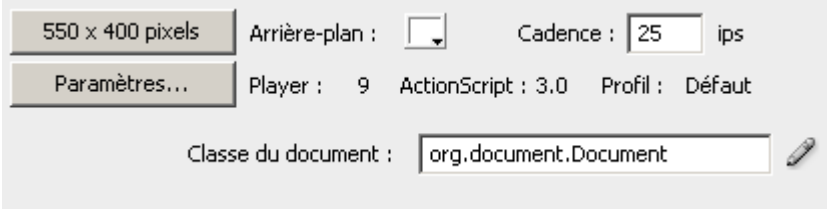

Figure 11-10. Affectation de la classe `Document`.

A la compilation `[object Document]` s'affiche dans le panneau *Sortie*.

La classe `ApplicationDefaut` devient ainsi une classe ne devant être qu'héritée. Une sorte de classe abstraite, même si comme nous l'avons vu lors du chapitre 8 intitulé *Programmation orientée objet*, ActionScript 3 ne permet pas la définition de vraie classe abstraite.

Tous les objets présents ou non au sein de la liste d'affichage devant faire référence à l'objet `Stage` devront cibler la propriété `ApplicationDefaut.GLOBAL_STAGE`.

Nous modifions donc la classe `Fenetre` :

```
package org.bytearray.ui

{

    import flash.display.Shape;
    import org.bytearray.abstrait.ApplicationDefaut;

    public class Fenetre extends Shape

    {

        public function Fenetre ( pLargeur:Number, pHauteur:Number,
pRayon:Number, pCouleur:Number )

        {

            graphics.beginFill ( pCouleur );
            graphics.drawRoundRect ( 0, 0, pLargeur, pHauteur, pRayon );

            // nous centrons la fenêtre
            // en utilisant la propriété statique Application.GLOBAL_STAGE
            //  nous bénéficions d'un point d'accès global à l'objet Stage
            x = (ApplicationDefaut.GLOBAL_STAGE.stageWidth - width)/2;
            y = (ApplicationDefaut.GLOBAL_STAGE.stageHeight - height)/2;

        }

    }

}
```

Puis nous modifions la classe `Document` afin d'instancier la classe `Fenetre` :

```
package org.bytearray.document

{

    import flash.events.Event;
    import org.bytearray.abstrait.ApplicationDefaut;
    import org.bytearray.ui.Fenetre;

    public class Document extends ApplicationDefaut

    {

          public function Document ()

          {

               // création de la fenêtre
               var maFenetre:Fenetre = new Fenetre( 250, 250, 8, 0x88CCAA );

               // ajout à la liste d'affichage
               addChild ( maFenetre );

          }

    }

}
```

La figure 11-11 illustre le résultat :

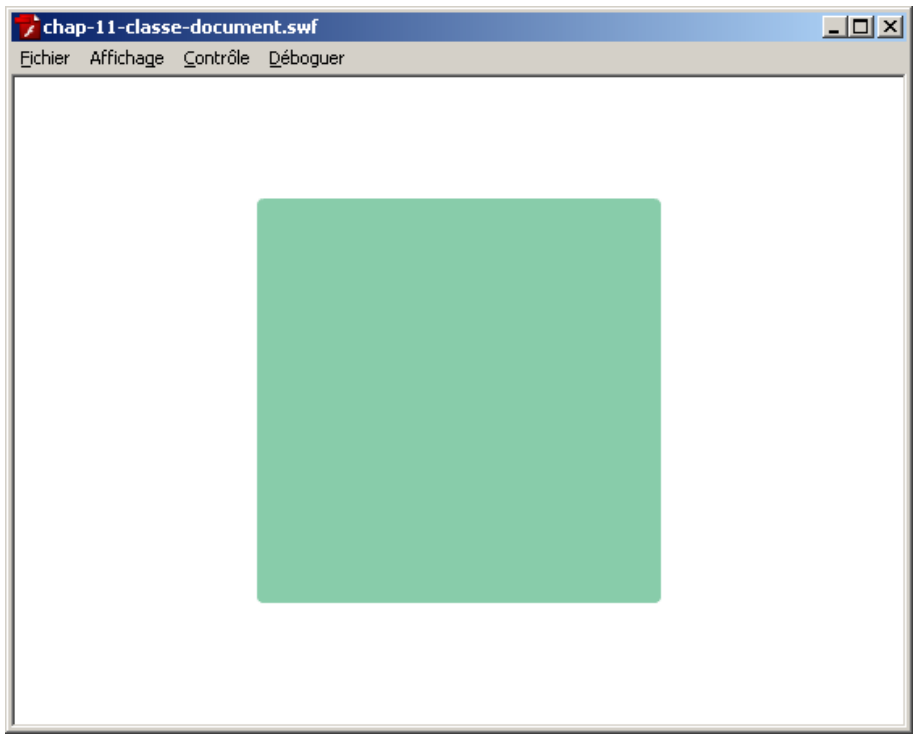

Figure 11-11. Instance de la classe `Fenetre`.

Souvenez-vous, la sous-classe n'hérite pas des propriétés statiques de la classe parente. Ainsi, la classe `Document` n'hérite pas de la propriété `GLOBAL_STAGE` de la classe `ApplicationDefaut`.

A retenir

- La fenêtre *Paramètres d'ActionScript 3* permet de définir un chemin d'accès global aux classes.

Nous allons nous intéresser au cours du prochain chapitre à la gestion des bitmap par programmation en ActionScript 3. Les classes `flash.display.Bitmap` et `flash.display.BitmapData` n'auront plus de secrets pour vous !