Programació amb contingut digital

Llogari Casas Torres

PID_00186386

UOC
Universitat Oberta de Catalunya

Índex

1. Introducció

1.1. Què és una classe? Classes i programació orientada a objectes

Una classe és en essència una plantilla que ens permet dur a terme un conjunt de feines sense necessitat de saber de manera concreta que és el que fa cada apartat concret de la classe.

Conceptualment la nostra vida quotidiana és plena de "classes", ja que, per exemple, podríem afirmar que la majoria no sabem com funciona el sensor d'una càmera fotogràfica però sí que sabem que, prement el botó del disparador, aquest sensor farà la fotografia que hem enquadrat en el visor.

Al llarg d'aquest apartat explicarem de manera planera i simple què és una classe, d'on pren les referències que en definiran les característiques, com les relaciona i quines en són les possibles utilitats. Al final d'aquest veureu a més que, sense adonar-vos-en, probablement sou molts els que ja heu usat classes en algunes tasques fetes en Flash.

Les classes són la base de construcció de qualsevol **programa orientat a objectes**, ja que en aquestes s'agrupen les propietats i mètodes que defineixen l'objecte. Aquest tipus de programació es basa en un model pres de la vida real, de l'experiència individual i col·lectiva de cada un de nosaltres, i encara que pugui semblar estrany, en molts aspectes, programar d'aquesta manera requereix més conceptes artístics i filosòfics que no pas conceptes tècnics. Això es deu al fet que els resultats no depenen tant de la tècnica del programador com del que la seva imaginació i destresa li permet imaginar i recrear.

Si mireu al vostre voltant podreu veure fàcilment que qualsevol objecte que tingueu al davant té uns **atributs** determinats i compleix un mínim d'una **funció** concreta. A més, aquesta funció pot ser **imprescindible** o **prescindible**.

Per exemple, si disposeu d'una càmera fotogràfica com la que esmentàvem un parell de paràgrafs més amunt, estareu ràpidament d'acord que una funció imprescindible del que correspondria a l'objecte cámara_fotográfica és que faci fotografies. Una funció prescindible d'aquest objecte és que faci, per exemple, un efecte de virado_a_sepia d'aquesta fotografia.

Si continuem explorant aquest mateix objecte trobarem que dins del grup de propietats podríem definir diferents apartats segons el color, segons si disposa de zoom, segons el nombre de megapíxels, etc.

D'altra banda si observem el grup de funcionalitats podrem veure si només permet fer fotografies estàtiques, si permet triar entre diferents formats d'imatge com poden ser RAW, JPEG o TIFF, si permet gravar seqüències d'imatges o gravar so, etc.

Exemple

Si l'exemple que acabeu de llegir l'extrapoléssim a un objecte programable sota ActionScript, identificaríem el grup d'atributs sota el nom de **propietats** i el de funcionalitats sota el nom de **mètodes**.

Quan mirem la càmera, cada atribut o propietat ens torna una resposta concreta i diferent de les altres.

Així, per exemple, per a la Propiedad_color la resposta serà el color del cos de la càmera. Si és de color negre obtindrem com a resposta el valor 0xFF000000 en cas de treballar en colors de l'escala hexadecimal, o el conjunt de valors (0, 0, 0) en cas de fer-ho en colors de l'escala RGB.

Un altre exemple podria ser la Propiedad_zoom.

En aquest cas la resposta estaria determinada pel fet de tenir o no tenir òptica de focal variable. Això significa que només podria admetre valors booleans[1], és a dir, sí o no.

(1)En informàtica la identificació d'aquesta propietat seria, doncs, vertadera o falsa segons disposés o no disposés l'òptica esmentada.

Si mirem el grup de funcionalitats o mètodes, cada un ofereix en si mateix respostes directes.

Així, per exemple, si escollim el format RAW obtindrem una fotografia diferent amb la càmera que si escollim el format JPEG.

Tot això, sumat a les altres característiques, és justament el que defineix un **objecte**.

Un objecte és un element que disposa d'atributs i funcionalitats i que, en el món de la informàtica, podem programar assignant-hi propietats que es mostraran en forma d'atributs i mètodes.

Ells seran els encarregats que aquest objecte compleixi les funcionalitats encomanades. Les propietats de l'objecte es fixen sempre durant el seu procés de creació i programació, mentre que les respostes poden ser variables en funció de diversos factors com, per exemple, accions de l'usuari.

Tot aquest grup de propietats i mètodes que acabem de definir i que conformen i identifiquen un objecte es denomina **classe**, i cada una de les seves propietats i mètodes són coneguts sota el nom de **membres de classe**. Així, doncs, la classe Cámara_foto definiria l'objecte càmera, i els membres de classe d'aquest objecte serien tant els components físics com les diferents opcions que poguéssim escollir per menú.

Fins aquí hem descrit què és un objecte i quina relació guarda amb la seva classe, però, **us imagineu a vosaltres mateixos construint una càmera real?** La veritat és que seria molt difícil. La raó cal buscar-la a l'interior de la mateixa càmera, a les seves peces i components interns, cada uns dels quals és en si mateix un objecte diferent i que, per tant, pertany a una classe diferent.

Vist això, una pregunta òbvia podria ser **com es relacionen entre ells tots aquests objectes?** La resposta la trobem en el fet que a més d'aquestes característiques que acabem de descriure n'hi ha d'altres que ajuden a definir tant l'objecte Cámara_foto com qualsevol altre. Aquestes altres característiques són l'**abstracció**, l'**encapsulació** i l'**herència**.

1.1.1. Abstracció

Si obrim una càmera fotogràfica real podem veure que disposa de multitud d'elements electrònics interns que no sabem ni per a què serveixen ni quina és la seva funció o funcions específiques. Però, **necessitem saber per a què serveix tot això quan fem una fotografia?** Segurament estem d'acord amb el que ja esmentàvem en iniciar l'apartat: per a molts de nosaltres és suficient de saber quan prémer el botó del disparador. Amb això podrem fer fotografies meravelloses sense necessitat de saber absolutament res dels components electrònics interns. Simplement hem de saber on és el botó i quan l'hem de prémer, i la càmera ja s'encarregarà d'efectuar la resta del procés: fer la fotografia i mostrar-la en el visor.

Tanmateix, si ens parem a pensar en quan a la fàbrica es va confeccionar la càmera fotogràfica de l'exemple, ens adonarem que el primer que es va fer no va ser pensar en les seves propietats externes ni tan sols en el temps que duraria el procés de la presa d'imatge, sinó en els mecanismes interns, justament en la part que desconeixem de la càmera però que sabem que la fa funcionar.

> En realitat, doncs, el que es va fer a la fàbrica durant aquesta etapa de creació només va ser **abstreure les seves funcions internes** perquè l'usuari sols hagi de prémer el botó del disparador i la imatge quedi gravada en la targeta de memòria.

Posem ara aquest exemple que acabem de veure en forma de **pseudocodi**. Comencem declarant una variable de tipus **Cámara_foto** i per mitjà de l'operador **new** la creem en la memòria.

```
var camara_fotos: Cámara_foto = new Cámara_foto ();
```

Assignarem el valor Nikon a la propietat *nom* de la càmera que acabem de crear.

```
camara_fotos.identificador = "Nikon D70";
```

Li indicarem que quan hagi acabat d'emmagatzemar una fotografia ens ho comuniqui.

```
camara_fotos.escribe (camara_fotos.identificador + "ha terminado de almacenar la imagen.");
```

Automàticament, quan s'hagi acabat d'emmagatzemar la imatge, la càmera que hem creat mostrarà el missatge següent:

Nikon D70 ha acabat d'emmagatzemar la imatge.

Atenció

No espereu que el Flash respongui a aquest codi amb alguna cosa que no sigui un missatge d'error. L'explicació només s'escriu com a exemple, ja que perquè realment funcionés en Flash el primer que hauria d'existir és la classe Cámara_foto. En apartats posteriors aprendrem com podem crear realment un arxiu que ens serveixi com a classe.

1.1.2. Encapsulació

Si seguim amb el mateix exemple de la classe Camara_foto que hem creat, veurem que hi ha algunes dades que pot ser convenient que qui l'usi conegui o que desconegui.

Per exemple, si la deixem a un conegut, a simple vista podrà identificar si es tracta d'una càmera rèflex o una càmera compacta, però hem de poder escollir si volem que pugui accedir a les fotografies que nosaltres havíem fet prèviament o bé preferim que algunes dades, imatges en aquest cas, quedin reservades per al nostre ús privat.

Aquesta part del procés de la creació d'una classe es coneix com a **encapsulació**. En aquesta decretarem les variables necessàries per al seu funcionament, però no seran accessibles per a l'usuari final tret que entrem a reescriure l'arxiu de la classe.

1.1.3. Herència

Fins ara l'objecte Cámara_foto és molt genèric, però tot i així es pot considerar un objecte abstracte, ja que evidentment en cap moment no permet crear una càmera concreta però tanmateix sí que permet que la funció que hem creat s'executi en qualsevol tipus de càmeres. Vist això podríem crear dos nous objectes que podríem anomenar **camara_reflex** i **camara_compacta**, que heretarien totes les característiques genèriques de l'objecte Cámara_foto. Les diferències que hi pugui haver s'han d'evidenciar en aquests nous objectes *a posteriori* quan ja els hàgim creat.

Així, doncs, podrem afirmar que totes les instàncies tant de cámara_reflex com de cámara_compacta tindran les mateixes característiques natives, i que mitjançant la programació podrem indicar mètodes específics per a cada una.

Per exemple, per a l'objecte cámara_reflex podrem crear un mètode que ens permeti "cambiar_la_óptica" mentre que per a l'objecte cámara_compacta en podrem crear un que ens permeti "activar_zoom_digital". Tot això serà possible gràcies al fet que tots dos objectes heretaran les seves propietats i mètodes de l'objecte primari Cámara_foto.

A hores d'ara seria fàcil preguntar-nos per què és necessari crear les dues noves classes camara_reflex i camara_compacta si, en realitat, en la majoria dels casos podríem treballar directament des de la classe superior Cámara_foto.

La resposta l'hem de buscar en el concepte de **classe abstracta**.

Una classe abstracta és una classe a la qual no es poden instanciar objectes. La seva funció no és cap altra que la de traspassar a altres objectes les seves propietats i mètodes, i l'objecte hereu és el que sí que podrà ser instanciat en l'escenari Flash.

En el cas del nostre exemple les classes cámara_reflex i cámara_compacta són instanciables perquè hereten les propietats i mètodes de la classe Cámara_foto. Tanmateix, la classe Cámara_foto no pot ser instanciada com un nou objecte, ja que en tractar-se d'una classe abstracta el programa no ho permet.

Resumint tot el dit fins aquí, és evident que podem tornar al principi de l'apartat per a reafirmar-nos en el fet que una classe funciona com una plantilla que únicament cal saber omplir, i que una vegada confeccionada es pot usar i emplenar infinitat de vegades amb elements del mateix tipus però de contingut diferent.

1.2. Un primer exemple molt senzill en AS 2.0

Probablement tots coneixeu àmpliament la possibilitat d'incorporar imatges dins del **Flash**. Quan incorporem una imatge a l'escenari del programa no fem sinó incrustar un ***bitmap*** (mapa de bits), usar aquest escenari com a visor d'imatges. En realitat, en fer aquesta acció utilitzem, sense saber-ho, una classe pròpia del programa. Tanmateix, en aquest cas es tracta d'una classe genèrica i, per tant, no instanciable, per això en finalitzar tan sols podem veure una rèplica de la imatge original sense que hàgim pogut fer res més amb ella.

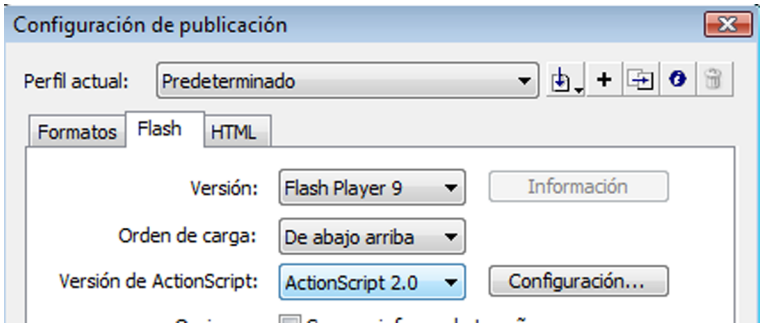
I, **què cal fer per a poder començar a treballar amb ella?** Una bona opció és recórrer a una instància d'una classe ja sigui en el Flash; en aquest cas podria ser la **classe MovieClip**. Si hi recorrem podrem crear fàcilment una instància de l'objecte movieclip que heretarà tots els mètodes i propietats de la classe original. En realitat això és el que fem quan creem un clip de pel·lícula en un arxiu Flash.

L'exemple següent crea una instància MovieClip en la qual es carrega una imatge externa al Flash utilitzant el mètode **loadMovie** –el qual pertany a la classe MovieClip– i se situa dins d'un clip situat en l'escenari i identificat en la finestra de *Propiedades* amb el nom *mc:*

```
loadMovie("imagen.jpg",_root.mc);
```

Atenció

Aquesta manera de carregar imatges està descatalogada en l'ActionScript 3.0 i, per tant, no funciona amb aquest llenguatge. Si treballeu amb una configuració d'AS3 aneu a *Archivo\Configuración de la publicación* i indiqueu en el quadre de diàleg emergent que voleu treballar amb l'ActionScript 2.0. Aquesta assignatura està pensada per a fer-la bàsicament en aquesta versió de l'ActionScript.



Observació

Fixeu-vos que el clip de pel·lícula (*movie clip*) en el qual es fa la càrrega va precedit de la partícula _root. Recordeu que aquesta partícula indica la relació entre els elements i l'escenari.

Encara que aparentment el resultat final serà el mateix que quan incorporàvem la imatge directament en l'escenari, hi ha diferències substancials com, per exemple, el fet de poder eliminar la imatge de l'escenari quan ens interessi usant mètodes propis de la mateixa classe MovieClip com pot ser, per exemple, **unloadMovie**().

Amb això farem desaparèixer de l'escenari la instància de MovieClip i, com a conseqüència, també desapareixerà la imatge que hi ha en aquest clip.

L'avantatge de treballar a partir d'objectes no acaba aquí sinó que tan sols comença. Vegem un exemple una mica més complicat. En aquest cas volem posicionar una imatge en l'escenari i fer que aquesta quedi reflectida respecte de la seva base.

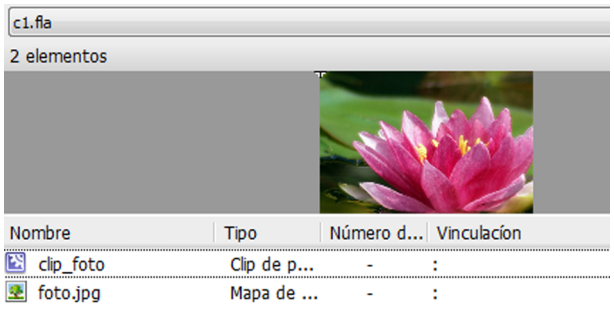
1.3. Creació dinàmica del reflex d'una imatge

1) Obriu l'arxiu **reflejo1.fla**, que hi ha en la carpeta de recursos de l'assignatura.

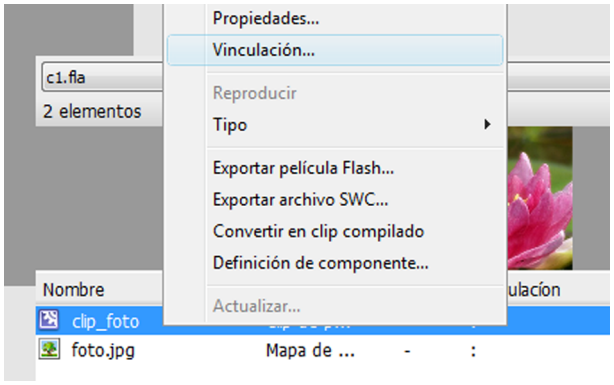
2) Fixeu-vos que l'**escenari del projecte** és completament buit.

3) Aneu a la finestra de la biblioteca. Hi trobareu dos elements: la imatge que usarem per a crear el reflex "**foto.jpg**" i el clip que la conté, "**clip_foto**".

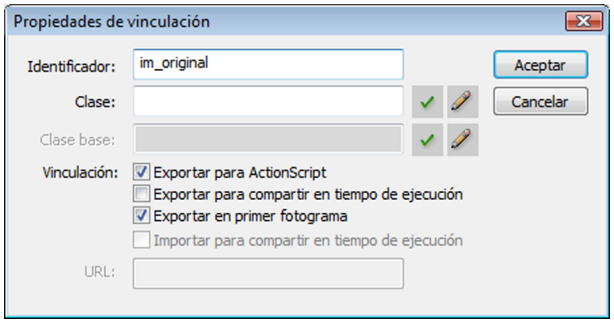
4) Deixeu-los en la **biblioteca**. Posicionarem el clip directament en l'escenari utilitzant codi.



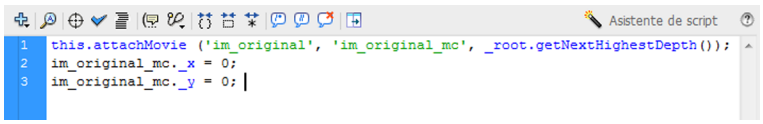
5) Sense abandonar la finestra de la biblioteca, localitzeu el clip que conté, i fent clic sobre el seu nom amb el botó secundari del ratolí apareixerà un menú contextual en el qual haureu d'escollir ***Vinculación***.



6) En fer-ho apareixerà un nou quadre de diàleg en el qual haureu d'activar les caselles que mostren la imatge inferior i assignar un nom que farà d'identificador del clip ubicat en la biblioteca a ActionScript. Assigneu com a nom "**im_original**", tal com es pot veure en la imatge inferior.



7) Accepteu el quadre de diàleg, aneu a la finestra d'**accions** del projecte Flash i introduïu el codi de la imatge següent. Recordeu que per a afegir el codi heu d'anar a la finestra d'accions.



Atenció

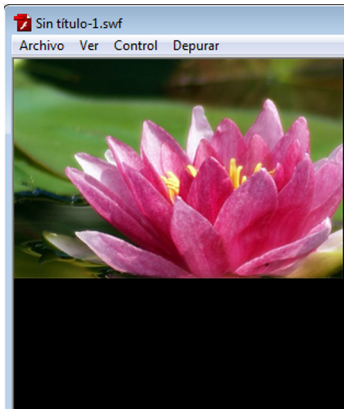
Els colors del text poden variar en funció de la configuració de cada usuari.

Si analitzem aquest fragment de codi podrem veure dues parts: la primera línia seria una part, i les altres dues una altra.

- En el cas de la primera línia el que fem en realitat és afegir a l'escenari una instància del clip que es troba identificat en la biblioteca. Identifiquem aquesta nova instància sota el nom *im_original_mc*.

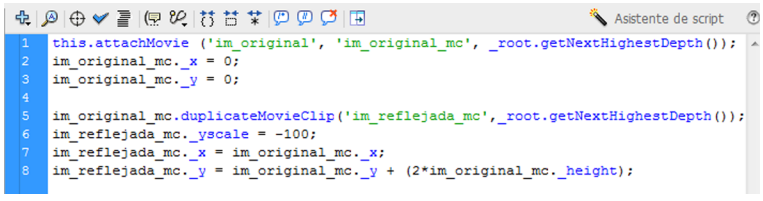
- Les altres dues línies defineixen el punt a partir del qual es començarà a mostrar el clip en l'escenari, en aquest cas seran les coordenades (0,0).

Si ara proveu l'escena veureu que la imatge que era en la biblioteca ara es troba col·locada en la part superior de l'escenari del Flash Player.



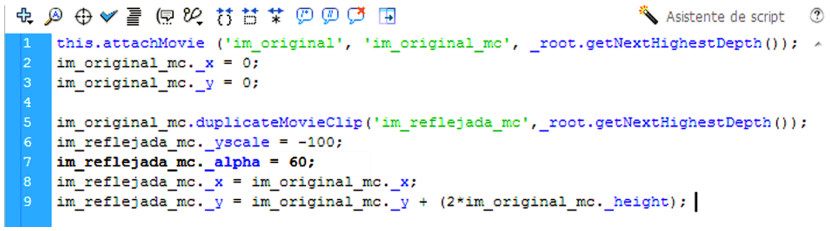
A continuació generarem la imatge reflectida. Farem aquesta operació duplicant la instància que acabem de col·locar en l'escenari mitjançant el codi anterior i reposicionant-la en el lloc que ens interessi.

Per a això, en la finestra d'accions afegirem les línies de codi de la imatge següent.



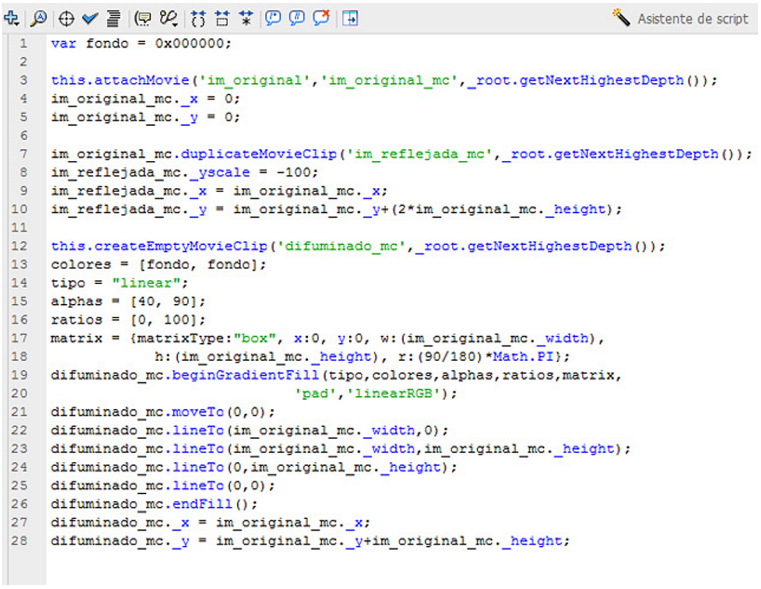
Fixeu-vos que partint de la instància im_original_mc hem duplicat la instància esmentada i l'hem rebatejat amb el nom *im_reflejada_mc* per a així poder-nos-hi referir i, per tant, poder-la reposicionar en el lloc que ens interessi. En aquest cas, sota el clip que conté la imatge original.

Si ara proveu el vostre arxiu veureu que es mostra la imatge original i la mateixa imatge invertida formant el reflex de la primera. Per ajustar una mica el reflex que hem fet, podem incorporar en el nostre codi una línia que reguli la intensitat d'aquest reflex.



Fins aquí hem creat el reflex que esperàvem obtenir mentre hem pogut comprovar alguns avantatges de treballar partint d'objectes que pertanyen a classes concretes, en el nostre cas a la classe MovieClip, però això no és ni de bon tros el final, solament és l'inici. En la imatge següent podeu veure l'efecte de reflex d'aquest arxiu que s'ha aconseguit usant de base el mateix codi però afegint-hi algunes línies més.

Al llarg d'aquests programes d'aprenentatge aprendrem com es poden fer coses similars de manera relativament senzilla usant tan sols objectes pertanyents a classes ja incorporades en Flash i sense saber com funcionen exactament per dins, sinó simplement coneixent alguns dels resultats que es poden obtenir. En tot cas i per si algú vol anar provant, el codi que genera la imatge anterior és el següent:




```
var fondo = 0x000000;

this.attachMovie('im_original','im_original_mc',_root.getNextHighestDepth());
im_original_mc._x = 0;
im_original_mc._y = 0;

im_original_mc.duplicateMovieClip('im_reflejada_mc',_root.getNextHighestDepth());
im_reflejada_mc._yscale = -100;
im_reflejada_mc._x = im_original_mc._x;
im_reflejada_mc._y = im_original_mc._y+(2*im_original_mc._height);

this.createEmptyMovieClip('difuminado_mc',_root.getNextHighestDepth());
colores = [fondo, fondo];
tipo = "linear";
alphas = [40, 90];
ratios = [0, 100];
matrix = {matrixType:"box", x:0, y:0, w:(im_original_mc._width),
          h:(im_original_mc._height), r:(90/180)*Math.PI};
difuminado_mc.beginGradientFill(tipo,colores,alphas,ratios,matrix,
                                 'pad','linearRGB');
difuminado_mc.moveTo(0,0);
difuminado_mc.lineTo(im_original_mc._width,0);
difuminado_mc.lineTo(im_original_mc._width,im_original_mc._height);
difuminado_mc.lineTo(0,im_original_mc._height);
difuminado_mc.lineTo(0,0);
difuminado_mc.endFill();
difuminado_mc._x = im_original_mc._x;
difuminado_mc._y = im_original_mc._y+im_original_mc._height;
```

2. Programació orientada a objectes: creant la nostra primera classe

En l'apartat anterior hem vist què és una classe i alguns petits avantatges de treballar usant elements d'una classe. És el moment de començar a posar en pràctica aquests conceptes que s'han desenvolupat més amunt i començar a veure com es crea una classe.

Per a això crearem, de manera conceptual i començant des de zero, una nova classe de ActionScript que ens serveixi per a visualitzar imatges com un **projector de diapositives**. Anomenarem aquesta classe **Proyector**.

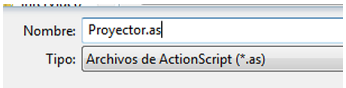
Per començar a crear la nostra classe el primer que farem és crear un arxiu per a allotjar-la. No és necessari que sigui un arxiu de Flash, ens pot servir qualsevol programa en el qual es pugui escriure text.

Si tot i així optem per usar Flash, no crearem un nou projecte Flash sinó que indicarem a aquest programa que volem crear un arxiu d'ActionScript. Això ens obrirà en la interfície Flash una finestra de Script en la qual podrem escriure el nostre codi.

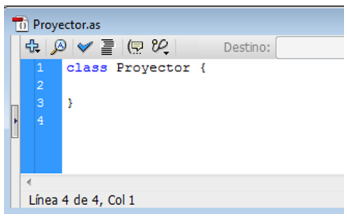
Fixeu-vos que no es crea cap escenari i que tampoc no hi ha línia de temps en aquest tipus d'arxius. Això és perquè el que farem no depèn de cap d'aquests dos elements.

Atenció

Perquè tot allò que fem funcioni correctament és important respectar dos aspectes bàsics. D'una banda, hem de desar el nostre arxiu amb el mateix nom que la classe que crearem respectant tant les lletres majúscules com les minúscules. D'altra banda, si treballem amb un programa d'edició de textos com pot ser la Llibreta del Windows, per exemple, quan acabem el nostre treball hem de rebatejar l'extensió com a *.as*. Les extensions *.as són propietàries dels arxius de codi de l'ActionScript.



Dit això ja podem començar a escriure la nostra classe dins de l'arxiu de text. Per a això, en l'arxiu escriurem el següent:



Simplement amb aquestes línies ja hem creat nostra nova classe. Ara toca omplir-la amb tot el que volem que faci, i ho situarem entre les claus.

Encara sense proposar-nos-ho de manera conscient, quan a l'inici de l'apartat dèiem que anàvem a crear una espècie de projector de diapositives ja estàvem definint de manera tàcita alguna de les seves funcions essencials: un projector de diapositives convencional, analògic, carrega una filmina del carro i la mostra en la pantalla.

> Això és justament el que volem fer, carregar una imatge d'un URL determinat i veure-la en el monitor de l'ordinador.

Evidentment podríem afegir moltes altres funcions –i de fet ho farem més endavant–, però si alguna d'aquestes dues no funciona, no ens servirà de res tot allò que vulguem afegir.

De fet, amb el que ens plantegem fer només fem una aplicació similar a una **API**[2]. Tanmateix, en aquest cas no es tractarà d'una **API de dibuix vectorial**[3] com la que ja està integrada dins del Flash, sinó que es tractarà d'una API de visualització d'imatges.

[2] **Què és una API?** El terme API (*application programming interface*) es refereix als serveis que subministra una determinada classe. Una API com la que estem començant a dissenyar podrà mostrar imatges carregades dinàmicament també en temps d'execució.

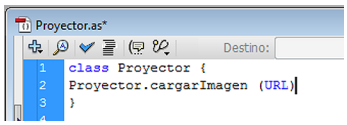
[3] **Què és l'API de dibuix del Flash?** És un conjunt de mètodes de la classe MovieClip que ens permet dibuixar traços i farciments. Els mètodes de l'API són els següents: beginBitmapFill, beginFill, beginGradientFill, clear, curveTo, endFill, lineGradientStyle, lineStyle, lineTo i moveTo.

Si a més partim del fet que hem de poder accedir a qualsevol instància de Proyector perquè la nostra classe funcioni i carregui la imatge que nosaltres volem i no la que li sembli a la classe, resulta força obvi que qualsevol instància de la classe Proyector que instanciem en l'escenari del nostre arxiu Flash haurà de poder cridar una ordre per a carregar la imatge.

Aquesta ordre necessitarà conèixer l'URL en el qual es troba la imatge que s'hagi de carregar.

Així, doncs, ja tenim un aspecte definit: necessitem un mètode que permeti especificar un URL per a cada imatge.

El mètode que necessitem és **cargarImagen**() i la manera bàsica d'escriure-ho en la classe podria ser:



El mètode **cargarImagen**() farà de descripció de l'acció dins del codi de la mateixa manera que un verb dins d'una oració gramatical.

El nen salta --- Què fa el nen? = Saltar

Proyector.cargarImagen --- Què fa Proyector? = Carregar una imatge

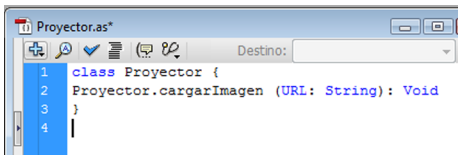
Tanmateix, en el nostre cas hem afegit un paràmetre concret al mètode **cargarImagen**(), l'**URL**.

Un URL no és més que una cadena de text.

Les cadenes de text d'un arxiu qualsevol no ens tornen cap valor numèric que puguem usar *a posteriori*. Una cadena de text simplement es limitarà a observar i transcriure el que hi hagi en la cadena. Així, doncs, ja podem començar a refinar una mica la línia de codi que abans hem escrit i a canviar-ne el contingut pel següent:

```
Proyector.cargarImagen (URL: String): Void
```

Amb la qual cosa el codi de la nostra classe quedaria momentàniament així:



Amb aquests dos afegits hem especificat dues coses:

- D'una banda, en introduir **String** hem indicat que el contingut del paràmetre URL serà una cadena de text.
- De l'altre, l'ús de **Void** indica que no hem d'esperar que la funció que hem escrit ens torni un valor.

A hores d'ara és fàcil que algú que sàpiga una mica d'ActionScript pensi que ens estem complicant la vida, ja que podríem carregar una imatge en l'escenari d'un arxiu Flash simplement escrivint en la finestra d'accions el codi que ja esmentàvem en el primer apartat:

```
loadMovie ("imagen.jpg", _root.mc);
```

Cert, això també funcionaria i carregaria la imatge en la interfície de Flash Player, però l'inconvenient és que únicament carregaria aquesta imatge, i tant per a descarregar-la com per a carregar i descarregar noves imatges necessitaríem introduir moltes altres ordres amb el consegüent increment de temps i de les possibilitats d'errors.

Tornem, doncs, a la classe que estem elaborant. Ara que ja tenim el codi corresponent a la funció de càrrega de la imatge, veurem com podem implementar en Proyector la seva segona funcionalitat, la que ens permetrà visualitzar en pantalla la imatge que acabem de carregar.

El millor que podem fer és crear un **clip** per a cada imatge. Això ens assegurarà que cada imatge es carregui correctament i, si fa falta, hi puguem interactuar sense por que perdi les referències respecte a la imatge original. En els llenguatges orientats a objectes, l'opció adequada és ocupar els **constructors de classe**.

> Un constructor de classe és un mètode pertanyent a la classe que té com a característiques especials el fet que, d'una banda, té el mateix nom que la classe i, de l'altra, no ens torna cap tipus d'informació.

Així, doncs, un possible codi seria el següent:

```
Proyector (destino: MovieClip);
```

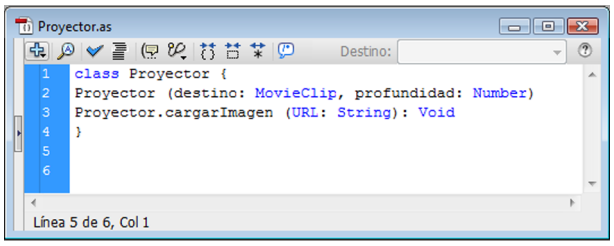
Amb això hem solucionat un problema, ja podem carregar la imatge en un clip concret, però encara podem fer més per a assegurar-nos que cada imatge carregada es col·locarà a sobre de l'anterior. Podem carregar-les **per nivells**[4].

(4)Els nivells del Flash no es mostren en temps de creació sinó en temps d'execució. Funcionen de manera similar a les capes i permeten sobreposar en nivells superiors continguts o arxius de manera dinàmica. Podeu consultar l'ajuda del programa per a obtenir més informació sobre com funcionen els nivells.

Perquè els clips contenidors de les imatges es mostrin en nivells n'hi haurà prou a afegir un altre paràmetre al constructor. Aleshores el codi queda de la manera següent:

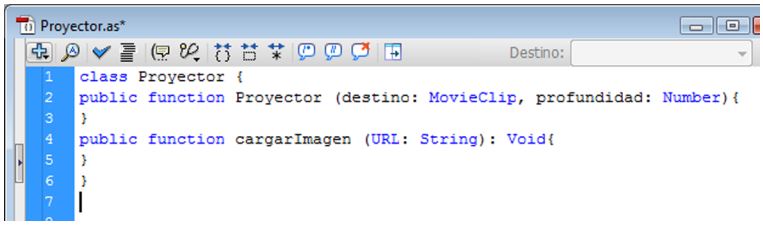
```
Proyector (destino: MovieClip, profundidad: Number)
```

Així, doncs, ara la nostra classe estaria formada pel codi següent:



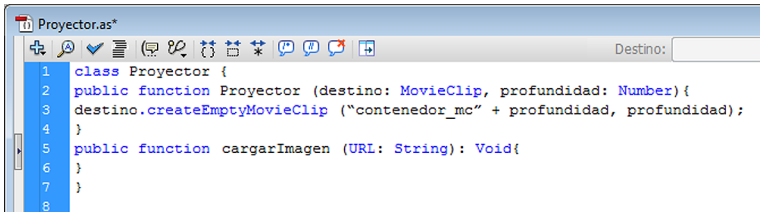
Bé, fins aquí ja tenim l'esquelet bàsic, encara que no operatiu, del que serà la nostra classe. Refinem-lo ara una mica per poder-hi començar a construir a sobre.

Uns paràgrafs més amunt dèiem que els paràmetres de la nostra classe han de ser accessibles, és a dir, qualsevol persona que estigui treballant en un projecte Flash i la vulgui usar ha de poder cridar-los. Això implicarà dues coses: la primera és que haurem d'establir el nostre codi com a **funció**, i la segona és que aquesta funció haurà de ser **públicament accessible**. D'acord amb aquestes premisses introduirem uns petits canvis en el nostre codi per deixar-lo de la manera següent:



Fixeu-vos que hem afegit dos parells més de **claus** dins dels quals allotjarem els cossos de codificació tant de la funció del constructor com del mètode de càrrega d'imatge.

Ara que ja sabem quin constructor tindrà la nova classe i quin mètode volem usar, és el moment de passar a codificar la funció del constructor. Hem dit que volem crear un clip de pel·lícula en què allotjarem la imatge carregada i que allotjarem aquest clip en una destinació i una profunditat prèviament definides. Vist això podríem dir el següent:

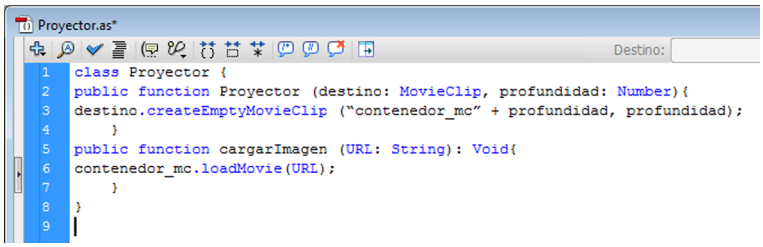


Això crearà un clip virtual per cada nivell de profunditat en el qual disposem imatges. Tots els nous clips que es vagin creant s'anomenaran **contenedor_mcX**, en què la *X* serà el nivell de profunditat en el qual queda allotjat. Si tenim en compte que allotjarem un únic clip per nivell resultarà impossible trobar un nom d'instància repetit.

Ara que ja tenim el constructor fet és el torn del mètode cargarImagen. Aquest mètode en realitat farà una crida a la funció loadMovie continguda en el Flash, i el codi de càrrega podria ser similar al següent:

```
contenedor_mc. loadMovie (URL);
```

Això deixaria el codi de la classe que estem construint de la manera següent:

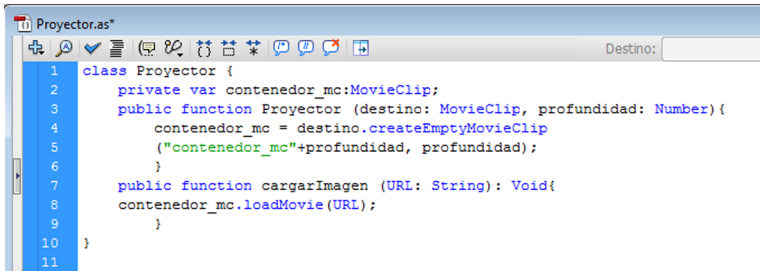


Si ara observem el codi de la nostra classe veurem que el mètode cargarImagen no té manera d'accedir al **clip buit** creat pel constructor, ja que aquest posarà un nombre final que el diferenciarà en cada cas. Així, doncs, haurem d'alterar el constructor de manera que prèviament emmagatzemi una referència al clip buit en una propietat d'instància i així es puguin comunicar tant el constructor com la funció.

Com a conseqüència del fet que aquesta referència no haurà de ser accessible des de la finestra d'accions d'un projecte Flash, la declararem com a **referència privada** en lloc de fer-ho com referència pública.

```
private var contenedor_mc: MovieClip;
```

Ara que l'estructura de classe Proyector ja comença a estar a punt hauria de tenir un aspecte com el següent:



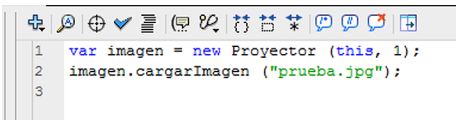
Aquest pot ser un bon moment per a comprovar com funciona la nostra nova classe i així poder verificar també que realment un projecte Flash i aquest arxiu són capaços de comunicar-se.

1) Creeu una carpeta en el vostre ordinador i copieu-hi l'arxiu **Proyector.as**.

2) Tot seguit busqueu una imatge en format JPG i copieu-la dins de la carpeta que acabeu de crear.

3) Rebategeu la imatge copiada d'acord amb el nom **prueba.jpg**.

4) Ja per finalitzar obriu un nou projecte Flash, deseu-lo en la mateixa carpeta, i en la finestra d'accions escriviu el codi següent:



```
1  var imagen = new Proyector (this, 1);
2  imagen.cargarImagen ("prueba.jpg");
3
```

Amb aquest codi el que fem és crear una instància de la classe Proyector, que situem en el nivell 1 de l'escenari del Flash Player, i la igualem mitjançant el nom de la variable imatge.

5) Immediatament després de crear aquesta instància hi carreguem la imatge **prueba.jpg**.

6) Abans de provar la pel·lícula és necessari que deseu l'arxiu **.fla** dins de la mateixa carpeta en la qual es troba la imatge **prueba.jpg** i l'arxiu **Proyector.as**.

7) Una vegada fet això podeu provar la pel·lícula. S'hauria de carregar la imatge correctament.

Ara que ja tenim en funcionament una primera versió de la nostra nova classe pot ser el moment de començar a afegir alguna funcionalitat més per a fer que el nostre projector presenti un aspecte més personalitzat. Això serà el que farem en el pròxim apartat.

3. Afegir funcionalitats a la nostra classe

En l'apartat anterior hem après a crear la nostra pròpia classe, vegem ara algunes coses que podríem fer per a millorar-la afegint algunes possibles funcionalitats com poden ser:

- Col·locar un marc negre al voltant de la imatge,
- mostrar només una part de la imatge de manera que ens permeti reenquadrar-la,
- reposicionar la zona en la qual es mostrarà la imatge,
- moure la part d'imatge visible dins de la zona destinada al visionament,
- etc.

Ens quedarem amb la primera opció:

> Col·locar un marc negre al voltant com un paspartú[5] (*passe-partout*) de la imatge.

(5)Un paspartú (*passe-partout*) és un marc, generalment de color blanc o negre, que té com a funció ajudar a centrar la vista en la imatge.

Per fer el que ens hem proposat necessitarem un clip creat en **temps d'execució** que contingui tots els elements necessaris perquè funcioni el nostre projector.

Si ara donem una mirada a la classe que havíem creat en el capítol anterior, veurem que incorporàvem un clip que anomenàvem ***contenedor_mc***, i és sabut que un clip pot allotjar a dins a altres clips. Així, doncs, resulta evident que la millor manera de modificar la nostra classe per a implementar aquestes funcionalitats és fer que **contenedor_mc** allotgi a dins tot allò que necessitem i que es correspon amb allò que detallem a continuació.

Contingut de **contenedor_mc:**

- Un clip de pel·lícula que faci les funcions de paspartú, que a partir d'ara anomenarem **pass_mc**.
- Un clip de pel·lícula que contingui la imatge que es dirà **imagen_mc**.

Si tornem a fer una ullada a la nostra classe veurem que abans havíem creat el clip del contenidor dins de la funció constructora, i aquest era aquest el codi que usàvem per a això:

```
contenedor_mc = destino.createEmptyMovieClip (contenedor_mc"+profundidad, profundidad);
```

Probablement un programador experimentat continuaria treballant en aquesta línia i crearia el clip del paspartú. Tanmateix, aquesta vegada no anirem per aquesta via sinó que actuarem definint mètodes interns que ens permetin manejar la creació dinàmica de diversos clips alhora. Això, a més de permetre'ns veure els anomenats *mètodes privats de classe* ens facilitarà, d'una banda, el procés de prova en permetre **verificar** cada mètode de manera individualitzada i, de l'altra, farà el codi molt més entenedor.

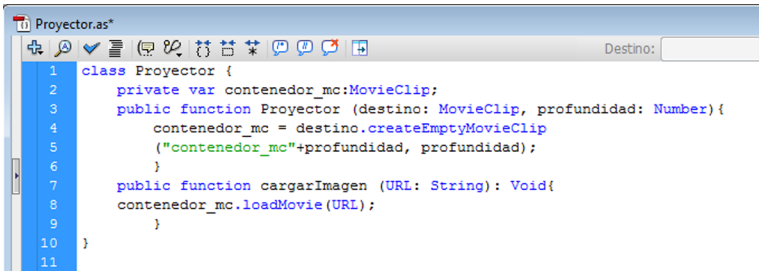
Dit això, comencem per posar un nom a cada mètode que volem usar i les propietats que necessitarem que efectuïn aquests mètodes. Aquesta és potser la tasca més important, ja que a més de definir allò que necessitem ens obliga a elaborar un **mapa conceptual** complet del que volem aconseguir.

Mètodes necessaris	
Què volem que faci?	**Nom de la funció**
Crear el clip contenidor.	creaContenedor
Crear el clip de paspartú.	creaPass
Crear el clip d'imatge.	crealmagen
Serà l'encarregat de relacionar tots els mètodes entre ells i acabar traçant la interfície final.	creaInterficie

Propietats que hauran de subministrar els mètodes anteriors	
Per a què és necessària?	**Nom de la propietat**
Ens indicarà la profunditat en la qual pass_mc s'ha d'allotjar dins de contenedor_mc.	profPass
Ens indicarà la profunditat en la qual imagen_mc s'ha d'allotjar dins de contenedor_mc.	profImagen
Farà referència al contenidor principal que contindrà tots els clips usats en la instància de Proyector.	contenedor_mc
Tal com especifica la classe que hem construït en el capítol anterior, funcionarà com a referència al clip que contindrà dins seu al clip contenedor_mc.	destino_mc
Ens indicarà la profunditat a què el clip contenedor_mc ha de ser creat dins de destino_mc.	profContenedor
Establirà el color que tindrà el paspartú que afegirem a la imatge.	colorPass

Ara que ja sabem què és exactament el que necessitem, podem començar a reescriure la nostra classe per a implementar-ne les noves funcionalitats.

Aquest era el codi al qual havíem arribat al final de l'apartat anterior:



Distribuirem la tasca de reformar la classe en quatre etapes:

1) Decretar les variables necessàries.

2) Reformar la funció del constructor principal.

3) Establir els constructors individualitzats.

4) Reescriure el mètode de càrrega de la imatge.

3.1. Primera etapa: decretar les variables necessàries

Col·locarem les variables dins del codi justament abans que comenci a decretar-se la funció del constructor. Començarem per decretar el **tipus de dades** de la variable **destino_mc**. Indicarem que el seu contingut pertanyerà a la classe **MovieClip**, per a això afegirem la línia següent:

```
class Proyector {
   private var contenedor_mc:MovieClip;
   private var destino_mc:MovieClip;
```

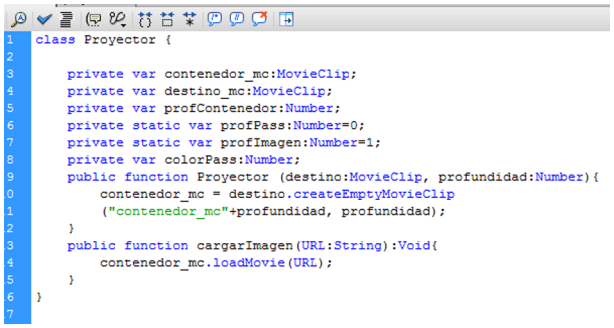
Continuem decretant el tipus de dades de totes les variables que necessitem, és a dir, una per cada element del contenidor i una d'única que ens permeti establir el color del paspartú. Com que el seu contingut serà numèric, en lloc d'associar-les a **MovieClip** les hem associat a **Number**. Recordeu en aquest sentit quan en apartats anteriors parlàvem dels avantatges de declarar quin tipus de dades seran vàlides en cada cas.

```
class Proyector {
   private var contenedor_mc:MovieClip;
   private var destino_mc:MovieClip;
   private var profContenedor:Number;
   private var profPass: Number;
   private var profImagen: Number;
   private var colorPass:Number;
```

Atès que també hem de definir la profunditat de cada clip que haurà d'anar allotjat dins del contenidor i que aquesta profunditat sempre mantindrà el mateix ordre, podem aprofitar per a indicar-la dins d'aquestes mateixes variables de la manera següent:

```
class Proyector {
   private var contenedor_mc:MovieClip;
   private var destino_mc:MovieClip;
   private var profContenedor:Number;
   private static var profPass: Number =0;
   private static var profImagen: Number =1;
   private var colorPass:Number;
```

La sentència **static** farà que aquestes variables es creïn una única vegada i ho facin segons el valor especificat en la igualtat.



```
class Proyector {

   private var contenedor_mc:MovieClip;
   private var destino_mc:MovieClip;
   private var profContenedor:Number;
   private static var profPass:Number=0;
   private static var profImagen:Number=1;
   private var colorPass:Number;
   public function Proyector (destino:MovieClip, profundidad:Number){
       contenedor_mc = destino.createEmptyMovieClip
       ("contenedor_mc"+profundidad, profundidad);
   }
   public function cargarImagen(URL:String):Void{
       contenedor_mc.loadMovie(URL);
   }
}
```

Amb això ja hem definit totes les variables que necessitarem, és doncs el moment d'iniciar la segona etapa.

3.2. Segona etapa: reformar la funció del constructor principal

Abans de començar fem una ullada a com teníem el codi inicial.

```
public function Proyector (destino:MovieClip, profundidad:Number) {
   contenedor_mc = destino.createEmptyMovieClip (contenedor_mc"+
profundidad, profundidad);
   }
```

Per començar ens quedarem amb els **atributs** del Proyector.

```
public function Proyector (destino:MovieClip, profundidad:Number)
```

Recordem que aquesta part és pública i, per tant, permetrà les especificacions que indiquem en la finestra d'accions del nostre projecte Flash. Sembla, doncs, un lloc adequat per a crear el buit que permeti especificar el **color** del paspartú. Així, doncs, introduirem aquest paràmetre en aquesta zona, de manera que ara pot quedar de la manera següent:

```
public function Proyector (destino:MovieClip,profundidad:Number,
colorPass:Number)
```

En aquesta part a més hi ha dos elements més que podem establir i que fins ara no hem esmentat. Si abans heu comprovat el vostre projector segur que haureu vist que la imatge carregada sempre es col·loca en la **coordenada (0,0)**. Aquesta és una de les coses que podem evitar si des d'aquí decretem la necessitat d'establir, des de l'arxiu Flash, paràmetres que es corresponguin amb els valors respecte a l'eix *X* i a l'eix *Y*.

A més d'això, quan l'aplicació funcioni, també serà necessari poder definir la **mesura** que volem que tingui el paspartú perquè s'adapti a les diferents imatges. Així, doncs, també hem de crear les mesures d'aquest representades en valors de **píxels**.

Un cop valorades aquestes dues noves necessitats, les col·locarem també en el nostre constructor perquè quedi així:

```
public function Proyector (destino:MovieClip,profundidad:Number,
colorPass:Number, x:Number, y:Number, anchura:Number, altura:Number)
```

- ***X*** i ***Y*** seran les coordenades que ocupi **contenedor_mc** tenint en compte que qualsevol clip creat dinàmicament col·loca el seu punt de registre en la seva coordenada (0,0).

- **Amplada** i **altura** seran els valors en píxels que tindrà **pass_mc**.

Abans de continuar avançant repassem un moment l'estat actual que hauria de tenir el nostre codi de classe.

```
1  class Proyector {
2
3      private var contenedor_mc:MovieClip;
4      private var destino_mc:MovieClip;
5      private var profContenedor:Number;
6      private static var profPass:Number=0;
7      private static var profImagen:Number=1;
8      private var colorPass:Number;
9      public function Proyector (destino:MovieClip,
0                                 profundidad:Number,
1                                 colorPass:Number,
2                                 x:Number,
3                                 y:Number,
4                                 anchura:Number,
5                                 altura:Number ){
6          contenedor_mc = destino.createEmptyMovieClip
7          ("contenedor_mc"+profundidad, profundidad);
8      }
9      public function cargarImagen(URL:String):Void{
0          contenedor_mc.loadMovie(URL);
1      }
2  }
```

Com a conseqüència que **contenedor_mc** ha sofert un canvi de rol perquè ha d'incloure no solament la imatge sinó dos clips uns dels quals contindrà la imatge, es fa imprescindible reformar tota la part del constructor.

Per a això, dins de la funció del constructor, haurem d'assignar valors a les propietats que tindrà el nou constructor i també establirem els valors que haurà de prendre el mètode **creaInterficie**, el qual recordem que serà el responsable principal de crear i allotjar tant **contenedor_mc** com els clips que conté.

Atenció

Recordeu que ja havíem definit la funció creaInterficie quan fèiem una relació amb tots els mètodes que necessitaríem. En aquell moment ja es va dir que aquest mètode tenia com a funció cridar els quatre mètodes individuals també definits llavors.

Així, doncs, el codi de la nostra classe haurà de sofrir els canvis següents:

```
class Proyector {
   private var contenedor_mc:MovieClip;
   private var destino_mc:MovieClip;
   private var profContenedor:Number;
   private static var profPass: Number =0;
   private static var profImagen: Number =1;
   private var colorPass:Number;
   public function Proyector(destino:MovieClip, profundidad:Number, colorPass:Number,
   x:Number, y:Number, anchura:Number, altura:Number) {
   contenedor_mc = destino.createEmptyMovieClip("contenedor_mc"+profundidad,
   profundidad);
      destino_mc = destino;
      profContenedor = profunditat;
      this.colorPass = colorPass;
      creaInterficie (x, i, amplada, altura);
   }
   public function cargarImagen(URL:String):Void {
      contenedor_mc.loadMovie(URL);
   }
   }
```

Ara, el codi tindrà l'aspecte següent.

```
class Proyector {

    private var contenedor_mc:MovieClip;
    private var destino_mc:MovieClip;
    private var profContenedor:Number;
    private static var profPass:Number=0;
    private static var profImagen:Number=1;
    private var colorPass:Number;
    public function Proyector (destino:MovieClip,
                               profundidad:Number,
                               colorPass:Number,
                               x:Number,
                               y:Number,
                               anchura:Number,
                               altura:Number ){
    destino_mc = destino;
    profContenedor = profundidad;
    this.colorPass = colorPass;
    creaInterficie (x, y, anchura, altura);
    }
    public function cargarImagen(URL:String):Void{
        contenedor_mc.loadMovie(URL);
    }
}
```

3.3. Tercera etapa: establir els mètodes individualitzats

Si recordem el codi que teníem anteriorment en aquesta part, veurem que igual com passava en la part del constructor necessitarem canviar-lo de manera força radical, ja que com en el cas de l'etapa anterior el contingut de **contenedor_mc** ha variat substancialment.

```
public function cargarImagen(URL:String):Void {
   contenedor_mc.loadMovie(URL);
   }
```

Començarem per definir el **mètode principal**, que serà el que s'encarregarà de delegar la seva funció als altres tres mètodes que anteriorment, quan analitzàvem les noves funcionalitats, dèiem que necessitàvem i que es corresponen amb creaContenedor, creaPass i creaImagen.

El mètode principal funcionarà únicament com a oient dels resultats d'aplicar els altres quatre mètodes.

```
private function creaInterficie (x:Number, y:Number, anchura:Number, altura:Number): Void {
   creaContenedor (x,y);
   creaPass (amplada, altura);
   creaImagen ();
   }
```

És convenient observar els **paràmetres** que podrem establir en cada una d'aquestes funcions que desenvolupem posteriorment.

- La funció **creaContenedor** necessitarà els paràmetres ***x*** i ***y*** per a determinar així les seves coordenades. Recordem que aquesta funció crearà el con-

tenidor i, per tant, les coordenades *x* i *y* seran en realitat les coordenades en les quals s'allotjarà el resultat visual del nostre Proyector.

- La funció **creaImagen** serà **exempta de paràmetres** perquè el que realment farem amb aquesta funció és simplement afegir el clip amb la imatge dins del clip contenidor. Per a dur a terme aquest procés no es necessita cap paràmetre.

- Finalment, la funció **creaPass** necessitarà els paràmetres d'**amplada** i **altura** per a determinar les dimensions de la màscara de retallada.

Ara és el torn de definir el primer mètode inclòs en el mètode anterior, el mètode **creaContenedor**. Aquest mètode pren part en el codi del constructor anterior, ja que en el fons la seva missió continuarà essent la mateixa. Així, doncs, anteriorment, dins de la funció del constructor, teníem el codi que es mostra a continuació i que podrem reaprofitar en part, però aquesta vegada el col·locarem dins d'una altra funció, la funció creaContenedor, que és la que hi correspon segons la nostra planificació actual.

```
private function creaContenedor (x:Number, y:Number): Void {
   contenedor_mc = destino.createEmptyMovieClip("contenedor_mc"+profundidad, profundidad);
   }
```

Si mirem unes línies més amunt veurem que aquesta funció necessita els paràmetres *x* i *y*, els quals serviran per a posicionar el clip contenidor dins de l'escenari Flash. Així, doncs, afegirem dues línies al codi d'aquest mètode perquè això sigui possible.

```
private function creaContenedor (x:Number, y:Number): Void {
   contenedor_mc = destino.createEmptyMovieClip("contenedor_mc"+profundidad, profundidad);
   contenedor_mc._x = x;
   contenedor_mc._y = y;
   }
```

Ara que ja tenim establerta la funció que crearà el contenidor dels tres clips, establirem la funció específica per a la creació de cada un dels seus clips interns començant, per exemple, amb la funció que permetrà crear el clip en què s'allotjarà la imatge pròpiament dita.

```
private function creaImagen (): Void {
   contenedor_mc.createEmptyMovieClip (imagen_mc, profImagen);
   }
```

Aquesta vegada no fa falta especificar res més, ja que tan sols volem carregar la imatge dins del clip **imagen_mc**.

Anem ara a la funció de creació de **creaPass**. En aquest cas crearem el clip del marc de la mateixa manera que quan incorporàvem clips de forma dinàmica al llarg del primer apartat; això és, indicant el color de farciment, definint el gruix de línia i el seu color, podem prescindir d'introduir altres dades com el grau de transparència.

Coordenada (0,0)

És important observar que la posició de referència del clip **pass_mc** serà la coordenada (0,0). Fixeu-vos que aquesta coordenada no fa referència a la posició que ocupa el clip **contenedor_mc** en l'escenari, les coordenades del qual ja hem vist que estan determinades per les variables *x* i *Y*, sinó a la que ocupa ell mateix dins del clip **contenedor_mc**.

També es pot destacar que les mesures en píxels de **pass_mc** no s'estableixen de manera absoluta sinó que es recolzen en unes altres dues variables, amplada i altura, per a així permetre que s'adaptin a la mesura individual de cada imatge.

Així, doncs, el codi de la creació del marc quedaria així:

```
private function creaPass (anchura:Number, altura:Number): Void {
   contenedor_mc.createEmptyMovieClip (pass_mc, profPass);
   contenedor_mc.pass_mc.beginFill (0x000000);
   contenedor_mc.pass_mc.lineStyle (1, 0x000000);
   contenedor_mc.pass_mc.moveTo (0, 0);
   contenedor_mc.pass_mc.lineTo (0, altura);
   contenedor_mc.pass_mc.lineTo (altura, anchura);
   contenedor_mc.pass_mc.lineTo (anchura, 0);
   contenedor_mc.pass_mc.lineTo (0, 0);
   contenedor_mc.pass_mc.endFill ();
   }
```

Observeu la forma que té ara la notació. Podreu observar que tant **imagen_mc** com **pass_mc** s'estan col·locant ara dins de **contenedor_mc**. Serà en aquest **clip pare** en el qual s'establiran els nivells de profunditat que ocuparà cada un.

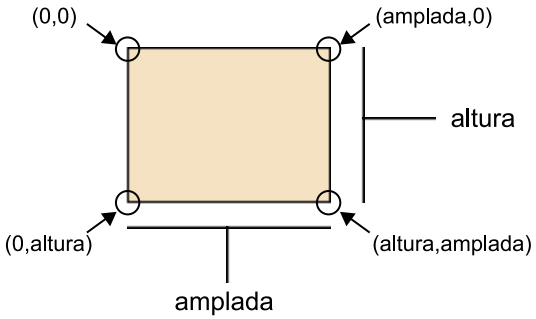
Abans de continuar amb la quarta i última etapa ens aturarem un moment per observar què fa cada un dels **mètodes** que conformen el codi que crea el **marc de la imatge:**

- **beginFill**. Indica el començament d'un nou traç de dibuix. Si hi ha un traçat obert i té un farciment de color associat, aquest traçat es tancarà amb el tipus de línia definit a **lineStyle** i s'omplirà amb el color especificat en aquest mètode.

- **lineStyle**. Especifica l'estil de línia, el gruix en píxels i el color que usaran les crides posteriors a **lineTo**().

- **moveTo**. Mou la posició de dibuix actual a les coordenades ***X*** i ***Y*** indicades.

- **lineTo**. Dibuixa una línia utilitzant l'estil de línia definit a lineStyle des de la posició en la qual es trobi fins a la coordenada especificada. En el cas del traçat que es descriu en el codi anterior, les posicions que recorrerien les diferents crides a aquest mètode serien les que es poden veure en la imatge següent.



- **endFill**. Aplica el color de farciment especificat en el mètode **beginFill**() en el cas de farciments sòlids, o en **beginGradientFill** en el cas de farciments amb gradient de color.

3.4. Quarta etapa: reescriure el mètode de càrrega de la imatge

Començarem per recordar el codi que teníem escrit en l'apartat anterior i que és el mateix que encara tenim situat just al final de l'*script*.

```
public function cargarImagen(URL:String):Void {
   contenedor_mc.loadMovie(URL);
   }
```

Si ho observem veurem que en general continua essent vàlid i l'únic que necessita és un redireccionament del lloc de càrrega, ja que ara no s'ha de carregar al clip **contenedor_mc** sinó al clip **imagen_mc** que està dins d'aquell.

```
public function cargarImagen(URL:String):Void {
      contenedor_mc.imagen_mc.loadMovie(URL);
   }
```

```
class Proyector {

    private var contenedor_mc:MovieClip;
    private var destino_mc:MovieClip;
    private var profContenedor:Number;
    private static var profPass:Number=0;
    private static var profImagen:Number=1;
    private var colorPass:Number;

    public function Proyector(destino:MovieClip,
                              profundidad:Number,
                              colorPass:Number,
                              x:Number,
                              y:Number,
                              anchura:Number,
                              altura:Number) {
        destino_mc=destino;
        profContenedor=profundidad;
        this.colorPass=colorPass;
        creaInterficie(x,y,anchura,altura);
    }
    private function creaInterficie(x:Number,
                                    y:Number,
                                    anchura:Number,
                                    altura:Number):Void {
        creaContenedor(x,y);
        creaImagen();
        creaPass(anchura,altura);
    }
    private function creaContenedor(x:Number,y:Number):Void {
        contenedor_mc=destino_mc.createEmptyMovieClip
        ("contenedor_mc" + profContenedor, profContenedor);
        contenedor_mc._x=x;
        contenedor_mc._y=y;
    }


    private function creaPass(anchura:Number,altura:Number):Void {
        contenedor_mc.createEmptyMovieClip("pass_mc",profPass);
        contenedor_mc.pass_mc.lineStyle(30,0xFF000000,100,true,
                                        "none","square","miter");
        contenedor_mc.pass_mc.moveTo(0,0);
        contenedor_mc.pass_mc.lineTo(0,altura);
        contenedor_mc.pass_mc.lineTo(anchura,altura);
        contenedor_mc.pass_mc.lineTo(anchura,0);
        contenedor_mc.pass_mc.lineTo(0,0);
        contenedor_mc.pass_mc.endFill();
    }

    private function creaImagen():Void {
        contenedor_mc.createEmptyMovieClip("imagen_mc",profImagen);
    }
```

Amb la classe ja finalitzada, és el moment de variar el codi del nostre projecte **.fla** perquè s'adapti als canvis fets en el cos de la classe.

Recordem el codi que teníem en la línia de temps de **proyector.fla**

```
1  var imagen = new Proyector (this, 1);
2  imagen.cargarImagen ("prueba.jpg");
3
```

Atès que la funció cargarImagen ha variat en nombre de paràmetres, serà convenient reflectir també aquí els canvis. Així, doncs, ara no solament indicarem la destinació i la profunditat en què volem ubicar el clip, sinó que també indi-

carem els paràmetres següents: el color, la posició respecte a l'eix *X*, la posició respecte a l'eix *Y*, i l'amplada i l'altura de la imatge. Vist això, el nostre codi podria quedar, per exemple, de la manera següent:

```
var imagen: Proyector = new Proyector(this,
                                      1,
                                      0x000000,
                                      10,
                                      10,
                                      500,
                                      464);
imagen.cargarImagen("prueba.jpg");
```

En pròxims apartats veurem algunes classes que ja van incorporades en el Flash i que estan a punt per a fer servir.

4. La classe BitmapData

Fins ara hem estat usant elements de la classe **MovieClip**, fins i tot quan en els dos apartats anteriors ens dedicàvem a crear la nostra classe, ens recolzàvem majoritàriament en elements d'aquesta classe.

Com hem vist fins ara una de les possibilitats de treball més potents de la classe **MovieClip** és que permeten representar **objectes d'imatge** i carregar-los en **temps real**.

Fins aquí podríem pensar que treballar ara amb la classe **BitmapData** és una molt semblant. Si bé això no és del tot fals, ja que **BitmapData** comparteix i/o complementa molts components amb la classe **MovieClip**, si que dista molt de les possibilitats específiques d'aquesta classe.

BitmapData no és una classe pensada per a la representació directa d'un mapa de bits, sinó que la seva funció principal està basada en la capacitat de representació i manipulació d'un objecte de mapa de bits que prèviament s'ha carregat en Flash Player.

En crear una nova instància de la classe, s'emmagatzema una imatge de bits en blanc en la memòria de l'ordinador que es pot omplir amb el mapa de bits que volem en cada moment. Això ens permetrà manipular aquest mapa de bits amb els diversos mètodes de la classe **BitmapData**.

4.1. Mapes de bits emmagatzemats en Flash Player

Parlem de mapes de bits com la descripció d'una imatge mitjançant una quadrícula de valors de color.

Cada element de la quadrícula descrita en un mapa de bits representa un píxel. Cada píxel s'omple amb un valor de color determinat i tot el conjunt de píxels forma la representació de la imatge.

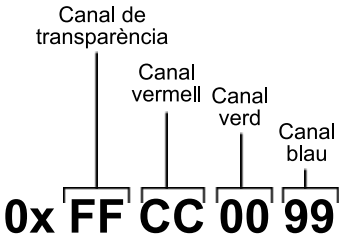
Els mapes de bits s'emmagatzemen en Flash Player amb una **profunditat de color** de 32 bits. Els valors de color utilitzats en combinació amb la classe BitmapData s'han de representar en ActionScript mitjançant un **nombre hexadecimal** de 32 bits. Un nombre hexadecimal de 32 bits és una seqüència de quatre parells de dígits hexadecimals. Cada parell hexadecimal defineix la intensitat de cada un dels quatre canals de color (alfa, vermell, verd i blau).

Representació hexadecimal de la intensitat d'un canal de color

La intensitat d'un canal de color és una representació hexadecimal d'un nombre decimal comprès entre 0 i 255; FF és la intensitat total (255), mentre que 00 equival a l'absència de color en el canal (0).

És important observar que el valor de qualsevol canal s'ha de completar amb zeros perquè tingui una longitud de dos dígits. Així, per exemple, si un canal ha de tenir un valor 3 hem d'escriure 03. Això garantirà que qualsevol nombre hexadecimal sempre sigui compost de vuit dígits.

També és convenient assegurar-se que s'especifica el prefix del nombre hexadecimal, 0x. Per exemple, blanc (intensitat total en tots els canals) es representa amb la notació hexadecimal següent: 0xFFFFFFFF. Per contra, negre és el valor oposat; no té color en els canals vermell, verd i blau: 0xFF000000. Fixeu-vos que el canal alfa (el primer parell) continua tenint intensitat total (FF). Intensitat total del canal alfa significa que no hi ha alfa (FF), mentre que absència d'intensitat (00) significa que l'alfa és complet. Així, el valor de color d'un píxel transparent de 32 bits començarà sempre de la manera següent: 0x00...



4.2. Creació d'un mapa de bits amb ActionScript

La classe **BitmapData** es troba ubicada dins del paquet **flash.display**. Un petit truc per a no haver d'escriure la ruta completa de la classe cada vegada que vulguem crear una nova instància és importar primer el paquet. Això podem fer-ho mitjançant la línia de codi següent:

```
1  import flash.display.BitmapData;
```

A partir d'aquí podrem crear directament una instància de la classe afegint el següent:

```
1  import flash.display.BitmapData;
2  myBitmap = new BitmapData(anchura,
3                            altura,
4                            transparencia,
5                            color);
```

Com es pot veure en la imatge anterior, el constructor de la classe **BitmapData** accepta quatre arguments:

- amplada,
- altura,
- transparència i

- color.

Els arguments **amplada** i **altura** serveixen per a especificar les dimensions del mapa de bits que es crearà en memòria.

L'argument de **transparència** només admet valors booleans de vertader o fals (*true*/*false*). Aquest paràmetre especifica si el mapa de bits tindrà transparència o no.

L'argument **color** s'utilitza per a especificar el color de 32 bits predeterminat del mapa de bits. Aquest argument es pot mostrar de dues maneres en funció de si hem definit el paràmetre de transparència en *true* o en *false*. En cas que aquest paràmetre sigui vertader (*true*) serà necessari establir un nombre hexadecimal de 32 bits. Si per contra establim el paràmetre de transparència en fals es podran ometre els vuit primers bits o dos dígits del nombre hexadecimal.

Així, per exemple, si suposem que volem crear una imatge negra totalment opaca, sense cap transparència, de 90 píxels d'amplada per 60 píxels d'altura, podrem usar indistintament els codis següents:

```
import flash.display.BitmapData;
myBitmap = new BitmapData(90, 60, false, 0xFF000000);

import flash.display.BitmapData;
myBitmap = new BitmapData(90, 60, false, 0x000000);
```

Fixeu-vos en la diferència de notació entre l'un i l'altre nombre hexadecimal.

Atenció

Treballar amb instàncies de BitmapData pot ser molt interessant, però té com a contrapartida que aquest tipus d'objectes consumeixen molta memòria RAM de l'ordinador en què es reprodueix l'arxiu. Per cada píxel d'un mapa de bits emmagatzemat s'empren 4 bytes de memòria RAM, així, per exemple, si creem un mapa de bits[6] de 720 × 576 (format estàndard en televisors PAL), estarem consumint una mica més d'1,5 MB de memòria RAM.

(6)Les dimensions màximes d'un mapa de bits en el Flash Player són 2.880 × 2.880 píxels. Un mapa de bits d'aquesta mida utilitza aproximadament 32 MB de RAM.

Com a conseqüència de l'anterior, quan ja no necessitem mostrar més un objecte BitmapData és recomanable **alliberar la memòria** que utilitza el mapa de bits. Per a això, el Flash disposa d'un mètode que permet fer exactament això i el qual s'usa amb notació al fotograma de la manera que es pot veure en la imatge següent, en què **miBitmap** no és altra cosa que el nom de la instància **BitmapData** que s'ha carregat prèviament.

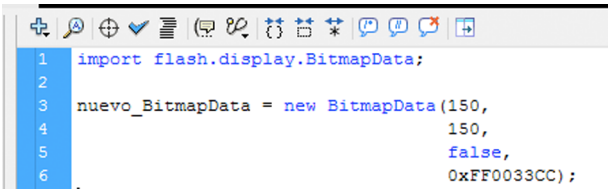
```
1 miBitmap.dispose();
```

4.3. Creació d'un primer mapa de bits de BitmapData

Com ja s'ha esmentat anteriorment el primer pas serà importar a la finestra d'accions del projecte Flash els components de la classe **BitmapData** que ja van inclosos en el Flash.

A continuació crearem el nou objecte **BitmapData** que heretarà totes les característiques de la classe **BitmapData**.

Fins aquí el nostre codi s'hauria d'assemblar al de la imatge següent.

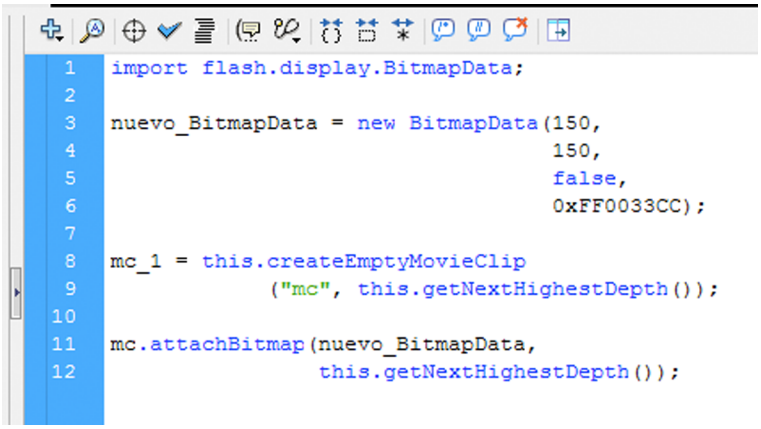


```
1 import flash.display.BitmapData;
2
3 nuevo_BitmapData = new BitmapData(150,
4                                   150,
5                                   false,
6                                   0xFF0033CC);
```

Ara haurem de crear el clip que farà de contenidor del mapa de bits esmentat. Per a això afegirem el fragment de codi següent:

```
mc_1 = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(nuevo_BitmapData, this.getNextHighestDepth());
```

Amb això ara el nostre codi serà el següent:



```
1  import flash.display.BitmapData;
2
3  nuevo_BitmapData = new BitmapData(150,
4                                    150,
5                                    false,
6                                    0xFF0033CC);
7
8  mc_1 = this.createEmptyMovieClip
9             ("mc", this.getNextHighestDepth());
10
11 mc.attachBitmap(nuevo_BitmapData,
12                 this.getNextHighestDepth());
```

Si ara provem el projecte Flash podrem veure que s'ha creat un mapa de bits de 150 per 150 píxels i color blau, el qual s'ha situat en la cantonada superior esquerra.

4.4. Comprovant alguns avantatges de BitmapData

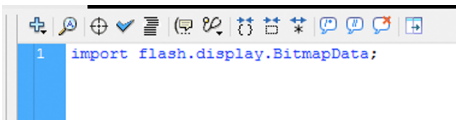
Si ens quedéssim amb el que hem vist fins ara, la veritat és que seríem molts els que pensaríem que l'après no val gaire pena, ja que el mateix es pot fer de maneres molt més senzilles, tanmateix la força dels mapes de bits de l'ActionScript va molt més enllà en permetre'ns entre altres coses afegir **efectes d'imatge** que

no es troben disponibles de cap altra manera en la interfície del Flash i que per a aconseguir-los haurem d'acudir a altres programaris d'edició d'imatges com poden ser **Photoshop** o **Fireworks** per posar algun exemple.

Ara veurem un exemple senzill de creació d'un mapa de bits al qual aplicarem un **filtre de soroll**. Si mireu els filtres disponibles en la interfície del Flash veureu que l'efecte esmentat no es troba disponible; tanmateix, sí que és en el paquet de la classe **BitmapData**.

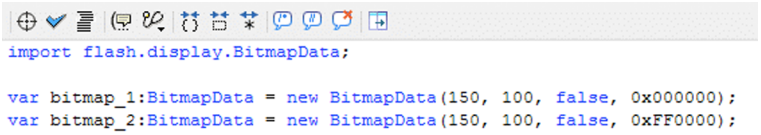
1) Obriu el Flash i creeu un **arxiu nou** de 300 × 100 píxels.

2) Ara aneu a la finestra d'***Acciones***. Iniciarem el nostre codi **important** els components de la classe **BitmapData** tal com ja hem fet en l'exemple anterior.



```
1 import flash.display.BitmapData;
```

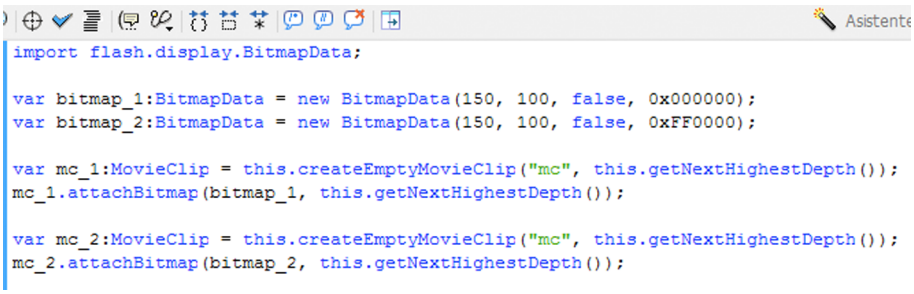
3) A continuació crearem **dos nous objectes** de **BitmapData** que mesurin 150 × 100 píxels, sense transparència i de colors diferents. El fet de donar-los colors diferents solament és per a diferenciar-los. Així, doncs, en el cas de la imatge inferior, crearem un primer mapa de bits de color negre i un segon mapa de bits de color vermell.



```
import flash.display.BitmapData;

var bitmap_1:BitmapData = new BitmapData(150, 100, false, 0x000000);
var bitmap_2:BitmapData = new BitmapData(150, 100, false, 0xFF0000);
```

Fixeu-vos que en la notació del color s'ha omès el primer parell de dígits corresponents a la transparència. Això és perquè, com ja s'ha comentat anteriorment, la condició del paràmetre de transparència ja és falsa, de manera que no tindria sentit donar ara transparència perquè tampoc no es mostraria.

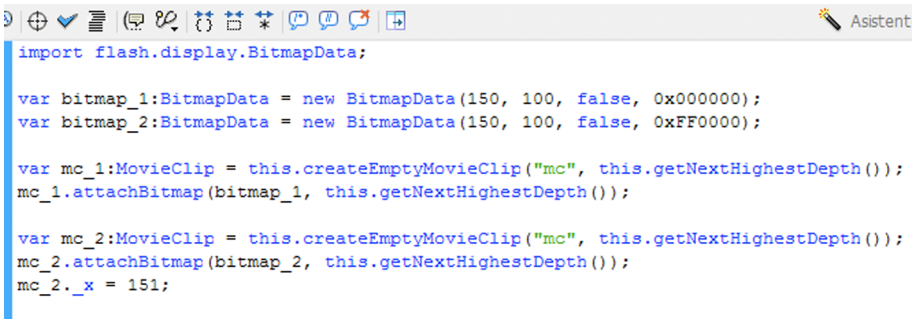
4) Tot seguit crearem els clips contenidors dels mapes de bits esmentats.



```
import flash.display.BitmapData;

var bitmap_1:BitmapData = new BitmapData(150, 100, false, 0x000000);
var bitmap_2:BitmapData = new BitmapData(150, 100, false, 0xFF0000);

var mc_1:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_1.attachBitmap(bitmap_1, this.getNextHighestDepth());

var mc_2:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_2.attachBitmap(bitmap_2, this.getNextHighestDepth());
```

Atès que no hem indicat les coordenades de creació dels mapes de bits esmentats, tots dos es desenvoluparan des de la coordenada (0,0), de manera que quedaran superposats a l'escenari. Perquè això no passi indicarem a continuació la posició de l'eix de les *X* en què volem situar el segon mapa de bits.

5) La posició en la qual col·locarem el segon mapa de bits serà la coordenada 151 de l'eix de les *X*, ja que tots dos clips tenen 150 píxels d'amplada.



Si ara proveu el vostre arxiu, veureu que efectivament s'han creat els dos mapes de bits i s'han col·locat correctament.

Apliquem ara el **filtre de soroll** que hem esmentat al principi de l'exercici.

Aquest filtre de soroll és un dels mètodes de la classe **BitmapData** i es troba identificat en l'ActioScript sota el nom de **perlinNoise**.

perlinNoise té una forma de declaració com la que es mostra a continuació i admet diversos paràmetres que expliquem breument a continuació.

```
public perlinNoise(baseX, baseY, numOctaves, randomSeed, stitch,
fractalNoise:, [channelOptions], [grayScale], [offsets])
```

- ***baseX* i *baseY***. Indiquen la posició respecte als eixos *X* i *Y* en la qual començarà a aplicar-se l'efecte de soroll.

- ***numOctaves***. Nombre d'octaves de soroll que es combinaran per a crear aquest soroll. Com més alt sigui el nombre més detallades seran les imatges, encara que com a contrapartida requeriran més temps de processament.

- ***randomSeed***. Indica un nombre d'inicialització aleatori que s'utilitzarà per a començar a generar el soroll de la imatge.

- ***stitch***. És un valor booleà. Si el valor és vertader (*true*), se suavitzaran les vores del soroll amb la finalitat de crear textures que es mostrin en mosaics totalment integrats entre ells.

- ***fractalNoise***. És un valor booleà. Si el valor és vertader genera soroll fractal, en cas contrari genera turbulències. Una imatge amb soroll fractal és més adequada per a mostrar núvols, mentre que una de turbulències mostrarà millor els moviments de l'onatge del mar.

- ***channelOptions***. És un nombre que indica els canals de color que es faran servir en la creació del soroll de la imatge. Els quatre valors corresponents als canals de color són els següents: l'1 indica el canal del color vermell, el 2 el del canal verd, el 4 el del blau, i el 8 la quantitat de transparència alfa. Les combinacions de color en el soroll es poden fer mitjançant la notació següent: 1 | 4. Aquesta notació indicarà que el soroll es generarà únicament amb els colors vermell i blau.

- ***grayScale***. És un valor booleà. Si el valor és vertader es crearà una imatge en escala de grisos utilitzant valors idèntics per a tots els canals de color vermell, verd i blau. El valor del canal alfa no quedarà afectat mantenint així el grau de transparència.

- ***offsets***. Es tracta d'una matriu de punts que correspon a desplaçaments *X* i *Y* per a cada octava. Cada punt de la matriu de desplaçament afecta una funció de soroll d'octava específica. De moment no entrarem a tocar aquest apartat, en temes posteriors tractarem de les matrius de transformació de manera extensa, especialment quan entrem a veure la classe **Matrix**, que també es troba integrada en el Flash.

Ara que ja sabem quins paràmetres admet **perlinNoise** és el moment de crear les funcions perquè es generin els sorolls que vulguem aplicar a cada un dels nostres mapes de bits.


```
import flash.display.BitmapData;

var bitmap_1:BitmapData = new BitmapData(150, 100, false, 0x000000);
var bitmap_2:BitmapData = new BitmapData(150, 100, false, 0xFF0000);

var mc_1:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_1.attachBitmap(bitmap_1, this.getNextHighestDepth());

var mc_2:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc_2.attachBitmap(bitmap_2, this.getNextHighestDepth());
mc_2._x = 151;

mc_1.onRollOver = function() {
    var randomNum:Number = Math.floor(Math.random() * 10);
        bitmap_1.perlinNoise(150, 100, 6, randomNum, false, true, 1, true, null);
}

mc_2.onRollOver = function() {
    var randomNum:Number = Math.floor(Math.random() * 10);
        bitmap_2.perlinNoise(150, 100, 4, randomNum, false, false, 1 | 4, false, null);
}
```

Si proveu ara el vostre projecte podreu comprovar com canvia l'aspecte de la imatge quan us situeu a sobre de cada rectangle. Observareu també que, si alterneu entre els rectangles, la imatge del soroll que es mostra canvia constantment.

Veureu també que un dels mapes de bits, el primer, es mostra únicament en escala de grisos. Això és degut al valor *true* del paràmetre **grayScale**.

L'altre mapa de bits crea el soroll a partir de dos canals de color, l'1 i el 4, que corresponen al vermell i el blau respectivament. Si canvieu aquests canals de color o n'afegiu de nous, veureu com canvia completament l'aspecte.

5. Las classes Rectangle i Point

A diferència de la classe BitmapData, les classes **rectangle** i **point** pertanyen al **grup geom** –geometria– ja inclòs en el Flash.

Els objectes **Rectangle** marquen una àrea definida per la posició que ocupa el seu angle superior esquerre, les coordenades (*x, y*) i la seva altura i amplada.

D'entre les diverses funcionalitats que permeten els objectes de la classe Rectangle n'hi ha dos que ens seran molt útils al llarg d'aquest i dels pròxims apartats: d'una banda, la capacitat que té l'objecte Rectangle per a utilitzar els filtres d'imatge disponibles en la classe **BitmapData** i, d'una altra banda, el fet de possibilitar la retallada visual de qualsevol instància de la classe **MovieClip** aplicant valors d'amplada, altura i desplaçament específics de la zona de visualització d'imatges, de manera que és possible, per exemple, mostrar únicament un fragment d'una imatge sense necessitat que hi hagi cap retallada en la mateixa imatge.

Per la seva banda, la classe **Point** permet, entre altres coses, accedir a informacions de píxels concrets, ja que aquesta classe està definida per una ubicació en un sistema de coordenades bidimensional *X* i *Y*.

5.1. El mètode Draw de la classe BitmapData

Aquest mètode permet crear un mapa de bits a partir de qualsevol instància situada en l'escenari, tant si aquest és una imatge fixa com un fotograma d'un clip de pel·lícula o un d'un objecte de vídeo. El seu ús traça una imatge d'origen en una altra imatge de destinació mitjançant el processador de vectors del Flash Player.

De manera predeterminada, la representació del nou mapa de bits respon a la imatge original sense aplicar cap transformació.

> Això vol dir que si prenem un mapa de bits d'un fotograma d'un objecte de vídeo que ha estat prèviament escalat i girat en l'escenari, la representació de mapa de bits no estarà ni escalada ni girada, sinó que mostrarà la imatge tal com es troba en l'objecte original. De la mateixa manera, si el mapa de bits pres és d'una imatge que conté un filtre, aquesta es mostrarà sense el filtre aplicat.

A simple vista, l'ús d'aquest mètode pot semblar limitat, ja que en la seva forma més simple el mètode **BitmapData.draw**() senzillament utilitza el processador de representació per a traçar una representació de mapa de bits de l'objecte especificat; no obstant això, quan aquest mètode es conjuga amb altres parà-

metres, es pot utilitzar per a aplicar tant transformacions d'imatge complicades com tipus de barreja o fusió entre diferents elements, com també per a transformar els seus valors de color.

Encara que el mètode **draw** de **BitmapData** admet diversos paràmetres en la seva construcció **Source**, és el primer i únic imprescindible, ja que s'hi indica l'objecte que es traçarà.

En l'exercici següent usarem aquest únic paràmetre deixant els altres per a exercicis posteriors.

5.2. Modificació de l'aspecte d'una imatge de bits amb Actionscript

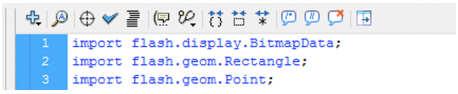
Les transformacions bàsiques que podem aplicar a un clip de pel·lícula són canviar la posició, la rotació, el grau de transparència, la visibilitat o l'escala. Tanmateix, si volem distorsionar un clip per a simular un **efecte de perspectiva** no podrem fer-ho directament amb les eines genèriques que ens proporciona el Flash.

Al llarg d'aquest apartat aprendrem alguns usos bàsics de la classe **BitmapData** en combinació amb les altres dues classes esmentades a l'inici d'aquest capítol. Integrarem aquestes tres classes en un únic projecte en el qual, a partir d'una imatge de format rectangular, simularem un efecte de perspectiva. Aplicarem aquest efecte en un clip carregat en temps d'execució, la qual cosa ens permetrà a més que en realitat el clip original resti amb l'aspecte original. Així, doncs, el que farem en realitat serà crear un nou clip que contindrà un objecte **BitmapData** amb la distorsió aplicada mentre ocultem el clip que no conté la distorsió. La percepció que l'usuari tindrà d'això és que realment s'ha modificat la imatge original.

1) Aneu a la carpeta de recursos de l'assignatura i obriu l'arxiu **distorsión.fla**.

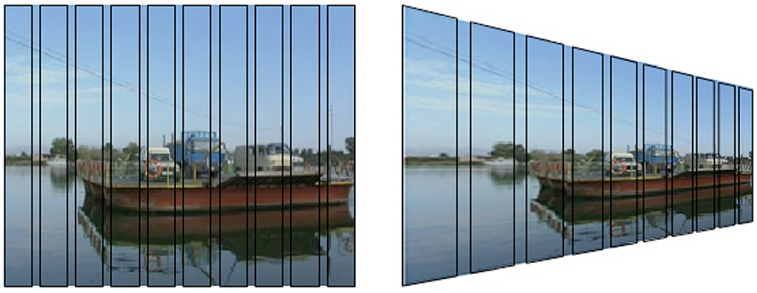
2) Observeu que en l'escenari hi ha una imatge a dins d'un clip de pel·lícula que hem identificat amb el nom ***imagen*** en la finestra de ***Propiedades***.

3) Tal com hem esmentat abans, per a iniciar el treball, haurem de començar per importar les tres classes ja incorporades en la finestra d'***Acciones*** de l'arxiu **.fla** del nostre projecte.



```
1 import flash.display.BitmapData;
2 import flash.geom.Rectangle;
3 import flash.geom.Point;
```

Ara ja estem en condicions de començar i el primer que farem és plantejar-nos com funcionaria conceptualment la transformació de la imatge que volem fer. Si observem la imatge següent veurem que en realitat el que volem fer és encongir o estirar els píxels per a simular l'efecte de perspectiva.



Un sistema per a dur a terme aquesta transformació és considerar la imatge com si estigués continguda partint d'elements verticals tal com mostra la imatge. El primer que hem de definir és l'objecte que contindrà la quantitat de deformació que tindrien aquestes presumptes columnes verticals. Així, doncs, començarem a crear aquest objecte i per a això ens basarem en la propietat ***prototype*** de la classe **MovieClip**.

Quan es crea un **objecte de funció**[7] mitjançant la propietat ***prototype***, aquesta propietat queda associada automàticament a l'objecte de funció. Aquesta propietat es considera estàtica perquè és específica de la classe o funció amb què s'ha creat. Així, doncs, si establim un valor de distorsió determinat per a cada columna, aquesta propietat aplicarà la distorsió de manera permanent en totes les columnes.

(7)En l'Actionscript els objectes es poden crear mitjançant una funció. En fer-ho d'aquesta manera i associar l'objecte esmentat a la propietat *prototype*, el que fem en realitat és permetre que altres funcions heretin allò que defineix l'objecte de la funció esmentada.

A priori aquest plantejament pot semblar que no provocarà cap distorsió en la imatge sinó tan sols un canvi de mida, però si relacionem la deformació d'una columna amb la mida que té la següent o l'anterior el resultat ja serà diferent. Si a més d'això considerem que cada columna pot tenir una amplada d'un únic píxel, la deformació serà constant i aconseguirem l'efecte desitjat.

Un cop dit tot això, definirem la funció de distorsió, que, com hem dit en el paràgraf anterior, tindrà un valor numèric. Per a això afegirem les línies de codi següents al nostre projecte Flash.

```
1  import flash.display.BitmapData;
2  import flash.geom.Rectangle;
3  import flash.geom.Point;
4
5  MovieClip.prototype.distorsionar = function(distorsion:Number) {
6      var ancho:Number = Math.round(this._width);
7      var alto:Number = Math.round(this._height);
8  }
```

És ara el moment de crear l'objecte BitmapData en el qual s'emmagatzemarà la informació del clip que es troba en l'escenari i que és el que volem distorsionar. Fixeu-vos que és en aquesta part en la qual s'inclou el mètode draw, ja que serà aquest l'encarregat de traçar la nova imatge.

```
1  import flash.display.BitmapData;
2  import flash.geom.Rectangle;
3  import flash.geom.Point;
4
5  MovieClip.prototype.distorsionar = function(distorsion:Number) {
6      var ancho:Number = Math.round(this._width);
7      var alto:Number = Math.round(this._height);
8      var bitmap_1:BitmapData = new BitmapData(ancho, alto, true, 0x00000000);
9      bitmap_1.draw(this);
10     this._visible = false;
11     distorsionado_mc.removeMovieClip();
12 }
```

És convenient observar que el codi afegit continua essent dins de la funció de la distorsió.

Ara crearem el clip que contindrà la distorsió.

```
1  import flash.display.BitmapData;
2  import flash.geom.Rectangle;
3  import flash.geom.Point;
4
5  MovieClip.prototype.distorsionar = function(distorsion:Number) {
6      var ancho:Number = Math.round(this._width);
7      var alto:Number = Math.round(this._height);
8      var bitmap_1:BitmapData = new BitmapData(ancho, alto, true, 0x00000000);
9      bitmap_1.draw(this);
10     this._visible = false;
11     distorsionado_mc.removeMovieClip();
12     var contenedor:MovieClip = this._parent.createEmptyMovieClip
13     ("distorsionado_mc", this._parent.getNextHighestDepth(), {_x:this._x, _y:this._y});
14     contenedor._x = this._x;
15     contenedor._y = this._y;
16 }
```

Fins aquí tot funcionaria bé, però la distorsió no s'aplicaria, ja que ens falta definir la quantitat de columnes que volem que contingui aquesta deformació. Si partim del que ja hem dit anteriorment, que volem que tingui tantes columnes com píxels tingui la imatge, el més senzill serà establir un bucle per a separar el clip en tantes columnes com píxels d'amplada tingui. Aprofitarem a més aquest bucle per definir les quantitats que s'aplicaran en les operacions matemàtiques de deformació. Així, doncs, després de col·locar el codi del bucle, el codi sencer de la funció s'hauria de correspondre amb el de la imatge inferior.

```
MovieClip.prototype.distorsionar = function(distorsion:Number) {
    var ancho:Number = Math.round(this._width);
    var alto:Number = Math.round(this._height);
    var bitmap_1:BitmapData = new BitmapData(ancho, alto, true, 0x00000000);
    bitmap_1.draw(this);
    this._visible = false;
    distorsionado_mc.removeMovieClip();
    var contenedor:MovieClip = this._parent.createEmptyMovieClip
    ("distorsionado_mc", this._parent.getNextHighestDepth(), {_x:this._x, _y:this._y});
    contenedor._x = this._x;
    contenedor._y = this._y;
    for (var k:Number = 1; k<ancho; k++) {
        var clip_temp:MovieClip = contenedor.createEmptyMovieClip
        ("clip"+k, contenedor.getNextHighestDepth(), {_x:k, _y:0});
        clip_temp._x = k;
        var bitmap_temp:BitmapData = new BitmapData(1, alto);
        bitmap_temp.copyPixels(bitmap_1,
                               new Rectangle(k, 0, k, alto),
                               new Point(0, 0));
        clip_temp.attachBitmap(bitmap_temp,
                               clip_temp.getNextHighestDepth(),
                               "auto",true);
        clip_temp._yscale = ((k/(alto-1))*(100-distorsion))+distorsion;
        clip_temp._y = (alto-clip_temp._height)*0.5;
    }
};
```

Amb això ja hem arribat al final i tan sols ens queda per indicar el valor numèric que determini la quantitat de distorsió que aplicarà la funció. Evidentment aquest element no formarà part del codi de la funció, ja que no en pot formar part i manar-hi, així que indicarem el valor numèric esmentat introduint el codi següent al final d'aquest.

```
imagen.distorsionar();
```

Recordeu que la paraula *imatge* és l'identificador del clip que originalment es trobava situat en l'escenari. A més d'això, és important tenir present que la funció de distorsionar necessita un valor numèric entre els parèntesis perquè faci efecte.

En aquest sentit es pot dir que tots els valors numèrics poden generar deformacions. Així, en les condicions establertes dins del bucle del codi de la imatge anterior, un valor 0 mostraria una deformació màxima en el costat esquerre de la imatge, mentre que un valor de 100 no mostrarà cap deformació. Els valors superiors a 100 deformaran la imatge pel costat dret, mentre que els inferiors a 0 generaran imatges amb aspecte doblat.

Tots aquests valors són modificables tant en quantitat de deformació com en posició de la imatge en funció del següent:

- El punt de registre del clip originari. Com passa en qualsevol clip, el punt de registre defineix uns valors concrets de posició. El fet de variar el punt de registre situat originàriament sobre el punt (0,0) farà variar la quantitat i l'aspecte de la deformació.

- La variació dels paràmetres que conté aquesta línia de codi i que afecten l'escala del clip amb referència a l'eix *y*.

```
clip_temp._yscale = ((k/(alto-1))*(100-distorsion))+distorsion;
```

- La variació dels paràmetres que conté aquesta línia de codi i que afecten la posició del clip amb referència a l'eix *y*.

```
clip_temp._y = (alto-clip_temp._height)*0.5;
```

En la imatge inferior podeu veure l'aspecte que ara hauria de tenir el vostre codi.

```
1  import flash.display.BitmapData;
2  import flash.geom.Rectangle;
3  import flash.geom.Point;
4
5  MovieClip.prototype.distorsionar = function(distorsion:Number) {
6      var ancho:Number = Math.round(this._width);
7      var alto:Number = Math.round(this._height);
8      var bitmap_1:BitmapData = new BitmapData(ancho, alto, true, 0x00000000);
9      bitmap_1.draw(this);
10     this._visible = false;
11     distorsionado_mc.removeMovieClip();
12     var contenedor:MovieClip = this._parent.createEmptyMovieClip
13     ("distorsionado_mc", this._parent.getNextHighestDepth(), {_x:this._x, _y:this._y});
14     contenedor._x = this._x;
15     contenedor._y = this._y;
16     for (var k:Number = 1; k<ancho; k++) {
17         var clip_temp:MovieClip = contenedor.createEmptyMovieClip
18         ("clip"+k, contenedor.getNextHighestDepth(), {_x:k, _y:0});
19         clip_temp._x = k;
20         var bitmap_temp:BitmapData = new BitmapData(1, alto);
21         bitmap_temp.copyPixels(bitmap_1,
22                                new Rectangle(k, 0, k, alto),
23                                new Point(0, 0));
24         clip_temp.attachBitmap(bitmap_temp,
25                                clip_temp.getNextHighestDepth(),
26                                "auto",true);
27         clip_temp._yscale = ((k/(alto-1))*(100-distorsion))+distorsion;
28         clip_temp._y = (alto-clip_temp._height)*0.5;
29     }
30 };
31 imagen.distorsionar(0);
```

Finalment, i amb l'únic objectiu d'alliberar memòria de l'ordinador de l'usuari final, és convenient introduir el codi d'eliminació de mapa de bits que s'ha utilitzat per a generar la distorsió i que ara ja no s'usa.

Això, recordem que es pot fer usant el mètode ***dispose***, del qual ja hem parlat en l'apartat anterior. Podem introduir el codi esmentat tant al final de la funció com fora d'aquesta, el resultat serà el mateix.

```
20         clip_temp.attachBitmap(bitmap_temp,clip
21         clip_temp._yscale = ((k/(alto-1))*(100-
22         clip_temp._y = (alto-clip_temp._height)
23     }
24     bitmap_1.dispose();
25 };
26 imagen.distorsionar(50);
```

```
22          clip_temp._y = (alto-clip_temp._height)*(
23      }
24  };
25  bitmap_1.dispose();
26  imagen.distorsionar(50);
```

6. Els mètodes getPixel i setPixel

Quan volem canviar l'aparença d'una imatge de mapa de bits amb els seus píxels, el primer que necessitem obtenir són els valors de color de cada un dels píxels que conté l'àrea que manipularem. Per a llegir aquests valors dels píxels s'utilitza el mètode **getPixel**().

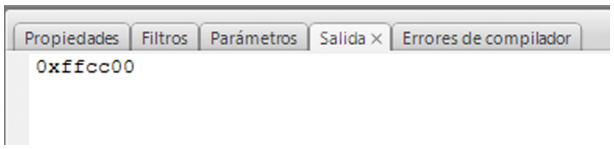
Aquest mètode obté el **valor RGB** del parell de coordenades que conformen un píxel i les converteix en paràmetres susceptibles de ser treballats amb altres mètodes com ara el **setPixel**().

> El mètode **getPixel** tan sols admet dos paràmetres, que són els que corresponen a les coordenades *X* i *Y*.

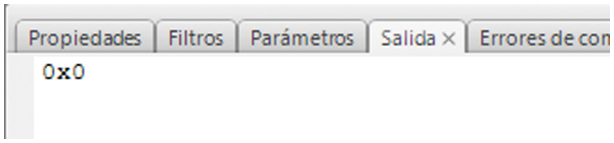
En l'exemple següent s'utilitza el mètode **getPixel**() per a recuperar el valor RGB d'un píxel en la posició (0,0).

```
import flash.display.BitmapData;

var miBitmapData:BitmapData = new BitmapData(200, 200, false, 0x00FFCC00);

var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
mc.attachBitmap(miBitmapData, this.getNextHighestDepth());
trace("0x" + miBitmapData.getPixel(0, 0).toString(16));
```

Si proveu el codi anterior podreu comprovar com el Flash, gràcies a l'acció del ***trace***, ens torna el valor del color del píxel escollit en notació hexadecimal per mitjà de la **finestra de *Salida***.



Si varieu les coordenades i les establiu fora del mapa de bits, podreu comprovar com en la finestra de *Salida* es deixa de mostrar el codi de color.



Per la seva banda, el mètode **setPixel**() estableix el color d'un sol píxel d'un objecte BitmapData situat en l'escenari del Flash Player. Amb l'aplicació del mètode es pot canviar el valor del color del píxel però no el seu grau de transparència, que es manté inalterable.

> Per la seva banda, el mètode setPixel() admet tres paràmetres: *X*, *Y* i color. Els paràmetres *X* i *Y* defineixen el píxel que es canviarà de color. El tercer paràmetre estableix el color de substitució.

Si al codi anterior ara afegim unes línies de manera que el clip tingui **comportament de botó** i es pugui executar la funció **pintar**, podrem comprovar com fent clic sobre el quadrat groc van apareixent punts negres. Cada punt negre correspon a un valor tornat de l'aplicació del mètode **setPixel**().

```
1  import flash.display.BitmapData;
2
3  var miBitmapData:BitmapData = new BitmapData(200, 200, false, 0x00FFCC00);
4
5  var mc:MovieClip = this.createEmptyMovieClip("mc", this.getNextHighestDepth());
6  mc.attachBitmap(miBitmapData, this.getNextHighestDepth());
7  trace("0x" + miBitmapData.getPixel(300, 0).toString(16));
8
9  mc.onPress = function() {
10     this.onEnterFrame = pintar;
11 }
12
13 mc.onRelease = function() {
14     delete this.onEnterFrame;
15 }
16
17 function pintar() {
18     miBitmapData.setPixel(_xmouse, _ymouse, 0x000000);
19 }
```

6.1. Creació d'una aplicació que permeti obtenir la informació de color dels píxels d'una imatge

Partint de la capacitat que té el mètode **getPixel**() d'obtenir la informació de color d'un píxel, crearem una miniaplicació que ens permeti saber amb exactitud el color concret dels píxels d'una imatge.

Per a això haurem de recórrer a alguns elements d'altres classes com, per exemple:

- ***MovieClipLoader***. Aquesta classe permet implementar funcions devolució de crida (*callback*) i proporcionar informació d'estat mentre s'estan carregant arxius d'imatge en clips de pel·lícula.

- ***Object***. Funció que crea un nou objecte buit que es podrà omplir amb la informació o contingut que ens interessi.

- ***Color***. La classe Color ens permetrà establir i/o recuperar els valors d'un color qualsevol en l'escala RGB.

1) Per començar aquest exercici recupereu de la carpeta de recursos l'arxiu **info_color0.fla** i l'arxiu **globos.jpg**. Deseu tots dos arxius en una carpeta del vostre ordinador.

Fixeu-vos que en l'escenari es troben els elements següents:

- Una forma vectorial d'un rectangle que tindrà com a única funció fer de fons d'escena com un marc de la imatge que carreguem mitjançant BitmapData.

- Un clip de pel·lícula de color gris que es troba identificat en la finestra de *Propiedades* amb el nom ***muestra_mc*** i que serà l'encarregat de representar la mostra de color presa amb el mètode getPixel().

- Quatre quadres de text dinàmic degudament identificats en la finestra de *Propiedades*. Aquests quadres de text corresponen als camps dels quals volem extreure informació: valor de color en escala hexadecimal, valor de color en escala RGB, posició respecte a l'eix *X* i posició respecte a l'eix *Y*.

2) Iniciarem el nostre codi creant el carregador d'imatge i el nou objecte buit tal com anunciàvem uns quants paràgrafs més amunt.

```
1  var foto:MovieClipLoader = new MovieClipLoader();
2  var obj_foto:Object = new Object();
```

3) Ara crearem el clip que farà de contenidor de la imatge.

```
1  var foto:MovieClipLoader = new MovieClipLoader();
2  var obj_foto:Object = new Object();
3
4  this.createEmptyMovieClip("nuevo_clip", 0);
5
```

Amb el contenidor ja creat hi podem carregar el clip mitjançant **MovieClipLoader**. Per a això recorrerem als mètodes **addListener**[8] i **loadClip**. El primer serà l'encarregat de registrar l'objecte (**obj_foto**) que hem creat en la segona línia de codi que hem escrit, mentre que el segon mètode serà l'encarregat d'omplir-lo amb el contingut de la imatge que volem analitzar.

[8]El mètode addListener és un detector d'esdeveniments. Aquest mètode permet que un objecte determinat rebi esdeveniments difosos per un altre objecte. L'objecte difusor registra l'objecte detector, que rebrà els esdeveniments generats pel difusor.

```
var foto:MovieClipLoader = new MovieClipLoader();
var obj_foto:Object = new Object();

this.createEmptyMovieClip("nuevo_clip", 0);

foto.addListener(obj_foto);
foto.loadClip("globos.jpg", nuevo_clip);
```

Amb la imatge ja carregada dins del clip **nuevo_clip**, ara necessitem poder establir les propietats i mètodes per a interactuar amb la imatge que hem carregat. Tanmateix, això no es pot fer directament, sinó que prèviament hem de crear una funció que cridi al detector **onLoadInit**[9]. Serà amb aquesta funció que podrem establir les propietats i mètodes que ens convinguin.

(9)El detector onLoadInit està directament lligat al registre efectuat anteriorment. Cridem aquest detector quan ja s'ha carregat el clip. Una vegada que hàgim cridat aquest detector, podrem establir les propietats i mètodes que ens convinguin per a interactuar-hi.

4) Així, doncs, afegirem el codi següent:

```
var foto:MovieClipLoader = new MovieClipLoader();
var obj_foto:Object = new Object();

this.createEmptyMovieClip("nuevo_clip", 0);

foto.addListener(obj_foto);
foto.loadClip("globos.jpg", nuevo_clip);

obj_foto.onLoadInit = function() {
    mostrar_imagen();
};
```

5) A continuació crearem la funció **mostrar_imagen** i afegirem les propietats i mètodes que necessitem.

```
function mostrar_imagen() {
    nuevo_clip.onMouseMove = function() {
        var imagen_cargada:flash.display.BitmapData =
                        new flash.display.BitmapData(this._width, this._height);
        imagen_cargada.draw(this);
        var color_escala_hexa = imagen_cargada.getPixel(this._xmouse,
                                                        this._ymouse);
        color_escala_hexa = "0x"+color_escala_hexa.toString(16);
        valor_hex.text = "Valores de la escala Hexadecimal ("+color_escala_hexa+")";
    };
```

Fixeu-vos que el procediment és molt simple, primer cridem la classe BitmapData, després tracem el mapa de bits mitjançant el mètode draw que ja hem vist anteriorment, i una vegada traçat el mapa de bits cridem el mètode getPixel() per obtenir la informació del color del píxel. Finalment abocarem aquesta informació en un dels quadres de text, el primer, el qual ens tornarà el valor numèric del color del píxel en escala hexadecimal.

El codi final de la miniaplicació que estem fent serà doncs el següent:

```
var foto:MovieClipLoader = new MovieClipLoader();
var obj_foto:Object = new Object();

this.createEmptyMovieClip("nuevo_clip", 0);

obj_foto.onLoadInit = function() {
    mostrar_imagen();
};
foto.addListener(obj_foto);
foto.loadClip("globos.jpg", nuevo_clip);

function mostrar_imagen() {
    nuevo_clip.onMouseMove = function() {
        var imagen_cargada:flash.display.BitmapData =
            new flash.display.BitmapData(this._width, this._height);
        imagen_cargada.draw(this);
        var color_escala_hexa = imagen_cargada.getPixel(this._xmouse,
                                                        this._ymouse);
        color_escala_hexa = "0x"+color_escala_hexa.toString(16);
        valor_hex.text = "Valores de la escala Hexadecimal ("+color_escala_hexa+")";
    };
}
```

Si ara provem la miniaplicació veurem que la imatge es carrega correctament i que a mesura que movem el ratolí per sobre de la imatge es mostra el valor del color en el primer quadre de text.

A més d'això també podrem veure el següent:

- La imatge es mostra descentrada.
- No es mostra cap informació de les coordenades del píxel que es pren de referència.
- No es mostren els valors del color en l'escala RGB.
- No es mostra la còpia del color seleccionat en el clip inferior.

Solucionarem tots aquests problemes de manera gradual.

1) La imatge es mostra descentrada

Això és perquè, com ja hem vist altres vegades, si no s'indica una coordenada específica, les imatges carregades en temps d'execució es desenvolupen a partir de la coordenada (0,0). Vist això, n'hi haurà prou d'afegir les coordenades adequades i el problema estarà resolt. Afegirem les línies de codi següents:

```
var foto:MovieClipLoader = new MovieClipLoader();
var obj_foto:Object = new Object();

this.createEmptyMovieClip("nuevo_clip", 0);
nuevo_clip._x = 10;
nuevo_clip._y = 10;
obj_foto.onLoadInit = function() {
    mostrar_imagen();
};
```

2) No es mostra cap informació de les coordenades del píxel que es pren de referència

Haurem de buscar solució a aquest problema en l'apartat en el qual descrivim les propietats i mètodes de la funció. En aquest cas n'hi haurà prou de relacionar els dos quadres de text situats ja en l'escenari amb les posicions *X* i *Y* del ratolí. Això ens donarà un valor numèric amb què podrem saber a quin punt concret ens referim en cada moment.

```
    valor_hex.text = "Valores de la escala Hexadecimal ("+color_escala_hexa+")";
    posicionX.text = "Valor del eje X = "+this._xmouse;
    posicionY.text = "Valor del eje Y = "+this._ymouse;
};
```

3) No es mostren els valors del color en l'escala RGB

El Flash no treballa amb colors d'aquesta escala sinó que únicament admet valors de l'escala hexadecimal. Tanmateix, es pot fer una funció que permet fer el canvi d'escala directament. El codi d'aquesta funció és el següent:

```
function deEXaRGB(valor:Number):Object {
   var RGB = new Object();
   RGB.red = (valor &gt; &gt;16) &amp; 0xFF;
   RGB.green = (valor &gt; &gt;8) &amp; 0xFF;
   RGB.blue = valor &amp; 0xFF;
   return RGB;
   }
```

Així, doncs, afegirem aquesta funció al nostre codi, la col·locarem just abans de la funció **mostrar_imagen** perquè la conversió d'escala es faci abans que s'executin els mètodes d'aquesta funció.

```
3  foto.loadClip("globos.jpg", nuevo_clip);
4
5  function deEXaRGB(valor:Number):Object {
6      var RGB = new Object();
7      RGB.red = (valor >> 16) & 0xFF;
8      RGB.green = (valor >> 8) & 0xFF;
9      RGB.blue = valor & 0xFF;
0      return RGB;
1  }
2
3  function mostrar_imagen() {
```

Un cop feta la reconversió d'escala perquè els valors RGB es mostrin en l'escenari del Flash Player haurem de relacionar el quadre de text que ja està en l'escenari amb els valors RGB obtinguts després de la reconversió d'escala. Això significa introduir una línia de codi molt semblant a la que ens permet mostrar l'escala hexadecimal en el quadre de text pertinent.

```
imagen_cargada.draw(this);
var color_escala_hexa = imagen_cargada.getPixel(this._xmouse, this._ymouse);
var color_escala_RGB = deEXaRGB(imagen_cargada.getPixel(this._xmouse, this._ymouse));
color_escala_hexa = "0x"+color_escala_hexa.toString(16);
```

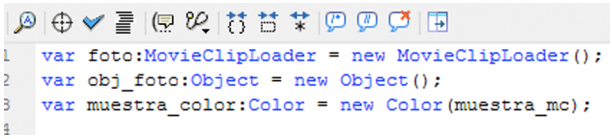
Si ara provem la nostra miniaplicació veurem que ja funciona tot a excepció del clip que ha de mostrar la còpia del color seleccionat.

4) No es mostra la còpia del color seleccionat en el clip inferior

Hem deixat aquest petit problema per al final, ja que la solució no passa únicament per cridar un mètode o una propietat, sinó perquè aquest color es pugui mostrar hem de cridar una classe que està condicionada pel fet que existeixin els valors RGB d'un color: és la **classe Color**.

Com hem explicat al principi d'aquest exercici, la classe Color ens permet recuperar els valors d'un color qualsevol en l'escala RGB i convertir-los en color visible, i per això abans de solucionar aquest problema hauríem de solucionar la reconversió dels valors hexadecimals que ens dóna el Flash a valors RGB, que són els que necessita la classe Color per a poder mostrar-nos el color seleccionat.

Vist això, amb el nostre codi crearem un nou element pertanyent a la classe Color. Per a això afegirem una línia de codi en la zona superior d'aquest, en la part en què cridem les diferents classes.



```
var foto:MovieClipLoader = new MovieClipLoader();
var obj_foto:Object = new Object();
var muestra_color:Color = new Color(muestra_mc);
```

Fixeu-vos que relacionem la variable **muestra_color** amb una nova instància de Color que queda registrada en el clip **muestra_mc**. Aquest clip no és altre que el clip de color gris que es troba en l'escenari, el qual ja té posat aquest nom en la finestra de *Propiedades*. Aquest clip serà l'encarregat de mostrar una còpia del color seleccionat en cada moment.

Si ara provem l'aplicació veurem que encara no funciona. Això és perquè encara ens queda per implementar el mètode que doni cos a la variable muestra_color i, per tant, faci visibles les dades de color recollides. Aquest mètode és **setRGB**(), i com tots els mètodes haurà de quedar reflectit dins de la funció generadora de l'aplicació. Així, doncs, el codi afegit serà:

```
var color_escala_RGB = deEXaRGB(imagen_cargada.getPixel(thi
color_escala_hexa = "0x"+color_escala_hexa.toString(16);
muestra_color.setRGB(color_escala_hexa);
valor_hex.text = "Valores de la escala Hexadecimal ("+colo
valor_rgb.text = "Valores de la escala RGB ("+color_escala_
```

Amb això hem completat l'aplicació que volíem dur a terme. El codi complet seria el següent:

```
var foto:MovieClipLoader = new MovieClipLoader();
var obj_foto:Object = new Object();
var muestra_color:Color = new Color(muestra_mc);

this.createEmptyMovieClip("nuevo_clip", 0);
nuevo_clip._x = 10;
nuevo_clip._y = 10;
obj_foto.onLoadInit = function() {
    init();
};
foto.addListener(obj_foto);
foto.loadClip("globos.jpg", nuevo_clip);
function deEXaRGB(valor:Number):Object {
    var RGB = new Object();
    RGB.red = (valor >> 16) & 0xFF;
    RGB.green = (valor >> 8) & 0xFF;
    RGB.blue = valor & 0xFF;
    return RGB;
}
function init() {
    nuevo_clip.onMouseMove = function() {
        var imagen_cargada:flash.display.BitmapData =
                new flash.display.BitmapData(this._width,
                                             this._height);
        imagen_cargada.draw(this);
        var color_escala_hexa = imagen_cargada.getPixel(this._xmouse,
                                                        this._ymouse);
        var color_escala_RGB = deEXaRGB(imagen_cargada.getPixel
                                        (this._xmouse, this._ymouse));
        color_escala_hexa = "0x"+color_escala_hexa.toString(16);
        muestra_color.setRGB(color_escala_hexa);
        valor_hex.text = "Valores de la escala Hexadecimal ("+color_escala_hexa+")";
        valor_rgb.text = "Valores de la escala RGB ("+color_escala_RGB.red+",
                         "+color_escala_RGB.green+", "+color_escala_RGB.blue+")";
        posicionX.text = "Valor del eje X = "+this._xmouse;
        posicionY.text = "Valor del eje Y = "+this._ymouse;
    };
}
```

Fixeu-vos que en el nostre codi final ha aparegut una nova funció, la funció **init**.

Quan fem una càrrega usant el mètode **load**, el Flash només envia l'ordre de carregar l'element indicat, en el nostre cas la imatge **globos.jpg**, però no espera que es faci la càrrega i continua amb el flux del que li indica la programació.

> El que se sol fer quan el resultat depèn fortament que es completi la càrrega és crear una funció init.

Aquesta funció s'assegurarà que la càrrega es completi. La funció **init** té a més com a finalitat la inicialització de les variables i objectes que tindrà l'aplicació. Aquesta funció s'executa només una vegada immediatament després de la càrrega de l'aplicació.

7. El mètode copyChannel de BitmapData

Aquest mètode transfereix les dades d'un canal de color d'un objecte **BitmapData** a l'altre. Per tant, utilitzant-lo podem afectar directament el color d'una imatge i fer així que una mateixa imatge d'origen es pugui mostrar sota múltiples aparences.

Els paràmetres que admet aquest mètode són **sourceBitmap**, **sourceRect**, **destPoint**, **sourceChannel**, **destChannel**:

- ***sourceBitmap***. Serveix per a indicar la imatge de mapa de bits que s'utilitzarà.

- ***sourceRect***. L'objecte Rectangle d'origen.

- ***destPoint***. Indica la cantonada superior esquerra de l'àrea rectangular en què se situaran les noves dades del canal en l'objecte Point de destinació.

- ***sourceChannel***. Indica el canal d'origen. Els canals són el vermell, el verd, el blau i el de transparència, i estan representats pels números 1,2,4 i 8 respectivament. Poden barrejar-se canals seguint la mateixa notació que s'ha fet servir en l'apartat anterior, –1 | 4, per exemple.

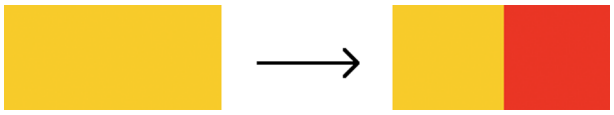
- ***destChannel***. Indica el canal de destinació. La seva mecànica d'ús és idèntica a l'anterior.

7.1. Un exemple senzill

Si copieu el codi següent en la finestra d'*Acciones* d'un arxiu nou del Flash, podreu veure com es copia el canal d'origen indicat des d'un objecte BitmapData de color groc en una part del mateix BitmapData definida pel punt (101,0). A partir d'aquesta coordenada, quan passeu per sobre amb el cursor del ratolí canviarà de color i passarà a mostrar-se de color vermell.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

var miBitmapData:BitmapData = new BitmapData(200,
                                             100,
                                             false,
                                             0x00FFCC00);

var mc:MovieClip = this.createEmptyMovieClip
                    ("mc", this.getNextHighestDepth());
mc.attachBitmap(miBitmapData,
                this.getNextHighestDepth());

mc.onRollOver = function() {
    miBitmapData.copyChannel(miBitmapData,
                             new Rectangle(0, 0, 100, 100),
                             new Point(101, 0),
                             4,
                             2);
}
```

Aquí podeu veure una imatge del resultat abans i després de fer **rollOver** amb el ratolí.



7.2. Jugant amb els canals de color d'una imatge

De la mateixa manera que variem el color d'una imatge de bits plana, amb aquest mètode també podem variar el color d'imatges amb contingut fotogràfic tal com podreu observar en l'exercici que es planteja a continuació.

1) Obriu l'arxiu **flores0.fla**. Veureu que es tracta d'un arxiu de 600 × 400 píxels, en el qual es mostra, a partir de la coordenada (0,0) de l'escenari, una imatge de 300 × 200 píxels.

Basant-nos en el codi de l'exemple anterior mostrarem aquesta imatge sota tres aparences diferents.

2) El primer que farem serà **importar les classes** necessàries.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
```

3) A continuació crearem l'**objecte BipmapData** sense cap modificació.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

bitmap1 = BitmapData.loadBitmap("flores.jpg");
```

4) Ara serà el moment de crear els tres nous objectes **BitmapData** i situar-los dins d'un clip contenidor.

Observació

Fixeu-vos en l'aparició de la paraula clau **this**. Aquesta paraula és reservada en el Flaix de la mateixa manera a que ho estan altres paraules (*if, else, function* o *return*), i quan s'usa en un *script* fa referència a l'objecte que conté aquest *script*. Dins d'un mètode, aquesta paraula fa referència a la instància de classe que conté el mètode cridat.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

bitmap1 = BitmapData.loadBitmap("flores.jpg");

var bitmap2:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
var bitmap3:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
var bitmap4:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
this.createEmptyMovieClip('mc2', this.getNextHighestDepth());
this.createEmptyMovieClip('mc3', this.getNextHighestDepth());
this.createEmptyMovieClip('mc4', this.getNextHighestDepth());
```

5) Per evitar que quedin encavalcats indicarem les coordenades en les quals s'han d'ubicar aquests nous elements creats. Fixeu-vos que per una banda el clip "mc2" només té la indicació de la coordenada *X*, i el clip "mc3" només la té en la coordenada *Y*. Això és perquè les coordenades no indicades continuen estant en 0 i, per tant, en l'origen de creació per defecte.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

bitmap1 = BitmapData.loadBitmap("flores.jpg");

var bitmap2:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
var bitmap3:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
var bitmap4:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
this.createEmptyMovieClip('mc2', this.getNextHighestDepth());
this.createEmptyMovieClip('mc3', this.getNextHighestDepth());
this.createEmptyMovieClip('mc4', this.getNextHighestDepth());
this.mc2._x = 301;
this.mc3._y = 201;
this.mc4._x = 301;
this.mc4._y = 201;
```

6) Ara omplirem els tres clips contenidors amb els **BitmapData** respectius.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;

bitmap1 = BitmapData.loadBitmap("flores.jpg");

var bitmap2:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
var bitmap3:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
var bitmap4:BitmapData = new BitmapData(300, 200, true, 0xFF000000);
this.createEmptyMovieClip('mc2', this.getNextHighestDepth());
this.createEmptyMovieClip('mc3', this.getNextHighestDepth());
this.createEmptyMovieClip('mc4', this.getNextHighestDepth());
this.mc2._x = 301;
this.mc3._y = 201;
this.mc4._x = 301;
this.mc4._y = 201;

this.mc2.attachBitmap(bitmap2, this.getNextHighestDepth());
this.mc3.attachBitmap(bitmap3, this.getNextHighestDepth());
this.mc4.attachBitmap(bitmap4, this.getNextHighestDepth());
```

7) Arribats a aquest punt és quan aplicarem el mètode de copiar el canal d'origen i canviar-lo en la destinació especificada. Per a això n'hi haurà prou a introduir el codi següent:

```
bitmap2.copyChannel(bitmap1, new Rectangle(0, 0,300, 200), new Point(0, 0), 2, 1);
bitmap3.copyChannel(bitmap1, new Rectangle(0, 0,300, 200), new Point(0, 0), 2, 2);
bitmap4.copyChannel(bitmap1, new Rectangle(0, 0, 300, 200), new Point(0, 0), 2, 4);
```

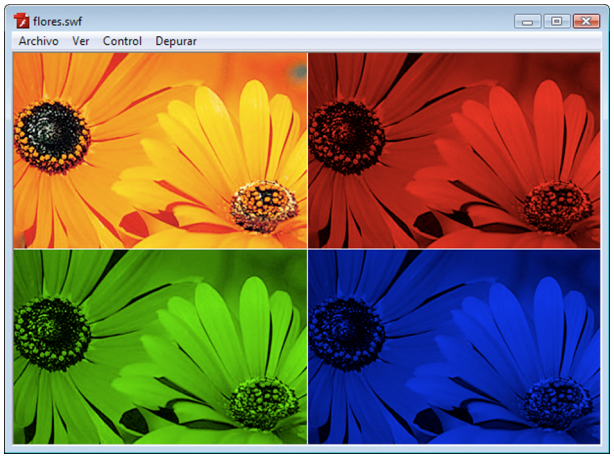
8) Per finalitzar l'exercici i alliberar memòria de l'usuari final, eliminarem el mapa de bits d'origen, el qual ara ja no necessitem per a res.

```
bitmap1.dispose();
```

A continuació podeu veure el codi usat.

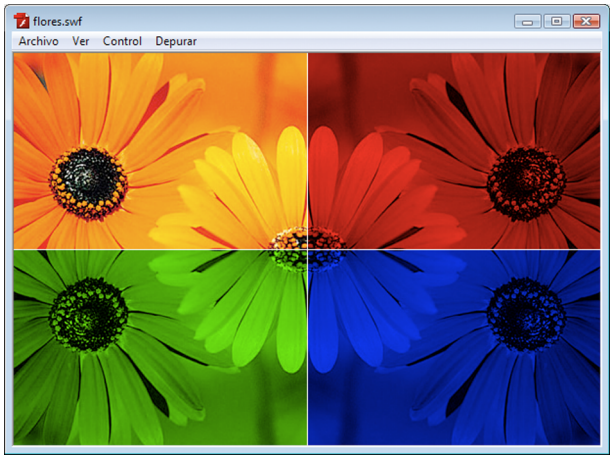
```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
bitmap1 = BitmapData.loadBitmap("flores.jpg");
var bitmap2:BitmapData = new BitmapData(300, 200, true,
                                        0xFF000000);
var bitmap3:BitmapData = new BitmapData(300, 200, true,
                                        0xFF000000);
var bitmap4:BitmapData = new BitmapData(300, 200, true,
                                        0xFF000000);
this.createEmptyMovieClip('mc2',this.getNextHighestDepth());
this.createEmptyMovieClip('mc3',this.getNextHighestDepth());
this.createEmptyMovieClip('mc4',this.getNextHighestDepth());
this.mc2._x = 301;
this.mc3._y = 201;
this.mc4._x = 301;
this.mc4._y = 201;
this.mc2.attachBitmap(bitmap2,this.getNextHighestDepth());
this.mc3.attachBitmap(bitmap3,this.getNextHighestDepth());
this.mc4.attachBitmap(bitmap4,this.getNextHighestDepth());
bitmap2.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                    new Point(0, 0),2,1);
bitmap3.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                    new Point(0, 0),2,2);
bitmap4.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                    new Point(0, 0),2,4);
bitmap1.dispose();
```

I el resultat obtingut



9) Ara deseu aquest arxiu en el vostre ordinador i feu-ne una còpia per poder continuar treballant. L'arxiu desat l'usarem novament al final de l'apartat.

Si ara modifiquem una mica el codi que tenim i variem la posició i l'escala dels mapes de bits creats, podrem obtenir resultats com el de la imatge següent:



Aquest seria el codi final que ha permès la imatge anterior.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
bitmap1 = BitmapData.loadBitmap("flores.jpg");
var bitmap2:BitmapData = new BitmapData(300, 200, true,
                                        0xFF000000);
var bitmap3:BitmapData = new BitmapData(300, 200, true,
                                        0xFF000000);
var bitmap4:BitmapData = new BitmapData(300, 200, true,
                                        0xFF000000);
this.createEmptyMovieClip('mc2',this.getNextHighestDepth());
this.createEmptyMovieClip('mc3',this.getNextHighestDepth());
this.createEmptyMovieClip('mc4',this.getNextHighestDepth());
this.mc2._x = 601;
this.mc3._y = 401;
this.mc4._x = 601;
this.mc4._y = 401;
this.mc2._xscale = -100;
this.mc3._yscale = -100;
this.mc4._xscale = -100;
this.mc4._yscale = -100;
this.mc2.attachBitmap(bitmap2,this.getNextHighestDepth());
this.mc3.attachBitmap(bitmap3,this.getNextHighestDepth());
this.mc4.attachBitmap(bitmap4,this.getNextHighestDepth());
bitmap2.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                    new Point(0, 0),2,1);
bitmap3.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                    new Point(0, 0),2,2);
bitmap4.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                    new Point(0, 0),2,4);
bitmap1.dispose();
```

7.3. Creació d'un òfset d'imatge

Si ens parem a pensar una mica en les possibilitats de mantenir o assignar graus de transparència que ofereix qualsevol objecte **BitmapData** i les possibilitats que hem après al llarg d'aquest apartat quant a poder assignar canals de color, arribarem ràpidament a la conclusió que és possible dur a terme òfsets d'imatges sense que això impliqui gaire esforç.

1) Recupereu l'arxiu que heu desat abans en el vostre ordinador i elimineu la imatge que hi ha en l'escenari. No cal dir que això no l'eliminarà de la **Biblioteca**, en la qual continuarà identificada amb el nom **flores.jpg**.

2) Ara aneu a la finestra d'**Acciones**. El primer que farem serà Ressituar tots els mapes de bits en el centre de l'escenari. Per a això haurem d'eliminar les coordenades de posició indicades anteriorment i substituir-les pel fragment de codi següent:

```
this.createEmptyMovieClip('mc4',this.getNextHighestDepth());

this.mc2._x = 150;
this.mc2._y = 100;
this.mc3._x = 150;
this.mc3._y = 100;
this.mc4._x = 150;
this.mc4._y = 100;

this.mc2.attachBitmap(bitmap2,this.getNextHighestDepth());
```

3) Si ara proveu el vostre projecte Flash veureu que tan sols podeu veure una imatge. Les altres dues no és que no s'hagin carregat sinó que es troben just a sota de la que és en primer terme, en el nivell més elevat.

4) Per solucionar aquest problema de visualització ens desplaçarem a l'inici del codi, al lloc on creàvem els nous objectes de BitmapData. Allà substituirem els valors del parell de transparència de manera que com més elevat sigui el **nivell del mapa de bits** més transparent sigui i, per tant, més s'apropi al valor 00, que correspon a un grau de transparència absoluta.

Nivell del mapa de bits

Els nivells d'ubicació del mapa de bits estan definits pel mètode **getNextHighestDepth**(). Si no s'ha indicat cap nivell en els paràmetres del mètode, l'element generat es col·locarà en el primer nivell superior que es trobi lliure. Amb això podem saber que el clip "mc2" estarà en el nivell 1, "mc3" serà en el nivell 2, i "mc4" serà el que es trobi per damunt de tot.

```
import flash.geom.Point;
bitmap1 = BitmapData.loadBitmap("flores.jpg");
var bitmap2:BitmapData = new BitmapData(300, 200, true,
                                        0xFF000000);
var bitmap3:BitmapData = new BitmapData(300, 200, true,
                                        0x66000000);
var bitmap4:BitmapData = new BitmapData(300, 200, true,
                                        0x33000000);
this.createEmptyMovieClip('mc2',this.getNextHighestDepth());
```

5) Si ara proveu el vostre projecte veureu que el color del mapa de bits ha canviat, ja que ara el que veiem és la suma dels diferents canals de color segons el grau de transparència de cada objecte **BitmapData**.

6) Ara copiarem un canal concret, en aquest cas amb el canal verd, del mapa de bits original en els altres mapes de bits. Per a això, en el mètode **copyChannel** n'hi haurà prou de canviar els dígits del canal d'origen i del de destinació. Recordeu que el canal número 2 és el que correspon al canal verd.

```
bitmap2.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                  new Point(0, 0),2,2);
bitmap3.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                  new Point(0, 0),2,2);
bitmap4.copyChannel(bitmap1,new Rectangle(0, 0, 300, 200),
                                  new Point(0, 0),2,2);
```

El nostre projecte ja està pràcticament a punt: hem definit els diferents graus de transparència i hem copiat en tots els mapes de bits el mateix canal. Ara tan sols ens queda crear la funció que permeti moure els dos clips situats en els nivells superiors i permetre així que, d'una banda, puguem veure les tres imatges al mateix temps creant el desfasament d'imatge que anunciàvem uns paràgrafs abans i, de l'altra, puguem moure aquestes imatges fins a fer-les encaixar i que el resultat visible sigui només el que correspondria a una única imatge.

Per a això crearem un objecte de la classe **Stage**[10].

[10]La classe **Stage** constitueix l'arrel de tota la llista de visualització. Cada arxiu del Flash Player conté una única instància de **Stage** que serà el contenidor principal de tots els elements.

La classe **Stage** és una classe de nivell superior que es va convertir en objecte natiu des de Flash Player 6. Com a conseqüència és possible accedir als seus mètodes, propietats i controladors sense necessitat d'emprar cap constructor previ. Partint d'això podem situar-la directament dins de la funció que crearem i combinar-ne els valors amb la posició a *X* i *Y* del ratolí per a obtenir així les variables de posició dels mapes de bits en cada moment.

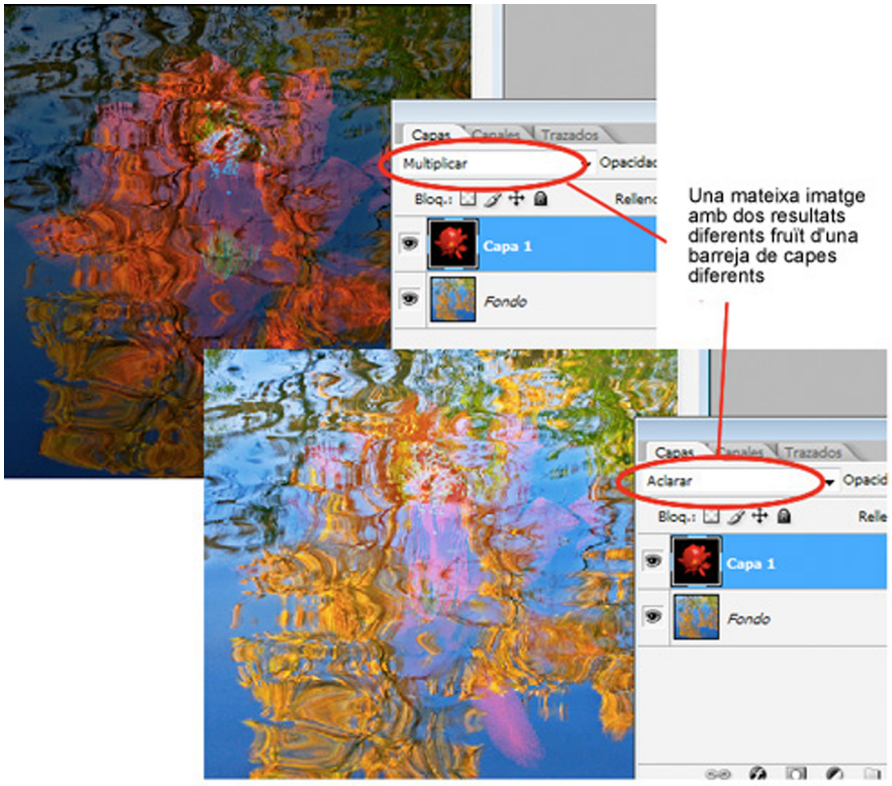
Així, doncs, la funció serà la següent:

```
31  bitmap1.dispose();
32
33  this.onEnterFrame=function(){
34      var pos_y=(this._ymouse-(Stage.height/2))*0.2
35      this.mc4._y = pos_y+(-100+Stage.height/2);
36      this.mc3._y = -pos_y+(-100+Stage.height/2);
37
38      var pos_x=(this._xmouse-(Stage.width/2))*0.2
39      this.mc4._x = pos_x+(-150+Stage.width/2);
40      this.mc3._x = -pos_x+(-150+Stage.width/2);
41  }
```

En la imatge podeu veure un possible resultat.

8. La propietat blendMode de MovieClip

Quan treballem amb programaris de tractament d'imatges com, per exemple, Photoshop és habitual usar tipus de barreja entre les capes d'una imatge per a aconseguir efectes que d'una altra manera seria complicat obtenir.



Encara que des de la interfície del Flash no és possible fer aquest tipus de barreges, amb l'ActionScript sí que és possible. Per a això serà necessari recórrer a la propietat **blendMode** de la classe MovieClip.

> Aplicant la propietat **blendMode** a un clip de pel·lícula aconseguirem que hi hagi un tipus de barreja entre el clip a què s'aplica i qualsevol altre que es trobi situat just a sota de l'anterior.

La barreja entre tots dos clips no s'aplica al clip en temps de treball sinó en temps d'execució, és a dir, des del Flash Player.

Per dur a terme aquesta barreja el Flash Player compara el valor de cada color primari d'un píxel del clip de pel·lícula que té aplicada la propietat de barreja amb el color corresponent del píxel situat en el nivell inferior, i mostra en l'escenari el resultat d'aquesta barreja.

Hi ha catorze tipus de barreja possibles:

- ***Normal***. El clip de pel·lícula apareix davant del fons. Els valors de píxel del clip de pel·lícula superior anul·len els del clip del fons. És el mode de visualització per defecte.

- ***Layer***. Anul·la en el resultat final la transparència que pugui contenir la imatge superior.

- ***Multiply***. Multiplica els valors dels colors primaris del clip de pel·lícula pels del color del clip de fons. Els resultats que s'obtenen són colors més foscos. Se sol utilitzar per a crear efectes d'ombres i profunditat.

- ***Screen***. Multiplica el complementari del color del clip superior pel complementari de l'inferior. Amb aquesta operació s'obtenen efectes de descoloració. Se sol utilitzar per a eliminar àrees massa fosques.

- ***Lighten***. Selecciona el color primari més clar del clip superior i el barreja amb els píxels més lluminosos del clip inferior.

- ***Darken***. Funciona al revés que l'anterior: selecciona el color primari més fosc del clip superior i el barreja amb els píxels menys lluminosos del clip inferior.

- ***Difference***. Compara els colors primaris del clip superior amb els de l'inferior. Mostra el resultat de restar al valor més clar dels píxels sobreposats dels dos clips el valor més fosc d'aquests píxels.

- ***Add***. Suma els valors dels colors del clip superior als del clip inferior.

- ***Subtract***. És l'invers de l'anterior: resta els valors dels colors del clip superior als del clip inferior.

- ***Invert***. Inverteix els colors del clip inferior.

- ***Alpha***. Aplica el valor de transparència de cada píxel del clip superior sobre l'inferior.

- ***Erase***. Esborra el clip inferior en funció del valor alfa del clip superior.

- ***Overlay***. Ajusta el color de cada clip en funció de la foscor del clip inferior. Si aquest és més clar que un gris mitjà (50%), els colors del clip de pel·lícula i del fons es tamisen, aconseguint-se un color més clar. Si el fons és més fosc que un gris mitjà, els colors del clip de pel·lícula i del fons es multipliquen, aconseguint-se un color més fosc. Se sol emprar per a aconseguir augmentar els efectes d'ombreig.

- ***Hardlight***. Funciona igual que l'anterior però prenent com a referència el clip superior.

La propietat blendMode es pot **cridar** de dues maneres, bé mitjançant el valor del camp numèric de tipus de barreja o bé mitjançant el nom de la barreja escrit en forma de cadena de text.

Així, per exemple, si volem fer una barreja del tipus *Difference* podrem escriure el següent:

```
clip_mezclado.blendMode =7
clip_mezclado.blendMode = "difference"
```

Si en el valor numèric establim valors superiors a 14 o inferiors a 1, el mode de barreja s'estableix en el mode per defecte, que és el mode *normal*.

8.1. Experimentant amb els diferents modes de barreja

El codi següent crea dos clips de pel·lícula amb farciments de degradat en l'escenari del Flash Player que tenen una part superposada perquè pugueu experimentar amb els diferents modes de barreja.

```
this.createEmptyMovieClip("mc1",this.getNextHighestDepth());
this.createEmptyMovieClip("mc2",this.getNextHighestDepth());

                                Cambiando los valores de los colores
                                del degradado variarán los resultados.

modoRelleno(mc1,0xFF0000,0xFFCC00,100,100);
modoRelleno(mc2,0x999999,0xCCCCCC,100,50);
mc2._x = mc1._x+50;
mc2._y = mc1._y+50;



                                Cambiando únicamente este valor
mc2.blendMode = 8;              variará el modo de mezcla

function modoRelleno(mc:MovieClip, color1:Number, color2:Number,
                     alpha1:Number, alpha2:Number) {
    matrix = {a:100, b:0, c:0, d:0, e:100, f:0, g:50, h:20, i:1};
    mc.beginGradientFill("linear",[color1, color2],
                         [alpha1, alpha2],[0, 0xFF],matrix);
    mc.lineStyle(2,0xCCCCCC,100);
    mc.moveTo(0,0);
    mc.lineTo(0,100);
    mc.lineTo(100,100);
    mc.lineTo(100,0);
    mc.lineTo(0,0);
    mc.endFill();
}
```

Només canviant els valors indicats en la imatge anterior obtindreu resultats molt diferents. És interessant que experimenteu amb diferents resultats per a familiaritzar-vos amb les diferents possibilitats que ofereix aquesta propietat.

8.2. Experimentant amb les textures en mode de barreja

En l'apartat anterior hem après a aplicar un mode de barreja entre dos clips amb contingut pla, però aquest mode també es pot aplicar a dos clips el contingut dels quals siguin objectes de BitmapData.

1) Obriu l'arxiu **mezcla.fla** que hi ha en la carpeta de recursos. Fixeu-vos que en la Biblioteca del projecte hi ha allotjades dues imatges que ja tenen identificador de vinculació per a ser exportades mitjançant AS.

2) Per a dur a terme aquesta barreja el primer que hem de fer és crear els nostres objectes **BitmapData**, adjudicar-los una imatge com a contingut, crear els clips contenidors i omplir aquests clips amb els mapes de bits per a finalitzar.

```
import flash.display.BitmapData;

bitmap1 = BitmapData.loadBitmap("imagen1");
bitmap2 = BitmapData.loadBitmap("imagen2");

this.createEmptyMovieClip('mc1',this.getNextHighestDepth());
this.createEmptyMovieClip('mc2',this.getNextHighestDepth());

this.mc1.attachBitmap(bitmap1,this.getNextHighestDepth());
this.mc2.attachBitmap(bitmap2,this.getNextHighestDepth());
```

A partir d'aquest codi tan sols ens faltaria establir el mode de barreja entre un clip i l'altre.

```
1  this.mc2.attachBitmap(bitmap2,this.getNextHighestDepth());
2
3  mc2.blendMode = 8;
```

3) Si ara provem el projecte Flash veurem que efectivament es produeix la barreja indicada. Podríem deixar el resultat així, tanmateix podem millorar o canviar fàcilment l'aspecte de la barreja obtinguda incloent altres efectes com, per exemple, l'efecte de soroll que produeix el mètode **perlinNoise**, que ja hem vist en apartats anteriors.

Vegeu també

Sobre el mètode perlinNoise podeu veure el subapartat 4, "Comprovant alguns avantatges de BitmapData", de l'apartat 4 d'aquest mòdul didàctic.

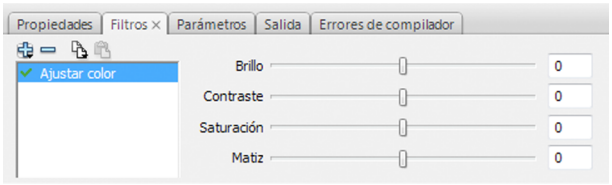
Com que aquest mètode pertany a la classe **BitmapData** no l'aplicarem al clip resultant, sinó que l'haurem d'aplicar al mapa de bits una vegada aquest ja està creat però abans que s'afegeixi al clip. Així, doncs, ara el nostre codi podria quedar així:

```
import flash.display.BitmapData;

bitmap1 = BitmapData.loadBitmap("imagen1");
bitmap2 = BitmapData.loadBitmap("imagen2");

bitmap2.perlinNoise(10,10,3,1,true,true, 2 | 4,false,null);

this.createEmptyMovieClip('mc1',this.getNextHighestDepth());
this.createEmptyMovieClip('mc2',this.getNextHighestDepth());

this.mc1.attachBitmap(bitmap1,this.getNextHighestDepth());
this.mc2.attachBitmap(bitmap2,this.getNextHighestDepth());

mc2.blendMode = 8;
```

Variant els paràmetres de **perlinNoise** i el nombre corresponent al mode de barreja podreu aconseguir fàcilment que la conjugació de les dues imatges tingui aspectes molt diferents que van des d'imatges semblants a il·lustracions de tintes planes a imatges obtingudes per mitjà de vidres gravats. Si no recordeu quins eren els paràmetres de **perlinNoise** reviseu el contingut del quart apartat.

9. La classe DisplacementMapFilter de BitmapFilter

Probablement tots sabeu que des de la versió 8 del Flash es va introduir la possibilitat d'incorporar un conjunt de filtres a les instàncies de botons i clips de pel·lícula. Aquesta operació es pot fer per mitjà de la **finestra de *Filtros*** que apareix quan seleccionem una instància que es troba en l'escenari i és dels símbols esmentats.



Encara que la llista de filtres disponible des de la interfície del Flash és relativament limitada, des de l'ActionScript les possibilitats d'incorporar efectes a una imatge, a una instància de botó o a un clip de pel·lícula creixen considerablement, ja que disposa d'una classe específica amb un conjunt de subclasses que contenen paràmetres suficients per a simular una gran quantitat d'efectes.

Aquesta classe és la classe **BitmapFilter** i les seves subclasses són les següents:

- BevelFilter
- BlurFilter
- ColorMatrixFilte
- ConvolutionFilter
- DisplacementMapFilter
- DropShadowFilter
- GlowFilter
- GradientBevelFilter
- GradientGlowFilter

Encara que alguns autors es refereixen a aquestes subclasses com a classes pròpiament dites, hem de tenir present que per a crear qualsevol d'aquestes subclasses és molt convenient que el nostre codi faci sempre una crida prèvia a la classe superior **BitmapFilter**. Recordem en aquest sentit el que s'esmentava en el primer apartat quan es parlava de l'herència que hi ha entre una classe superior i una subclasse de la primera.

Vegeu també

Sobre l'herència entre classes podeu veure el subapartat 1.3, "Herència," de l'apartat 1 d'aquest mòdul didàctic.

Al llarg d'aquest apartat veurem algunes possibilitats de la subclasse **DisplacementMapFilter**.

Aquesta subclasse utilitza els valors individuals dels diferents píxels d'un objecte BitmapData per a fer un desplaçament d'aquest objecte, o d'un altre, en l'escenari.

Els paràmetres que admet DisplacementMapFilter són:

- **mapBitmap**. Indica el mapa de bits d'origen sobre el qual s'aplicarà el desplaçament.
- **mapPoint**. Indica el punt des del qual s'ha de traçar el nou mapa de bits.
- **componentX** i **componentY**. Són valors numèrics que s'obtenen a partir del valor del canal de color del contingut del mapa de bits d'origen amb referència a les coordenades (x - mapPoint[11] .x) i (y - mapPoint.y).
- **scaleX** i **scaleY**. Són els valors de desplaçament que tindran els píxels del mapa de bits.
- **[mode]**, **[color]** i **[alpha]**. Permeten canviar els modes de barreja, els canals de color i el grau de transparència del resultat.

[11] mapPoint és un valor numèric que indica el desplaçament que sofrirà el clip de pel·lícula de destinació.

Tots els paràmetres de construcció són indispensables excepte els que pertanyen a les matrius del mode, el canal de color i el grau de transparència.

9.1. Creació d'una imatge reflectida sobre aigua

En aquest exercici treballarem amb la intenció d'arribar a un resultat similar al que ja hem obtingut en el primer apartat. La diferència respecte al primer apartat és que en aquell el reflex que hem obtingut és estàtic mentre que ara obtindrem un reflex similar al que es produeix sobre l'aigua.

Vegeu també

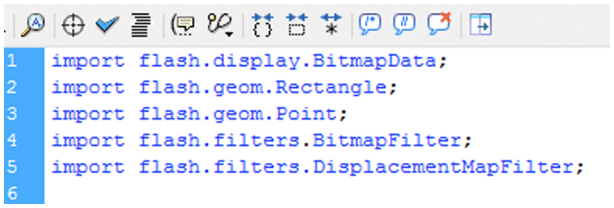
Vegeu el subapartat 3, "Creació dinàmica del reflex d'una imatge," de l'apartat 1 d'aquest mòdul didàctic.

1) Obriu l'arxiu **reflejo2.fla** que hi ha en la carpeta de recursos. Fixeu-vos que es tracta d'un escenari buit de 300 × 400 píxels i que en la Biblioteca del projecte es troba allotjada la mateixa fotografia que ja hem fet servir en el capítol primer. Aquesta vegada la fotografia té posat el nom *imagen* com a identificador per a ActionScript.

2) El primer que farem serà incorporar les classes necessàries. Al llarg d'apartats anteriors ja hem importat diverses vegades les classes **display.BitmapData**, **geom.Rectangle** i **geomPoint**. Aquesta vegada, a més, haurem d'importar les classes que pertanyen als filtres.

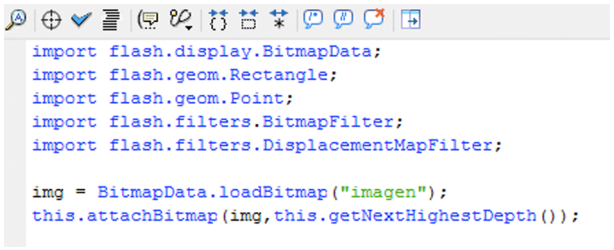
Tal com ja hem anunciat anteriorment, primer importarem la classe de rang superior, que és **filters.BitmapFilter**, i després la subclasse **filters.DisplacementMapFilter**, que és la subclasse del filtre que volem aplicar. Recordeu que si treballeu amb Flash8 o versions posteriors això no és necessari.

Un cop fetes aquestes importacions, el nostre codi hauria de tenir un aspecte similar al següent:



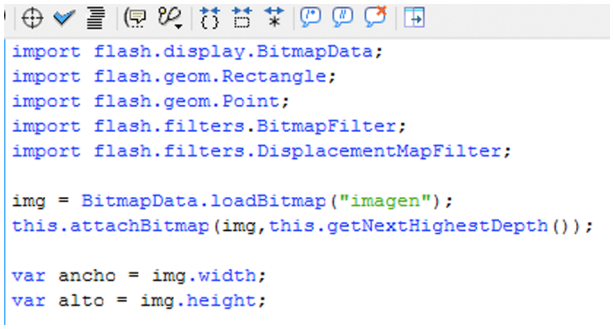
```
1 import flash.display.BitmapData;
2 import flash.geom.Rectangle;
3 import flash.geom.Point;
4 import flash.filters.BitmapFilter;
5 import flash.filters.DisplacementMapFilter;
6
```

3) A continuació procedirem a crear l'**objecte BitmapData** i a ubicar el mapa de bits esmentat en un clip de pel·lícula que faci de contenidor.



```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());
```

4) Si proveu l'arxiu Flash veureu que el mapa de bits ja es troba carregat i situat en la zona superior de l'escenari del Flash Player. Ara establirem els valors de dues variables en amplada i altura, de manera que quan al pas següent creem el segon mapa de bits, que serà el que actuarà com a reflex, aquest mapa de bits prengui com a mesures les mateixes que el primer.



```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;
```

5) Amb les variables ja decretades crearem un segon mapa de bits i l'afegirem a un contenidor.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());
```

Si ara proveu el projecte veureu que aparentment es mostra igual que abans de crear les variables. Encara que l'aparença sigui aquesta, el que passa en realitat és el mateix que passava en l'apartat 7, tots dos mapes de bits es troben encavalcats en l'escenari del Flash Player. Vist això, el que ara farà falta indicar serà la posició que ha d'ocupar el segon mapa de bits.

Si ens parem a pensar que els reflexos d'una imatge es mostren invertits respecte a la imatge original, sabrem que a més de modificar la posició també hem de modificar l'orientació de la imatge reflectida respecte a l'eix *Y*.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;
```

Si ara proveu el vostre arxiu podreu veure que totes dues imatges ja es troben disposades correctament en l'escenari del Flash Player. És, doncs, el moment de començar a treballar en el codi del filtre que volem aplicar. Tanmateix, abans hem de fer una petita observació. El primer mapa de bits que hem creat restarà en l'escenari quan finalitzem el nostre projecte, el segon no. Això és perquè aquest segon mapa de bits, el que hem afegit en segon lloc i que hem canviat de posició i orientació, no l'hem creat perquè allotgi l'efecte de desplaçament sinó que l'hem creat únicament perquè nodreixi de contingut, posició i orientació del mapa de bits que realment contindrà l'efecte de desplaçament.

6) Així, doncs, començarem per crear aquest nou mapa de bits, al qual indicarem que prengui com a referència el que conté la imatge reflectida.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto);
```

Si ara repassem els paràmetres imprescindibles que necessita **DisplacementMapFilter** per a funcionar, veurem que necessitem definir un punt, objecte pertanyent a la classe **Point**, a partir del qual es desenvoluparà el filtre de desplaçament. Vist això, el primer que haurem de fer és crear aquest objecte.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto);

var punto = new Point();
```

Observació

Fixeu-vos que l'objecte punt s'ha creat com una variable. Això és perquè volem que aquest objecte prengui constantment els valors definits per un petit càlcul matemàtic que descriurem posteriorment.

Amb l'objecte punt ja creat podrem passar a crear l'objecte de desplaçament que mostrarà la imatge. Per a això haurem de completar els paràmetres imprescindibles d'aquest objecte. Així, doncs, si cridem aquest objecte com a desplazar haurem d'introduir el següent:

- L'objecte d'origen, que no serà altre que el nou mapa de bits que creem un parell de línies de codi abans i que en el seu moment hem identificat amb el nom de *reflejo*.

- El punt a partir del qual es desenvoluparà el mapa de bits de desplaçament. En aquest cas es tracta de l'objecte punto.

- Els valors numèrics en *X* i *Y* obtinguts a partir del valor de color del mapa de bits reflejo amb referència a les coordenades de l'objecte punto.

- Els valors de desplaçament que tindran els píxels del mapa de bits. Un valor igual a 0 indicarà que no hi haurà desplaçament.

Així, doncs, un possible codi per al nostre desplaçament seria el següent:

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto);

var punto = new Point();

desplazar = new DisplacementMapFilter(reflejo, punto, 0, 1, 0, 15);
```

Si ara proveu el projecte veureu que no funciona. Això és perquè ens falta per crear una funció que **actualitzi** constantment, cada vegada que entri en el fotograma, la imatge del fotograma reproduït anteriorment.

Aquesta funció haurà de contenir els apartats següents:

a) Indicar el grau de variació en què s'ha de situar l'objecte punt. El càlcul matemàtic al qual ja hem fet referència.

b) Indicar la quantitat de soroll que tindrà la imatge resultant. Les imatges reflectides en l'aigua no ho fan com si es tractés d'un mirall sinó que han d'adaptar-se a les ondulacions.

c) Haurem de contenir una crida a l'aplicació del filtre perquè cada vegada que es produeixi l'actualització s'apliqui de nou.

Per a definir el **primer apartat** que tindrà la funció que necessitem utilitzarem un operador que ens torni un valor amb qual puguem omplir el punt que ha d'ocupar la imatge reflectida cada vegada que s'actualitzi el resultat. Una bona opció serà recórrer a la mateixa coordenada que ocupa l'objecte Punt en cada moment, així, doncs, un possible codi seria el següent:

```
punto.y = punto.y - 0.15;
```

o el que és el mateix escrit en una notació més simplificada:

```
punto.y -= 0.15;
```

En adjudicar un valor de resta al punt que ocupa la imatge, el reflex final es produirà partint de la base de la imatge original cap a la base del reproductor del Flash Player. Si canviem aquesta operació per la de suma, el moviment de l'onatge es produirà des de la base del Flash Player cap a la imatge d'origen del reflex. Això és així perquè el clip **imagen_ref** té les coordenades verticals invertides com a conseqüència d'haver-lo escalat fins a –100 en aquesta coordenada.

```
punto.y += 0.15;
```

Per definir la **quantitat de soroll** que tingui la imatge reflectida recorrerem a **perlinNoise**. Encara que ja hem usat aquest mètode altres vegades, si no en recordeu els paràmetres podeu revisar-los en apartats anteriors.

Com a novetat respecte a les altres vegades en què usàvem aquest mètode hi haurà el fet d'incorporar un **òfset** en funció de la posició que ocupi l'objecte punt. Així, doncs, un possible codi seria el següent:

```
reflejo.perlinNoise(0,2,1,0,true,true,4,true,[punto]);
```

Observació

És interessant observar que el paràmetre de l'òfset és entre claudàtors. Això ens indica que la crida es farà a un índex de la matriu punt. Això vol dir que en realitat es tracta més d'un conjunt de valors que no d'un valor únic.

Una matriu pot contenir elements de diversos tipus que s'identifiquen mitjançant un nombre denominat índex. En el nostre cas només contindrà el valor de posició de l'objecte punt, que recordem que serà diferent cada vegada que el reproductor entri en un nou fotograma.

Ja per finalitzar hem de fer la **crida** perquè s'efectuï a l'aplicació de l'objecte de desplaçament que conté la imatge reflectida (**imagen_ref**) i s'aboqui aquesta informació en el mapa de desplaçament (**desplaçar**).

```
imagen_ref.filters = [desplazar];
```

Un cop acabada aquesta la funció, per acabar tot el codi del reflex alliberarem memòria eliminant el mapa de bits que no necessitem, el de la imatge reflectida perquè ara aquest mapa de bits queda sota del mapa de bits que veiem i que conté l'efecte aplicat.

El nostre codi quedarà, doncs, de la manera següent:

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto);

var punto = new Point();

desplazar = new DisplacementMapFilter(reflejo, punto, 0, 1, 0, 15);

onEnterFrame = function () {
    punto.y -= 0.15;
    reflejo.perlinNoise(0,2,1,0,true,true,4,true,[punto]);
    imagen_ref.filters = [desplazar];
};

imagen_ref.dispose();
```

Si ara proveu l'aplicació veureu que funciona correctament però que es mostren unes petites línies força molestes en la zona en què s'ajunten la imatge original i el reflex d'aquesta. Eliminarem aquestes línies usant la propietat ***Clamp***.

Aquesta propietat indica que la imatge s'ha de fixar en les seves vores. *Clamp* admet únicament valors booleans i per defecte el seu valor és *true*. Això significa que si indiquem que s'apliqui aquesta propietat sense més ni més, els píxels dels marges quedaran fixats i eliminarem les molestes línies que provoquen discontinuïtat visual.

El lloc indicat en el qual podrem aplicar aquesta propietat és en el paràmetre mode –anteriorment no l'usàvem– del mapa de desplaçament. Així, doncs, ara aquesta línia de codi quedarà de la manera següent:

```
desplazar = new DisplacementMapFilter(reflejo, punto, 0, 1, 0, 15, "clamp");
```

Ara el reflex s'hauria de mostrar correctament. Si proveu de variar els diferents paràmetres podreu comprovar que hem anat escrivint com varia el resultat.

10. Observant la paral·laxi de les imatges

En l'apartat anterior vam aprendre a fer un reflex i a modificar-ne les característiques. Al llarg d'aquest millorarem el reflex amb la finalitat de dotar-lo de més realisme.

Quan treballem en dues dimensions acostumem a veure sempre l'escenari de l'acció com un element que respon a les dues dimensions, altura i amplada, de la pantalla que limita l'escena i el que s'hi mostra.

Tanmateix, hi ha una altra dimensió que és important tenir en compte, ja que contràriament al que passa amb l'amplada i amb l'altura no té limitació: és la **profunditat**.

Si construïm l'acció al llarg d'aquest eix aconseguirem dotar els nostres resultats de molt més interès visual, però perquè això sigui possible hem de tenir en compte alguns aspectes importants sobre com es veuran i es relacionaran entre ells els diferents elements que intervenen en l'escena.

Aquests aspectes visuals de percepció de la profunditat són:

- superposició
- volum dels elements
- ombres
- canvis de paral·laxi

Els tres primers aspectes són els més senzills de representar, però, **què és i què implica el canvi de paral·laxi en una imatge**?

Parlem de paral·laxi per referir-nos a la desviació angular de què és objecte la posició d'un objecte en funció del punt de vista des del qual és vist.

Així, per exemple, quan ens desplacem amb un vehicle a tota velocitat observem que els elements que tenim en primer terme passen davant de nosaltres a molta més velocitat que els que es troben en plans allunyats. Encara més, els elements situats al fons d'un paisatge amb prou feines si es mouen. Si extrapolem això a la visió que tenim d'un reflex en l'aigua descobrirem que en la zona més pròxima a l'origen el reflex restarà força estàtic, mentre que en la zona més propera a nosaltres els moviments seran més grans.

Mantenir aquesta relació visual entre elements situats en plans diferents és important per a obtenir resultats adequats.

10.1. Creació d'un reflex realista

Dit això, el que farem ara és dotar de profunditat visual el resultat que hem obtingut en l'apartat anterior.

Reprenguem per començar el codi al qual hem arribat en l'apartat anterior.

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());

var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto);

var punto = new Point();

desplazar = new DisplacementMapFilter(reflejo, punto, 0, 1, 0, 15);

onEnterFrame = function () {
    punto.y -= 0.15;
    reflejo.perlinNoise(0,2,1,0,true,true,4,true,[punto]);
    imagen_ref.filters = [desplazar];
};

imagen_ref.dispose();
```

Aprofitarem pràcticament tot aquest codi però introduint algunes modificacions, ja que ara potenciarem la sensació de profunditat que volem donar a la imatge. Concretament, el fragment de codi que aprofitarem serà el següent:

```
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());


var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;
```

Abans, a continuació d'aquest codi creàvem un mapa de bits que era el que contenia el reflex. Aquesta vegada el que farem serà crear tres zones amb una intensitat de soroll diferenciat en l'àrea del reflex. Així, doncs, el primer que haurem de fer és crear els altres dos mapes de bits necessaris per a així poder disposar d'un per a cada zona.

```
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());


var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto/6);
reflejo2 = new BitmapData(ancho, alto/3);
reflejo3 = new BitmapData(ancho, alto/2);
```

Fixeu-vos que en crear aquests nous mapes de bits s'han definit unes mesures concretes d'altura diferents de les mesures d'altura de l'original. Amb això aconseguirem que aquests mapes de bits cobreixin tan sols una part concreta del resultat final i més endavant els puguem situar l'un a continuació de l'altre, i puguem així aplicar diferents graus d'intensitat de l'efecte a cada un.

A continuació crearem els tres objectes **Point** necessaris i els mapes de desplaçament. Fixeu-vos que en la imatge inferior s'han variat els desplaçaments en l'eix *Y* per a potenciar així l'efecte de desplaçament.

```
import flash.filters.DisplacementMapFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());


var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto/6);
reflejo2 = new BitmapData(ancho, alto/3);
reflejo3 = new BitmapData(ancho, alto/2);

var punto = new Point();
var punto2 = new Point();
var punto3 = new Point();

desplazar = new DisplacementMapFilter(reflejo, punto, 0, 1, 0, 10, "clamp");
desplazar2 = new DisplacementMapFilter(reflejo2, punto2, 0, 1, 0, 15, "clamp");
desplazar3 = new DisplacementMapFilter(reflejo3, punto3, 0, 1, 0, 20, "clamp");
```

Si deixéssim el codi de creació dels objectes Point així, tots es crearien a partir de la coordenada (0,0). Per evitar això i al mateix temps recol·locar correctament els mapes de bits que acabem de crear, introduirem valors en aquests punts. Per a això podem introduir el valor concret de la coordenada o podem utilitzar una forma més genèrica basada en el resultat de la variable *alto* que ja hem decretat en l'apartat anterior i que ara, en aquest capítol, hem deixat com a part del codi reutilitzable. Així, doncs, introduirem els següent en la creació dels objectes Point.

```
var punto = new Point(0,0);
var punto2 = new Point(0, alto/6);
var punto3 = new Point(0, alto/2);
```

Ja per acabar aquesta primera variació tan sols ens queda retocar la funció d'actualització introduint les operacions necessàries per al canvi de valors dels objectes **punto2** i **punto3**, assignar un grau de soroll diferent per a cada zona i aplicar la propietat *Filters* als tres mapes de desplaçament creats.

Així, doncs, la funció quedarà de la manera següent:

```
onEnterFrame = function () {
    punto.y -= 0.2;
    punto2.y -= 0.5;
    punto3.y -= 0.9;
    reflejo.perlinNoise(0,2,1,0,true,true,4,true,[punto]);
    reflejo2.perlinNoise(80,4,5,0,true,true,4,true,[punto2]);
    reflejo3.perlinNoise(150,8,10,0,true,true,4,true,[punto3]);
    imagen_ref.filters = [desplazar, desplazar2, desplazar3];
};
```

Observeu els canvis de valor en els primers valors de **PerlinNoise**. Recordeu que amb això s'aconsegueix incrementar l'ondulació.

10.2. Creació d'aigua en moviment

Aplicant alguns canvis ara podrem aprofitar el codi que hem obtingut per a simular el reflex de la imatge sobre l'aigua d'un rierol.

Per a això n'hi haurà prou a introduir en la funció un valor de desplaçament que afecti la posició sobre l'eix *X* dels objectes punt.

```
onEnterFrame = function () {
    punto.x -= 0.3;
    punto.y -= 0.1;
    punto2.x -= 0.8;
    punto2.y -= 0.5;
    punto3.x -= 1.5;
    punto3.y -= 0.9;
    reflejo.perlinNoise(3,2,1,0,true,true,4,true,[punto]);
    reflejo2.perlinNoise(5,2,2,0,true,true,4,true,[punto2]);
    reflejo3.perlinNoise(9,2,4,0,true,true,4,true,[punto3]);
    imagen_ref.filters = [desplazar, desplazar2, desplazar3];
};
```

Amb aquesta modificació es produirà un desplaçament horitzontal en l'efecte creat, el qual es podrà incrementar o reduir en funció de la variació de les quantitats definides en els paràmetres dels mapes de desplaçament i en els del soroll perlín. Proveu amb diferents valors i podreu comprovar que les possibilitats gràfiques dels resultats poden ser molt diverses.

```
desplazar = new DisplacementMapFilter(reflejo, punto, 15, 1, 20, 25, "clamp");
desplazar2 = new DisplacementMapFilter(reflejo2, punto2, 30, 1, 30, 35, "clamp");
desplazar3 = new DisplacementMapFilter(reflejo3, punto3, 50, 1, 40, 50, "clamp");

onEnterFrame = function () {
    punto.x -= 0.3;
    punto.y -= 0.1;
    punto2.x -= 0.8;
    punto2.y -= 0.5;
    punto3.x -= 1.5;
    punto3.y -= 0.9;
    reflejo.perlinNoise(3,2,1,0,true,true,4,true,[punto]);
    reflejo2.perlinNoise(5,2,2,0,true,true,4,true,[punto2]);
    reflejo3.perlinNoise(9,2,4,0,true,true,4,true,[punto3]);
    imagen_ref.filters = [desplazar, desplazar2, desplazar3];
};
```

Després de provar els resultats haureu pogut comprovar fàcilment que el reflex creat conté massa detalls, especialment si es compara amb la realitat. Quan l'aigua es mou, els reflexos acostumen a quedar una mica desenfocats.

En el pas següent farem que el resultat quedi una mica desenfocat amb l'objectiu de dotar de més realisme el resultat. Per a això partirem del codi obtingut fins ara.

Per desenfocar el resultat haurem d'importar un nou element del paquet de **flash.filters**. En aquest cas serà **BlurFilter**. Així, doncs, afegirem aquest apartat a la part del codi en la qual hem importat els diferents paquets.

```
1  import flash.display.BitmapData;
2  import flash.geom.Rectangle;
3  import flash.geom.Point;
4  import flash.filters.BitmapFilter;
5  import flash.filters.DisplacementMapFilter;
6  import flash.filters.BlurFilter;
7
```

A continuació afegirem un nou element de **BlurFilter** al nostre codi, que anomenarem ***desenfoque***.

```
desplazar = new DisplacementMapFilter(reflejo, punto, 15, 1,
desplazar2 = new DisplacementMapFilter(reflejo2, punto2, 30,
desplazar3 = new DisplacementMapFilter(reflejo3, punto3, 50,

var desenfoque:BlurFilter = new BlurFilter (12, 12, 3);

onEnterFrame = function () {
    punto.x -= 0.3;
    punto.y -= 0.1;
```

Els paràmetres que admet BlurFilter són els següents:

- ***blurX***. Indica el grau de desenfocament que sofrirà la imatge en horitzontal.

- ***blurY***. Indica el grau de desenfocament que sofrirà la imatge en vertical.

- ***quality***. Indica la quantitat de vegades que s'aplicarà el desenfocament. El valor predeterminat és 1. Encara que els valors acceptats van del 0 al 15, com més alt és aquest valor més és el temps de processament que necessitarà el resultat per a mostrar-se correctament.

Per a finalitzar l'aplicació d'aquest filtre ja només ens queda incloure'l en l'apartat corresponent a filtres dins de la funció d'actualització. Això deixaria el codi completat de la manera següent:

```
import flash.display.BitmapData;
import flash.geom.Rectangle;
import flash.geom.Point;
import flash.filters.BitmapFilter;
import flash.filters.DisplacementMapFilter;
import flash.filters.BlurFilter;

img = BitmapData.loadBitmap("imagen");
this.attachBitmap(img,this.getNextHighestDepth());


var ancho = img.width;
var alto = img.height;

this.createEmptyMovieClip("imagen_ref",this.getNextHighestDepth());
this.imagen_ref.attachBitmap(img,this.getNextHighestDepth());

imagen_ref._y = img.height*2;
imagen_ref._yscale = -100;

reflejo = new BitmapData(ancho, alto/6);
reflejo2 = new BitmapData(ancho, alto/3);
reflejo3 = new BitmapData(ancho, alto/2);

var punto = new Point(0, 0);
var punto2 = new Point(0, alto/6);
var punto3 = new Point(0, alto/2);

desplazar = new DisplacementMapFilter(reflejo, punto, 15, 1, 20, 25, "clamp");
desplazar2 = new DisplacementMapFilter(reflejo2, punto2, 30, 1, 30, 35, "clamp");
desplazar3 = new DisplacementMapFilter(reflejo3, punto3, 50, 1, 40, 50, "clamp");

var desenfoque:BlurFilter = new BlurFilter (12, 12, 3);

onEnterFrame = function () {
    punto.x -= 0.3;
    punto.y -= 0.1;
    punto2.x -= 0.8;
    punto2.y -= 0.5;
    punto3.x -= 1.5;
    punto3.y -= 0.9;
    reflejo.perlinNoise(3,2,1,0,true,true,4,true,[punto]);
    reflejo2.perlinNoise(5,2,2,0,true,true,4,true,[punto2]);
    reflejo3.perlinNoise(9,2,4,0,true,true,4,true,[punto3]);
    imagen_ref.filters = [desenfoque, desplazar, desplazar2, desplazar3];
};

imagen_ref.dispose();
```

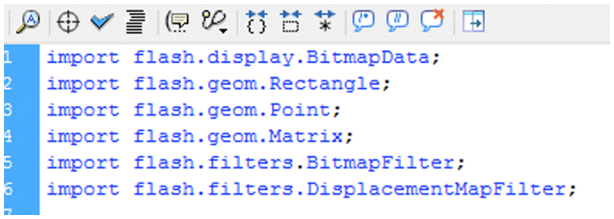
A continuació podeu veure un dels possibles resultats obtinguts abans d'aplicar el filtre de desenfocament i després d'això.

11. Simular vent

Simular els efectes del vent en temps de disseny en una bandera o en un element qualsevol que vulguem moure pot ser molt complicat, especialment si volem que tingui un moviment suau. Per contra, simular aquest mateix efecte en temps d'execució mitjançant l'ActionScript pot ser força més senzill.

1) Obriu l'arxiu **viento0.fla** que es troba en la carpeta de recursos. Fixeu-vos que en la Biblioteca del projecte es troba allotjada una imatge amb l'identificador de vinculació ja introduït.

2) Començarem com sempre per importar les classes que necessitarem.



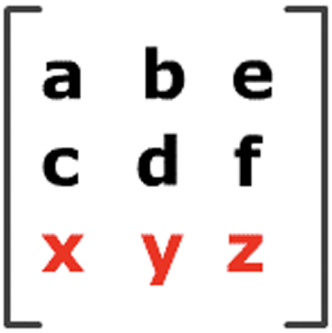
```
1 import flash.display.BitmapData;
2 import flash.geom.Rectangle;
3 import flash.geom.Point;
4 import flash.geom.Matrix;
5 import flash.filters.BitmapFilter;
6 import flash.filters.DisplacementMapFilter;
```

Fixeu-vos que aquesta vegada en la llista de classes importades apareix un element del qual ja hem parlat, però que havíem deixat per a més endavant: és l'element **Matrix**.

> Un objecte de la classe Matrix representa una matriu de transformació que determina com s'assignen punts concrets d'un espai de coordenades a l'altre.

En definir les propietats de l'objecte Matrix i aplicar-les a objectes de les classes **MovieClip** o **BitmapData**, és possible fer diverses transformacions gràfiques en aquests objectes, com poden ser translacions, girs, canvis d'escala o esbiaixats.

Les matrius d'un objecte Matrix tenen un contingut distribuït en tres files per tres columnes, les quals responen al contingut següent:



- Les dades dels camps **a**, **b**, **c**, **d**, **e** i **f** són dades numèriques que l'usuari ha d'introduir en l'ordre que descriuen les lletres.

- Els camps corresponents a les lletres ***x***, ***y*** i ***z*** no funcionen en entorns bidimensionals i per això no és ni possible ni necessari introduir-los. El Flash adjudica automàticament a aquests camps el valor 0 (camps ***x*** i ***y***) i el valor 1 (camp *z*).

Si consulteu l'ajuda del Flash trobareu matrius que compleixen funcions específiques, com poden ser girar o esbiaixar un element.

3) Ara que ja tots sabem com funciona un objecte d'aquest tipus, reprenguem el codi inicial. Hem vist que a mesura que hem avançat s'ha fet necessari importar cada vegada més i més elements. Això, a més de ser un inconvenient, pot significar que ens en deixem algun i l'aplicació no funcioni. Així, doncs, simplificarem aquesta part del codi i la deixarem de la manera següent:

```
import flash.display.*;
import flash.geom.*;
import flash.filters.*;
```

Notació amb asterisc

La notació amb asterisc el que fa és importar tot el contingut de cada paquet. D'aquesta manera és molt més senzill d'escriure i és més complicat cometre errors.

A continuació crearem l'objecte de **BitmapData** a partir de la imatge que hi ha en la Biblioteca i, com ja hem fet altres vegades, establirem unes variables d'amplada i altura per determinar la posició que ocuparà el resultat final en l'escenari.

```
1  import flash.display.*;
2  import flash.geom.*;
3  import flash.filters.*;
4
5  img = BitmapData.loadBitmap("imagen");
6
7  var ancho = img.width;
8  var alto = 7*img.height/5;
9
```

Per a crear l'efecte de vent necessitarem tres nous mapes de bits que crearem a partir del primer.

Començarem per crear el primer i traçar-lo usant el mètode **draw**, que ja hem fet servir abans. Recordeu que aquest mètode permet traçar una imatge d'origen en una imatge de destinació i que aquesta imatge de destinació estigui controlada per una matriu de transformació.

Per completar aquest pas crearem el clip contenidor del mapa de bits i l'hi afegirem.

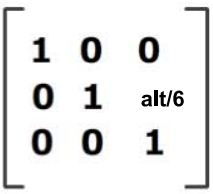
Així, doncs, el codi a introduir seria el següent:

```
1  import flash.display.*;
2  import flash.geom.*;
3  import flash.filters.*;
4
5  img = BitmapData.loadBitmap("imagen");
6
7  var ancho = img.width;
8  var alto = 7*img.height/5;
9
10 vaiven1 = new BitmapData(ancho, alto, true, 0);
11 vaiven1.draw(img,new Matrix(1, 0, 0, 1, 0, alto/6));
12 this.createEmptyMovieClip("efecto",this.getNextHighestDepth());
13 efecto.attachBitmap(vaiven1,this.getNextHighestDepth());
```

4) Aturem-nos un moment per analitzar els nombres que conté la matriu. Veient-los podem afirmar que es tracta d'una matriu específica de translació, és a dir, que el que farà serà moure l'objecte anomenat ***vaiven1*** al llarg de l'eix de la *Y*.

Estructura general d'una matriu.

Estructura típica d'una matriu de translació.



Estructura de la matriu escrita. Fixeu-vos que el valor del paràmetre de translació en l'eix X és zero.

5) Ara que ja tenim amb les seves característiques de translació descrites el primer dels tres objectes que necessitem, el que farem serà dues còpies d'aquest element. Per a això recorrerem al mètode clone. L'aplicació d'aquest mètode dóna com a resultat un objecte nou que és un clon exacte de la instància original amb una còpia exacta del mapa de bits contingut en aquesta instància.

```
import flash.display.*;
import flash.geom.*;
import flash.filters.*;

img = BitmapData.loadBitmap("imagen");

var ancho = img.width;
var alto = 7*img.height/5;

vaiven1 = new BitmapData(ancho, alto, true, 0);
vaiven1.draw(img,new Matrix(1, 0, 0, 1, 0, alto/6));
this.createEmptyMovieClip("efecto",this.getNextHighestDepth());
efecto.attachBitmap(vaiven1,this.getNextHighestDepth());

vaiven2 = vaiven1.clone();
vaiven3 = vaiven1.clone();

```

6) A continuació, crearem el mapa de desplaçament. Aplicarem aquest mapa al segon mapa de bits que hem creat. Recordeu que un dels paràmetres d'aquest objecte és l'objecte **Point**; així, doncs, abans de crear aquest mapa necessitarem crear aquest objecte.

```
import flash.display.*;
import flash.geom.*;
import flash.filters.*;

img = BitmapData.loadBitmap("imagen");

var ancho = img.width;
var alto = 7*img.height/5;

vaiven1 = new BitmapData(ancho, alto, true, 0);
vaiven1.draw(img,new Matrix(1, 0, 0, 1, 0, alto/6));
this.createEmptyMovieClip("efecto",this.getNextHighestDepth());
efecto.attachBitmap(vaiven1,this.getNextHighestDepth());

vaiven2 = vaiven1.clone();
vaiven3 = vaiven1.clone();

punto = new Point();
desplazamiento = new DisplacementMapFilter(vaiven2, punto, 8, 8, 24, 20, "clamp");

```

7) Amb això ja hem acabat la fase de creació. Ara és el moment d'entrar a construir la funció d'actualització que generarà el moviment de la imatge.

Tal com hem fet abans alguna vegada, primer establirem valors per a les coordenades *X* i *Y* de l'objecte punt, i una vegada fet això aplicarem **perlinNoise** a les instàncies **vaiven1** i **vaiven2**. Finalment aplicarem la propietat ***Filters***.

La funció de construcció quedarà així:

```
onEnterFrame = function () {
    punto.x -= 3*ancho/50;
    punto.y -= 4*ancho/100;
    vaiven1.perlinNoise(ancho,alto,1,0,true,true,1 | 2 | 4,true,[punto]);
    vaiven2.perlinNoise(ancho,alto,1,0,true,true,8,true,[punto]);
    efecto.filters = [desplazamiento];
};
```

És important observar les diferències que en **perlinNoise** hem indicat per a les dues instàncies. Aquestes diferències rauen en els canals de color de destinació. Mentre que el soroll de la instància **vaiven1** afectarà els canals de color, el soroll aplicat a **vaiven2** únicament afectarà el canal de transparència.

Si ara proveu el projecte Flash, podreu comprovar que únicament es mostra una espècie de fum. El que estareu veient en realitat és el clip **vaiven2** perquè aquest es troba en un nivell superior.

Per tant, el que farem serà copiar la informació de la instància **vaiven1** en el tercer clip que havíem creat, **vaiven3**. Tanmateix, no farem un **clone** com abans quan hem creat aquesta instància sinó que aquesta vegada el que farem serà barrejar la informació dels canals de color. Per a això recorrerem al **mètode merge**. Aquest mètode admet els paràmetres següents:

- ***sourceBitmap***. Indica la imatge de mapa de bits que s'utilitzarà.

- ***sourceRect***. És un objecte Rectangle que defineix l'àrea de la imatge d'origen que s'utilitzarà com a entrada.

- ***destPoint***. Indica el punt de la imatge de destinació que correspon a la cantonada superior esquerra del rectangle d'origen.

- ***redMult***. Nombre pel qual es multiplica el valor del canal vermell. Admet valors de 0 a 256. Com més s'apropi a 256 més quantitat d'informació de color de l'arxiu original rebrà aquest canal.

- ***greenMult***. Nombre pel qual es multiplica el valor del canal verd. Admet valors de 0 a 256. Com més s'apropi a 256 més quantitat d'informació de color de l'arxiu original rebrà aquest canal.

- ***blueMult***. Nombre pel qual es multiplica el valor del canal blau. Admet valors de 0 a 256. Com més s'apropi a 256 més quantitat d'informació de color de l'arxiu original rebrà aquest canal.

- ***alphaMult***. Nombre pel qual es multiplica el valor del canal de transparència. Admet valors de 0 a 256. Com més s'apropi a 256 més quantitat d'informació de transparència de l'arxiu original rebrà aquest canal.

8) Així, doncs, dins de la funció escriurem el codi que permeti la barreja entre les instàncies **vaiven1** i **vaiven3**.

```
onEnterFrame = function () {
    punto.x -= 3*ancho/50;
    punto.y -= 4*ancho/100;
    vaiven1.perlinNoise(ancho,alto,1,0,true,true,1 | 2 | 4,true,[punto]);
    vaiven2.perlinNoise(ancho,alto,1,0,true,true,8,true,[punto]);
    vaiven1.merge(vaiven3,new Rectangle(0, 0, ancho, alto),new Point(),180,180,180,256);
    efecto.filters = [desplazamiento];
};
```

Fixeu-vos que els valors de barreja dels canals de color RGB s'han establert en 180. Això permet que pugui veure's una mica l'efecte aplicat sobre **vaiven2**. Per a valors propers a 256, la visualització del resultat serà més plana, ja que no deixarà veure l'efecte perlín de **vaiven2**. Per a valors molt baixos l'únic que es veuria seria l'efecte aplicat a **vaiven2**.

Finalment podem eliminar el mapa de bits original mitjançant **img.dispose**(), ja que ara no ens fa falta.

El codi final d'aquest projecte quedaria, doncs, de la manera següent:

```
import flash.display.*;
import flash.geom.*;
import flash.filters.*;

img = BitmapData.loadBitmap("imagen");

var ancho = img.width;
var alto = 7*img.height/5;

vaiven1 = new BitmapData(ancho, alto, true, 0);
vaiven1.draw(img,new Matrix(1, 0, 0, 1, 0, alto/6));
this.createEmptyMovieClip("efecto",this.getNextHighestDepth());
efecto.attachBitmap(vaiven1,this.getNextHighestDepth());

vaiven2 = vaiven1.clone();
vaiven3 = vaiven1.clone();

punto = new Point();
desplazamiento = new DisplacementMapFilter(vaiven2, punto, 8, 8, 24, 20, "clamp");

onEnterFrame = function () {
    punto.x -= 3*ancho/50;
    punto.y -= 4*ancho/100;
    vaiven1.perlinNoise(ancho,alto,1,0,true,true,1 | 2 | 4,true,[punto]);
    vaiven2.perlinNoise(ancho,alto,1,0,true,true,8,true,[punto]);
    vaiven1.merge(vaiven3,new Rectangle(0, 0, ancho, alto),new Point(),180,180,180,256);
    efecto.filters = [desplazamiento];
};

img.dispose();
```

12. Algunes diferències en les classes d'AS 3.0

Fins ara hem vist què són i com funcionen alguns membres de classe en AS2.0. Al llarg d'aquest apartat veurem algunes de les diferències principals en la manera de declarar la classes AS2.0 i AS3.0.

En l'**ActionScript 3.0** totes les classes s'han de col·locar en **paquets** (*package*).

> Un paquet és una manera d'organitzar classes en grups, ja que cada paquet s'organitza com un sistema d'arxius.

Els paquets s'ordenen en relació amb la via d'accés a les classes, la qual és definida com una ruta relativa al projecte Flash. Això fa que la crida a una classe sigui una cosa tan simple com introduir la ruta des de la qual es troba el projecte Flash fins al lloc en què s'ubica l'arxiu de classe.

En **AS2.0** podíem incloure en l'escenari elements que no estiguessin inclosos en un contenidor. Per exemple, una imatge es podia cridar mitjançant el mètode **loadMovie** i quedava col·locada en l'escenari. En **AS3.0** això no és possible, ja que aquesta imatge no es visualitzarà mentre no estigui inclosa dins d'un contenidor.

La inclusió d'elements en un contenidor és una cosa que hem après àmpliament en aquesta assignatura però que en AS3.0 ha variat una mica, ja que, per exemple, s'ha eliminat el mètode **attachMovie**()i ha passat a ser **addChild**().

Un altre dels elements que hem usat moltes vegades i que també ha desaparegut en AS3.0 és el mètode **loadBitmap**. En aquest cas la desaparició es deu a l'aparició de la pròpia **classe Bitmap**.

12.1. Les classes BitmapData i Bitmap

Per la seva banda, la classe **BitmapData** continua reunint característiques molt semblants a les que ja tenia en AS2.0, ja que la seva funció principal continua essent la de permetre treballar amb els píxels, en aquest cas no solament d'un objecte pròpiament de **BitmapData** sinó també d'un objecte de la nova classe Bitmap.

La classe Bitmap no estava disponible en AS2.0. Aquesta nova classe permet crear i representar objectes de visualització que representen imatges de mapa de bits.

El constructor Bitmap() permet crear un objecte Bitmap que conté una referència a un objecte BitmapData.

Després de crear l'objecte Bitmap és imprescindible recórrer al mètode **addChild**() o **addChildAt**() de la instància del contenidor per a poder veure el mapa de bits en l'escenari del Flash Player.

Un objecte Bitmap pot compartir la seva referència a **BitmapData** amb molts altres objectes Bitmap. Aquest fet no n'afectarà les propietats individuals. Així, doncs, cada objecte Bitmap podrà tenir propietats diferents d'un altre encara que tots dos facin referència al mateix objecte de **BitmapData**. Això permet treballar amb objectes de **BitmapData** complexos sense necessitat de sobrecarregar l'ús de la memòria pel fet d'haver de crear una instància per a cada objecte de visualització.

Un objecte Bitmap pot dibuixar en pantalla un objecte **BitmapData** de dues maneres: amb el **processador de vectors** com a figura de farciment de mapa de bits (aquest seria el procediment que usa AS2.0), o amb una **rutina de còpia de píxels**, la qual cosa agilita molt més el procés. Per poder traçar el Bitmap mitjançant aquesta rutina s'han de complir les condicions següents:

- No es podran aplicar escalats, rotacions o esbiaixats a l'objecte Bitmap.
- No es pot aplicar transformació de color a l'objecte Bitmap.
- No es pot aplicar mode de barreja a l'objecte Bitmap.
- No es pot fer cap retallada mitjançant capes de màscara o mètodes com setMask().
- La imatge en si mateixa no pot ser una màscara.
- Les coordenades de destinació han d'estar en un límit d'un píxel complet. No s'admeten fraccions de píxel.

12.2. Mètodes dels contenidors en AS3.0

Ja hem esmentat anteriorment que qualsevol objecte que s'hagi d'incorporar en temps d'execució haurà d'estar inclòs dins d'un contenidor o altrament no serà visible en l'escenari. A continuació trobareu una llista amb els mètodes més importants que afecten el contingut dels contenidors d'objectes.

- **addChild()**. És probablement el mètode més important d'AS3.0, ja que qualsevol cosa que vulguem afegir a l'escenari ha de portar incorporat aquest mètode. En el codi següent es pot veure un exemple de com s'ha d'agregar una imatge en un contenidor perquè sigui visualitzada en l'escenari del Flash Player.

```
var carga:Loader = new Loader();
   carga.load(new URLRequest("imagen_Cualquiera.jpg"));
```

```
        this.addChild(carga);
```

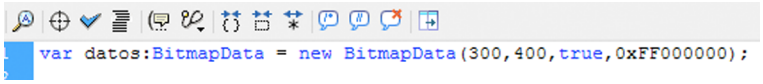
- ***numChildren***. Ens permet saber de quants elements consta un contenidor.
- ***getChildAt()***. Ens indica l'objecte que ocupa la posició que indiquem.
- ***removeChildAt()***. Extreu del contenidor l'objecte indicat. Aquest objecte no és eliminat sinó que únicament s'ha extret del contenidor, de manera que deixa de ser visible. Per eliminar-lo s'ha d'utilitzar el mètode **delete**.
- ***getChildByName()***. Busca l'objecte que tingui com a identificador el nom indicat.
- ***setChildIndex()***. Canvia la posició de l'objecte indicat fins a la posició que indiquem.
- ***addChildAt()***. Agrega un element en el lloc que indiquem d'un contenidor que ja existeix.
- ***getChildIndex()***. Torna les dades de la posició en què es troba l'element indicat.

12.2.1. Creació d'un mapa de bits i addició d'efectes

Ja hem dit fins ara que hi ha algunes diferències en la forma de creació i farciment de mapes de bits. Al llarg d'aquesta etapa en veurem un exemple.

1) Creeu un arxiu nou i importeu a l'escenari una imatge qualsevol perquè us serveixi de fons. El que farem a continuació serà crear un mapa de bits buit al qual aplicarem un filtre de soroll que barrejarà canals de color amb la imatge inferior.

2) Començarem per crear el nou objecte de **BitmapData**.



```
var datos:BitmapData = new BitmapData(300,400,true,0xFF000000);
```

Observem ja una diferència en la forma de creació d'aquest objecte: va seguit de dos punts i una declaració del tipus de classe a què pertany, en aquest cas BitmapData. Aquesta és la manera correcta de crear objectes en ActionScript 3.

AS2

Encara que en AS2 ja s'usava una notació similar per a la creació d'objectes, aquesta notació no incloïa la declaració del tipus classe de l'objecte que estàvem creant.

3) A continuació crearem el **nou objecte Bitmap** i el vincularem a l'objecte les dades del qual pertanyen a **BitmapData**.

```
var datos:BitmapData = new BitmapData(300,400,true,0xFF000000);
var bmp:Bitmap = new Bitmap(datos);
```

4) Per finalitzar aquesta part del codi indicarem el mode de barreja que volem que es produeixi entre aquest objecte Bitmap i la imatge que hem col·locat com a fons. Una vegada fet això afegirem l'objecte Bitmap a l'escenari.

```
var datos:BitmapData = new BitmapData(300,400,true,0xFF000000);
var bmp:Bitmap = new Bitmap(datos);
bmp.blendMode = BlendMode.SCREEN
addChild(bmp);
```

Observeu el senzill que és en AS3 indicar que es produeixi una barreja entre dos mapes de bits. En el cas de l'exemple hem escollit Screen com a mode de barreja, tanmateix n'hi ha molts més. Si voleu, els podeu recordar anant a l'apartat vuitè.

5) Ara crearem la funció d'actualització igual com ja hem fet altres vegades. Tanmateix, la manera de declarar aquesta funció potser és la part que més ha canviat entre AS2 i AS3.

```
onEnterFrame = function (crearRuido) {

}

stage.addEventListener(Event.ENTER_FRAME,crearRuido);
function crearRuido(event:Event){

}
```

Format de declaració en AS2

Format de declaració en AS2

És important observar que es declara el tipus d'esdeveniment de manera independent de la funció i que aquest esdeveniment s'afegeix a l'escenari de l'objecte de visualització identificat com a ***stage*** i del qual ja havíem parlat en capítols anteriors.

6) Un cop feta aquesta puntualització sobre la forma de la notació d'una funció en AS3, ja podrem omplir els apartats que volem que tingui aquesta funció.

En aquest cas, els apartats seran, d'una banda, la quantitat de transparència que volem donar a l'objecte Bitmap perquè es pugui mostrar així la imatge de fons i, de l'altra, els paràmetres que tindrà el soroll generat.

Així, doncs, la funció soroll en l'aspecte més bàsic podria quedar, per exemple, així:

```
function crearRuido(event:Event)
{
    bmp.alpha = 10;
    datos.noise(100,70,220,8,true);
}
```

La propietat **alpha** indica el grau de transparència d'un objecte. El seu comportament és el mateix que l'antiga propietat **_alpha**. Admet únicament camps numèrics.

El mètode **noise** omple un Bitmap amb píxels que mostren un soroll aleatori. Els seus paràmetres de construcció són:

- ***randomSeed***. Indica el nombre d'inicialització aleatori que s'utilitzarà. La funció de soroll és una funció d'assignació i no una vertadera funció de generació de soroll aleatori. Com a conseqüència d'això sempre produirà els mateixos resultats amb el mateix valor d'inicialització.

- ***low***. Indica el mínim valor de color que es generarà per a cada canal. Aquests valors poden oscil·lar entre 0 i 255.

- ***high***. Indica el màxim valor de color que es generarà per a cada canal. Aquests valors poden oscil·lar entre 0 i 255.

- ***channelOptions***. Indica un nombre que pot ser una combinació de qualsevol dels quatre valors de canal de color. Igual com passava en AS2, és possible usar l'operador lògic (|) per a combinar valors entre canals.

- ***grayScale:****Boolean*. El seu funcionament és idèntic al que ja tenia en AS2. Indica un valor booleà i per defecte sempre està en *false*. Si s'indica com a vertader (*true*), es crearà una imatge en escala de grisos i s'establiran tots els canals de color amb el mateix valor.

A continuació podeu veure un codi una mica més refinat de la funció. Fixeu-vos que amb la finalitat d'obtenir resultats més aleatoris en cada actualització s'han introduït unes variables que determinaran els valors del soroll en cada fotograma.

```
function crearRuido(event:Event)
{
    var base_ruido:uint = Math.random() * 100;
    var valor_bajo:uint = Math.random() * 70;
    var valor_alto:uint = Math.random() * 220;
    bmp.alpha = .5 + Math.random() * .5;
    datos.noise(base_ruido,valor_bajo,valor_alto,8,true);
}
```

És important observar que en aquest cas també ha aparegut la partícula var davant de la variable i que s'ha declarat que pertany a la classe **uint**, que és de nivell superior com la classe **int**. Totes dues classes permeten treballar amb dades de nombres enters de fins a 32 bits.

Ja per finalitzar, podeu veure tot el codi que hem usat en aquest apartat. És con venient que proveu amb diferents paràmetres i amb altres tipus de barreja. Es poden obtenir resultats espectaculars sense gaire esforç.

```
var datos:BitmapData = new BitmapData(300,400,true,0xFF000000);
var bmp:Bitmap = new Bitmap(datos);
bmp.blendMode = BlendMode.SCREEN
addChild(bmp);

stage.addEventListener(Event.ENTER_FRAME,crearRuido);

function crearRuido(event:Event)
{
    var base_ruido:uint = Math.random() * 100;
    var valor_bajo:uint = Math.random() * 70;
    var valor_alto:uint = Math.random() * 220;
    bmp.alpha = .5 + Math.random() * .5;
    datos.noise(base_ruido,valor_bajo,valor_alto,8,true);
}
```