5.3. Relacionamento entre Classes

Neste momento a nossa class passa a pertencer ao grupo de classes pessoais que podemos adicionar ao nosso projecto.
Um dos pontos muiti utéis é o reaproveitamento do código para as mais variadas coisas, e de entre as quais podemos usar um relacionamento entre varias classes pessoais.

Um exemplo pode ser a definição de certas constantes numa class. Vamos supor que necessitamos de ter definidos tamanhos e um titulo para varios paineis feitos pela class em cima, poderemos efectuar 2 tipos de classes para este efeito, embora para este tipo de definições possamos usar uma class sem inicializador (apenas servirá para definir constantes / variaáveis)

```
package teste
{
	public class def
	{
		public const PANEL_W:int = 400;
		public const PANEL_H:int = 300;
		public const PANEL_TITLE:String = "Painel personalizado";
	}
}
```

Esta class não necessita de ser inicializada porque apenas apresenta parametros fixos da nossa aplicação e não têm um construtor.

Na nossa class em cima vamos colocar em prática esta nossa calss, editando a class teste.as e fazendo o seguinte.
No topo a seguir aos imports coloquem:

```
import teste.def;
```

e na nossa função criado() substituimos a linha:

```
_nomePessoal="Teste Painel "+Math.floor(Math.random()*15);
```

por

```
_nomePessoal=def.PANEL_TITLE+Math.floor(Math.random()*15);
```

Neste momento o titlo de cada painel criado será iniciado pela definição colocada na class def:

```
public const PANEL_TITLE:String = "Painel personalizado";
```

vamos também definir o tamanho fixo do nosso painel da class, trocando na função teste() o seguinte:

```
//definimos o painel
this.width=270;
this.height=270;
```

por

```
this.width=def.PANEL_W;
this.height=def.PANEL_H;
```

Desta forma, todos os painei criados pela nossa class teste terão sempre as definições:

```
public const PANEL_W:int = 400;
public const PANEL_H:int = 300;
public const PANEL_TITLE:String = "Painel personalizado";
```

Este tipo de definições são muito uteis quando há necessidade de ter sempre constantes na nossa aplicação, po exemplo num caso de uma aplicação que se liga a um banco de dados, podemos guardar o host e informações de login.. neste caso, apenas com um problema. Estas constantes apenas podem ser lidas, visto que são constantes e apenas podem ser alteradas pelo cirador da aplicação e compilando de novo a aplicação.

Este trabalho poderia ser feito pela class seguinte:

```
package teste
{
	public class definicoes
	{
		public const PANEL_W:int = 400;
		public const PANEL_H:int = 300;
		public const PANEL_TITLE:String = "Painel personalizado";

		public function definicoes()
		{
		super();
		}

	}
}
```

Mas neste caso teriamos algum código inválido, ou seja, o definicoes() ira ser inicializado em vão.. não temos necessidade de o ter no código (embora o possamos ter)

O action Script 3 é tão versátil que nos permite inclusive extender uma class já feita para trabalhar com outros exemplos, vejam o exemplo seguinte que extende a nossa class de definições.

```
package teste
{
	public class fullDef extends def
	{
		public const PANEL_DATE:String = "20/09/2008";

		public function fullDef()
		{

		}

	}
}
```

Este método cria uma nova class fullDef que extende a nossa class def, por palavras mais simples esta class usa as definições da class def em si própria, ou seja, herda (hereditária) as variaveis da class que está a ser extendida:

```
public const PANEL_W:int = 400;
public const PANEL_H:int = 300;
public const PANEL_TITLE:String = "Painel personalizado";
```

e trata-as como se fossem suas, ou seja estas mesmas variaveis estarão disponiveis na class fullDef, basta testarem da seguinte forma:

```
var fid:fullDef = new fullDef;
```

e a partir deste momento as constantes da nossa class def de definições estarão também disponiveis nesta nossa class, juntamente com a constante PANEL_DATE que adicionei em cima, pela seguinte forma:

```
fid.PANEL_DATE;
fid.PANEL_H;
fid.PANEL_TITLE;
fid.PANEL_DATE;
```

De notar que uma constante não pode ser alterada depois do nosso código ter sido compilado num ficheiro .swf.

Para esse efeito usamos apenas private var, e se o formos fazer usando a nossa class de definições e a class fullDef deveremos usar o método [Bindable] já explicado anteriormente e que vai permitir que todas as classes que utilizem a nossa class def sejam actualizados com novos valores.

Esta extensão funciona com variaveis, constantes, e funções quer das nossas classes pessoais ou das classes do Flex / Flash.

5.4. Re-escrever métodos em classes

Outro ponto muito util é no caso de já termos uma class totalmente feita, mas necessitamos de outra igual mas que substitua ou invalide uma ou outra função da class pai. Este método chama-se override, quem em português será parecido com re-escrever, ou seja, escrever por cima / substituir determinados pontos. De notar que só é possivel utilizar este método override em funções que tenham sido declaradas como public, por exemplo:

```
public function info():void{}
```

Para demonstrar este método, teremos que fazer algumas alterações na nossa classe que criamos anteriormente, já que é esse exemplo que será usado.

Vamos inicar com o exemplo da nossa ultima class criada e vamos escrever o simples código:

```
public function painelInfo():String {
      return PANEL_DATE + " - " + PANEL_H + "-" + PANEL_W + "-" + PANEL_TITLE;
            }
```

Adicionando esta função a nossa class devolveria uma suposta string de informação do nosso painel.

Ficando a class assim:

```
package teste
{
	public class fullDef extends def
	{
		public const PANEL_DATE:String = "20/09/2008";

		public function fullDef()
		{

		}

	public function painelInfo():String {
	return PANEL_DATE + " - " + PANEL_H + "-" + PANEL_W + "-" + PANEL_TITLE;
		}

	}
}
```

O meu objectivo vai ser criar uma class que irá extender esta anterior e evitar que ela devolva qualquer valor quando for chamada a função painelInfo() ou que devolva um valor por mim definido. Para efectuar esta acção exste um método chamado override que vou passar a explicar.

Criando uma nova class que irá extender esta, ficamos asism:

```
package teste
{
	public class customFullDef extends fullDef
	{
		public function customFullDef()
		{
			super();
		}
    }
}
```

aqui a class apenas se encontra extendida, e se a função painelInfo() está disponivel nela e vai devolver:

PANEL_DATE + " - " + PANEL_H + "-" + PANEL_W + "-" + PANEL_TITLE;

podem testar da seguinte forma:

```
var cfdef:customFullDef = new customFullDeff();

trace ("->"+cfdef.painelInfo());
//o resultado será "->20/09/2008 - 300 - 400 - Painel personalizado
```

Quando esta função for chamada, o que eu quero é evitar que esta se comporte da mesma maneira, para isso adiciono:

```
		override protected function painelInfo():String {
			return "Painel Protegido / sem definições";
		}
```

à class, ficando assim:

```
package teste
{
	public class customFullDef extends fullDef
	{
		public function customFullDef()
		{
			super();
		}

		override protected function painelInfo():String {
			return "Painel Protegido / sem definições";
		}
	}
}
```

se agora chamar a class:

```
var cfdef:customFullDef = new customFullDeff();

trace ("->"+cfdef.painelInfo());

//o resultado será "-> Painel Protegido / sem definições"
```

como podem ver podemos interceptar, substituir, incrementar ou até eliminar por completo a utilidade de uma função desde que esta seja publica (public).

No caso em cima foi substituida a resposta normal da nossa class de origem e atribuida uma nova resposta quando essa função for chamada.

5.4.1. Explicação da função Super();

Existem alguns truques que podem ser muio uteis, no caso anterior estavamos a usar variaveis e dados da class extendida (fullDef), mas com esta extensão podemos obter ainda mais funcionalidades, como repararam em certos casos foi usado o metodo super() que na realidade em cima nada faz, e até pode confundir, mas que é importante e muito util em alguns casos principalmente em hierarquias de dados.

O super faz referencia à class que foi extendida, podem de entre muitas utilidades ser usado para chamar a função principal da class extendida, no caso em cima chama a função fullDef() da class fullDef... mas pode ser usado para chamar outras funções de dentro dessa class como no caso em cima
super.painelInfo() e em casos mais uteis passar dados emtre a class e a class extendida, vejam o exemplo:

```
package exemplo
{
	class turma
	{
		public function turma(nome:String)
		{
			trace(nome);
		}
    }
}
```

e uma class extendida:

```
package exemplo
{
	class aluno extends turma
	{
		public function aluno(nome:String, nometurma:String)
		{
			super(nometurma);
			trace(nome);
		}
    }
}
```

Neste caso o super(nometurma) vai passar o nome da turma para a class extendida, e executa-la, ou seja, se fizermos:

```
var alunoInfo:aluno = new aluno("paulo","5 ano – D");
```

vamos obter 2 trace's no painel de debug :

paulo
5 ano – D

Neste momento podem não estar a ver grande utilidade neste tipo de operações faladas anteriormente, como o interface, implements, override mas em projectos derivados das vossas classes, projectos de grupo, criação de componentes pessoais e muitas outras coisas este tipo de operações passam a ser indispensáveis.

Como viram é possivel criar definições, usa-la, implementa-las, extende-las e até modifica-las (override), e muito por causa destas operações é que o Action Script 3 é considerado uma verdadeira jóia dos programadores, principalmente orientados ao open-source.

5.5. Classes, Exemplo Painel personalizado

Vamos a um exemplo prático, usando o flex o meu objectivo é fazer o seguinte:

Criar um painel personalizado que permita:

- ser arrastado (drag & drop) na nossa aplicação
- ter botões de maximizar, minimizar e restaurar
- Adicionar efeitos de maximizar, minimizar e restaurar.

Vamos aplicar todos os conhecimentos adquiridos anteriormente bem como a utilização de algumas classes nativas de efeitos disponiveis no flex como efeitos que mais à frente falaremos. Este exemplo apenas funcionará no Flex, visto que o AS3 do flash ainda não suporta alguns pontos e classes de efeitos que foram adicionadas no Flex.

Vejam o seguinte código que foi devidamente comentado para perceberem facilmente toda a sua estrutura.

```
package com.msdevstudio
{
	//imports necessários ao nosso painel / class
	//Eventos
	import flash.events.Event;
	import flash.events.MouseEvent;

	import mx.containers.Panel;
	import mx.controls.Image;
	import mx.core.Application;
	import mx.effects.Move;
	import mx.effects.Parallel;
	import mx.effects.Resize;

	//extndemos o nosso painel
	public class customPanel extends Panel
	{
		//incluimos as nossas imagens de maximizar, minimizar e resturar
		//para a resposta do carregamento / troca das imagens seja imediata

		[Embed(source="maxBut.png")]
		public var max:Class;

		[Embed(source="minBut.png")]
		public var min:Class;

		[Embed(source="restBut.png")]
		public var rest:Class;

		//variaveis
		//estados do painel

		private var maximizado:Boolean = false;
		private var minimizado:Boolean = false;

		//imagens dos botões
		private var minBut:Image;
		private var maxBut:Image;


		//usado para guardar os tamanhos e posições originais do nosso panel

		private var originalSizeW:int;
		private var originalSizeH:int;
		private var originalPosX:int;
		private var originalPosY:int;



		//construtor da nossa class
		public function customPanel()
		{
		//definimos alguns estilos do nosso panel para ficar mais agradável

			this.setStyle("headerHeight",19);
			this.setStyle("borderThicknessBottom", 0);
			this.setStyle("borderThicknessLeft", 0);
			this.setStyle("borderThicknessTop", 0);
			this.setStyle("borderThicknessRight", 0);
			this.title="Painel 1";
		}

		//re-escrevemos a função createChildren que o componente painel usa
		//para definir o novo layout que ele terá, ou seja vamos adicionar
```

```
		//os botões

		override protected function createChildren():void {
			super.createChildren();

			//guardamos as posições e tamanhos originais
			originalSizeW=this.unscaledHeight;
			originalSizeH=this.unscaledHeight;
			originalPosX=this.x;
			originalPosY=this.y;

			//apenas vamos criar os botões se eles não existirem, temos
			//que fazer isto porque esta função é chamada sempre que é
			//adicionado um elemento novo ao painel, no caso de
			//adicionarem uma caixa de texto esta função é chamada de novo
			//e se não indicasse-mos o "if" ele iria criar novos botões
			//sempre que fosse chamada

			//verificamos se o botão de minimizar existe
			if(!minBut){
			//criamos a nova imagem
			minBut=new Image;

			//atribuimos a class (imagem) min ao nosso botão e definimos
			//alguns parametros
			minBut.source=min;
			minBut.visible=true;
			minBut.toolTip="Minimiza";

			//a imagem mostra o icon por defeito (seta), usamos os 3
			//metodos em baixo para que o botão se comporte como um icon e
			//apresente o cursor "hand" / mão.
			minBut.mouseChildren=false;
			minBut.useHandCursor=true;
			minBut.buttonMode=true;
			//Evento para detectar quando foi clicada.
minBut.addEventListener(MouseEvent.CLICK, minOnClick, false, 0, true);

//adicionamos como filho. Aqui adicionamos no metodo rawChildrem que faz
//um "overlay" à nossa imagem e vai apresenta-la em cima do painel, se
//usarmos apenas this.addChild() o nosso botão iria aparecer dentro da
//area util do painel e não no topo. Poderiamos usar também
//this.titleBar.addChild() onde a titlebar é o topo do nosso painel

			this.rawChildren.addChild(minBut);
			}



			//o mesmo procedimento para a imagem de maximizar, notem que
			//não defini posições, elas serão definidas mais abaixo.
			if(!maxBut) {
			maxBut=new Image;
			maxBut.visible=true;
			maxBut.source=max;
			maxBut.toolTip="Maximiza";
			maxBut.mouseChildren=false;
			maxBut.useHandCursor=true;
			maxBut.buttonMode=true;
		maxBut.addEventListener(MouseEvent.CLICK, onClick, false, 0, true);
			this.rawChildren.addChild(maxBut);
			}

			//adicionamos um listner para saber quando o titleBar foi
```

```
                    //clicado para iniciar o drag e drop

this.titleBar.addEventListener(MouseEvent.MOUSE_DOWN, beginDrag,false,0,true);
this.titleBar.addEventListener(MouseEvent.MOUSE_UP, endDrag,false,0,true);

                    // actualiza a displayList.
                    invalidateDisplayList();
              }

              //aqui vamos subscrever o metdo chamado pelo invalidateDisplayList()
              //que vai ser chamado sempre que existam modificações a fazer no
              // nosso layout, em cima como lhe adcionei duas imagens forcei a
              //chamada da função em baixo para poder definir as posições dos meus
              //botões.

protected override function updateDisplayList(unscaledWidth:Number,
unscaledHeight:Number):void{

                    super.updateDisplayList(unscaledWidth, unscaledHeight);
                    //definimos os tamanhos da nossa imagem
                    //ela por defeito está com 16px por 16px mas vamos certificar-
                    //nos que elas ficam com esses tamanhos
                    minBut.setActualSize(16,16);
                    maxBut.setActualSize(16,16);

                    //vamos definir a posição x e y dos nossos botões
                    // o unscaledWidth indica-nos o tamanho "on-execution" do
                    //nosso painel, ou seja o tamanho actual, poderiamos usar o
                    // this.width, mas provocaria falhas no posicionamento visto
                    //que se o painel for aumentado esta actualização do
                    //this.height teria que aguardar pela actualização da
                    //displayList, ou seja, o tamanho final depois de maximizado
                    //apenas estaria disponivel no final da execução desta função.

                    // colocamos o botão de minimizar à direita do painel, mas
                    //deixamos um espaço em px para o botão de maximizar
                    // (-maxBut.width)

                    var xPos:Number = unscaledWidth - (minBut.width+maxBut.width);

                    // o botão ficará a 2px do topo
                    var yPos:Number = 2;

                    //posicionamos o botão na devida posição
                    minBut.move(xPos, yPos);

                    //usamos as mesmas variaveis em cima, mas agora para
                    //posicionar o botão de maximizar

                    // como vai ficar à direita subtraimos apenas o seu tamanho ao
                    //total do nosso painel
                    xPos=unscaledWidth - maxBut.width;
                    yPos=2;
                    //subtraimos 4px para que o botão fiquem mais perto do
                    //minimizar e mais afastado da lateral direita
                    maxBut.move(xPos-4,yPos);
              }


              // chamada para iniciar o drag do panel
              private function beginDrag(evt:MouseEvent):void {
                    this.startDrag();
              }
```

```
// chamada para parar o drag do panel
private function endDrag(evt:MouseEvent):void {
      this.stopDrag();

      //ao parar temos que guardar a posição em que o painel foi
      //largado para que ao restaurar o painel ir para a ultima
      //posição onde foi deixado.

      originalPosX=this.x;
      originalPosY=this.y;
}

//chamada quando o botão de minimizar é clicado.
private function minOnClick(evt:MouseEvent):void {

//vamos inicializar alguns efeitos para executar na transição do
//nosso painel para o estado minimizado
//paralel, usado para exectuatr mais que um efeito ao mesmo tempo
//os efeitos executados por este paralel serão so seus child's
//(efeitos)

var ambos:Parallel = new Parallel;

//efeito de mover
var moveP:Move = new Move;

//efeito de dimensionar
var dim:Resize = new Resize;

//definimos o target, a instancia, que será alvo do nossos efeitos
//dentro do paralel
ambos.target=this;

//movemos o painel para o x=0;
moveP.xTo=0;
// a nivel de y, movemos para o fundo da aplicação deixando apenas
//19px de altura
// que serão os mesmos 19px com que icará o painel de altura depois
//de o efeito resize
moveP.yTo=Application.application.height-19;

//efeito para dimensionar, alture=19, comprimento=150
dim.heightTo=19;
dim.widthTo=150;

//adiciona-mos o efeito de mover e dimensionar ao nosso paralel
ambos.addChild(moveP);
ambos.addChild(dim);
//duração do efeito
ambos.duration=250;
//iniciamos o efeito
ambos.play();

//usamos a variavel para indicar que o estado de minimizado está
//activo
minimizado=true;
// como está minimizado o botão de maximizar para a ser substituido
//pela imagem de restaurar
maxBut.source=rest;
//toolTip informativa
maxBut.toolTip="Restaura";

}
//Funcção chamado quando o botão de maximizar é clicado.
```

```
		private function onClick(evt:MouseEvent):void {
			//usamos as mesmas instancia de efeitos que no minimiza
			var ambos:Parallel = new Parallel;
			var dim:Resize = new Resize;
			var moveP:Move = new Move;

			ambos.target=this;

			//O painel vai-se comportar da mesma maneira caso esteja
			//minimizado ou maximizado
			//já que a animação vai ser sempre a mesma = restaurar
			if(maximizado==true || minimizado==true ) {
			//definimos a altura e posição originais do nosso painel
			//estas posições podem ser afectadas pelo drop, por isso sao
			//automaticamente actualizadas
				dim.heightTo=originalSizeH;
				dim.widthTo=originalSizeW;
				moveP.xTo=originalPosX;
				moveP.yTo=originalPosY;

				//passa a estar no estado original
				maximizado=false;
				minimizado=false;
				//mudamos a imagem para maximizar
				maxBut.source=max;
				maxBut.toolTip="Maximiza";
			//duração do efeito
			ambos.duration=100;
			//adicionamos como filhos do nosso paralel
			ambos.addChild(dim);
			ambos.addChild(moveP);
			//iniciamos o efeito
			ambos.play();
			}
			else {
			//usado para maximizar o painel caso o nosso painel esteja na
			//posição original:
			//altura em que o maximizado=false; e minimizado=false;
			//definimos como tamanho a totalidade da area da nossa
			//aplicação
			dim.heightTo=Application.application.height;
			dim.widthTo=Application.application.width;
			//movemos para a posição x=y=0
			moveP.xTo=0;
			moveP.yTo=0;
			//o seu estado passa a ser maximizado
			maximizado=true;
			maxBut.toolTip="Restaura";
			//mudamos para a imagem maximizada
			maxBut.source=rest;
			//duração do efeito
			ambos.duration=100;
			//childs
			ambos.addChild(dim);
			ambos.addChild(moveP);
			//play
			ambos.play();
			}
		}
	}
}
```

Como podem ver, o processo é muito lógico e bem simples de executar, neste momento ao adicionares o vosso customPanel a uma aplicação, ele ficará exactamente igual a um painel normal do flex, mas a magia vem na hora da execução da vossa aplicação...

Antes da execução da aplicação (Design View do Flex Builder / Eclipse)

Após a execução:

Para que isto funcione, ncessitam de ter as 3 imagens:

restBut.png maxBut.png minBut.png

na mesma pasta (**com/msdevstudio**) que o nosso **customPanel.as** (ficheiro em cima).

E pronto, têm um painel pronto a ser usado nas vossas aplicações que suporta drag & drop, Maximizar, Minimizar e restaurar... muito simples, pratico e eficiente.

Se conseguiram entender o processo posso afirmar que estão prontos para se aventurar de armas e bagagens pelo mundo do action script e suas classes.

6. Efeitos e Filtros

Como vimos anteriormente foram utulizados alguns efeitos (resize, move e parallel) que fizerem com que o nosso painel aumenta-se de tamanho e se coloca-se na devida posição indicada pelo move.

O AS3 do Flash traz algumas funções de animação, mas são muito limitadas, por isso vou falar das funções de animação do AS3 no Flex, bem como alguns efeitos e também como criar a nossa propria função de animação e criar efeitos paralelos e sequenciais.

Começando pelos mais simples, um simples efeito de movimento feito programaticamente sem recurso aos efeitos nativos do flex..

6.1. Move - Movimento

Pretende-se movimentar uma imagem no 'eixo dos xx' desde x=0 até x=100, para isso facilmente conseguria-mos perceber o seguinte código feito com recurso a um timer:

```
//variavel timer, que define o seu tempo de execução em 10ms
private var timed:Timer = new Timer(10);

//função para iniciar o timer
public function timedMovement():void {
      imagem.x=0;
      timed.addEventListener(TimerEvent.TIMER, timedMove, false, 0, true);
      timed.start();
      }

//função chamada pelo timer a cada 10 milisegundos
private function timedMove(evt:TimerEvent):void {
//move a imagem 4 em 4 px, verificando antes de mover a imagem se o proximo
//movimento ficará acima do nosso maximo (100) imagem.x+4>100 e se for o caso é
//sinal que a imagem já se encontra no final do movimento pretendido, parando o
//timer.
       if(((imagem.x+4)+imagem.width)<100) imagem.x+=4;
       else {
       timed.stop();
       timed.removeEventListener(TimerEvent.TIMER, timedMove);
 }
```

Este simples movimento efectua uma transição do x=0 até ao x=100 de 4 em 4 pixeis, como exemplifica a imagem em baixo:

imagem.x=0 imagem.x=100

Este é o basico de qualuqer movimento fundamentado pelas leis damatemática, e para eveitar estes calulos todos foram feitas varias funções de animação e incluidas no package mx.effects que pode ser acedido no flex.

Tendo o objectivo de simplificar, este package também nos propões algumas soluções simples que necessitariam de grandes conhecimentos matetáticos e acima de tudo a necessidade de saber implementar esses conhecimentos no AS3... mas como isso é tudo ultrapassado pelo package effects podemos provar a sua simplicidade fazendo o mesmo movimento que em cima criamos, mas desta feita, nativamente com a função Move do AS3;

Vejam o seguinte código que fará a mesma coisa que o exemplo em cima:

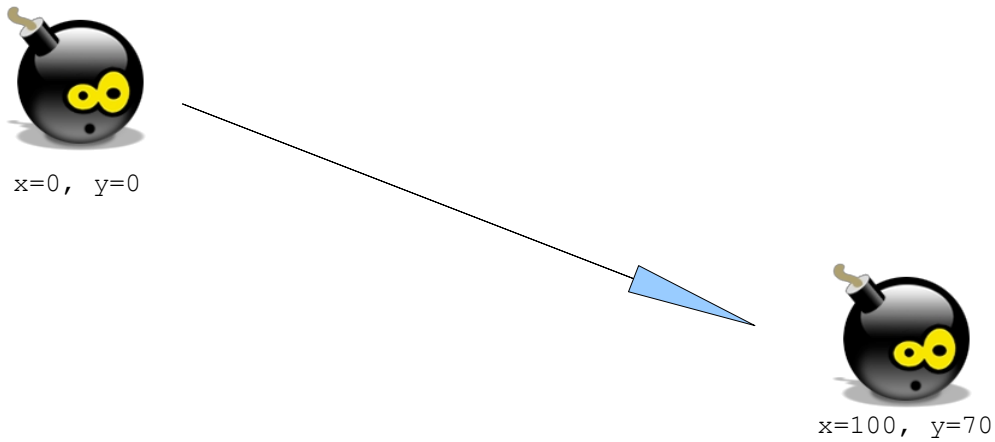
```
var move:Move = new Move();
imagem.x=0; //colocamos a nossa imagem no x=0;
move.duration=2000; //duração do movimento em milisegundos 200 =~ 2s
move.target=imagem; //definimos o target do nosso movimento
move.xTo=100; //Indicamos que o x deve-se mover até ao x=100;
move.play(); //iniciamos o nosso movimento.
```

Este função não necessita de qualquer eventListener e faz exactamente o que criamos em anteriormente. Coloquei estes 2 exemplos para conseguirem entender que todas as funções que vamos falar mais à frente são possiveis de fazer com calculos matematicos.

A partir desta funçâo, falcilmente poderiamos criar uma outra que move no eixo dos xx e também no eixo dos yy, vejam:

```
public function nativeMultiMove():void {
                    var move:Move = new Move();
                    imagem.x=0;
                    imagem.y=0;
                    move.duration=2000;
                    move.target=imagem;
                    move.xTo=100;
                    move.yTo=70;
                    move.play();
            }
```

Esta função criará um movimento como a imagem em baixo demonstra:

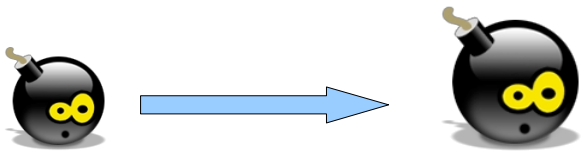


6.2. Resize - Redimensionamento

O segundo efeito apresentado anteriormente no exemplo do painel foi o resize, que é mais um dos efeitos nativos nas classes effects do AS3 no Flex, que pode ser facilmente feito pela segunte função.

```
public function nativeSize():void {
                    //Função que cada vez que for executada aumenta em 20px o
                    //tamanho da nossa imagem.
                    var size:Resize=new Resize;
                    imagem.x=0;
                    size.target=imagem;
                    size.heightTo=imagem.height+20; //aumenta a altura em 20px
                    size.widthTo=imagem.width+20 //aumenta o comprimento em 20px
                    size.duration=500; //duração
                    size.play(); //inicio
            }
```

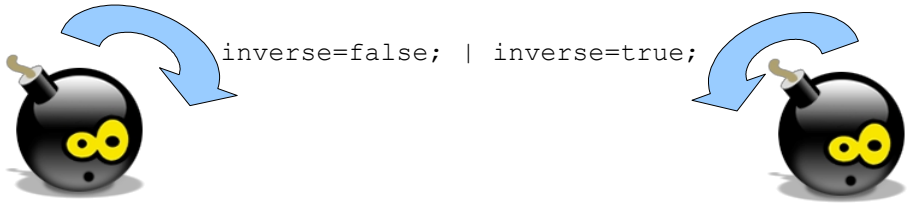
Ao chamar esta função, a imagem será aumentada na largura e altura em 20px como demonstrada pela imagem:



6.3. Rotate - Rodar

Temos ainda outros efeitos, como o rotate que nos permite rodar a imagem em determinados graus como mostra a função:

```
public function nativeRotate(inverse:Boolean=false):void {
                    var rot:Rotate = new Rotate;
                    //colocamos a imagem numa posição em que seja possivel rodar
                    // isto não é necessário mas é por uma questão que se a imagem
                    // estiver com x=0 e y=0 o seu movimento será parcialmente
                    // ocultado
                    imagem.x= 100;
                    imagem.y= 100;
                    rot.angleTo=+360; // definir o angulo, 360= 1 volta completa
                    rot.duration=1000; // duração.
                    rot.play([imagem],inverse);
                    //nesta linha vemos umas novidades, em vez de definir o
                    //rot.target=imagem, posso defini-la quando se chama a função
                    //indicando um ou mais target's num formato de array, e uso
                    //também o inverse, que permite executar o mesmo efeito mas ao
                    //contrario como exemplifica as imagens em baixo.
            }
```



6.4. Efeitos; Exemplos

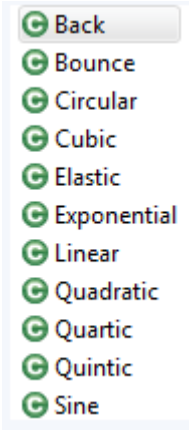
Com estas funções e a com a ajuda de um eventListner podemos fazer um efeito bem agradavel de uma imagem a saltar infinitamente... vejam o código:

```
private var moveDir:String="Up";
private var customMove:Move = new Move();

public function customJump():void {
	moveDir="Up"; //usado para saber o sentido da animação (up/down)
	imagem.x=100;
	imagem.y=100;
	customMove.target=imagem;
	customMove.yTo=imagem.y-50; //move 50px para cima
	customMove.addEventListener(EffectEvent.EFFECT_END, doJump,false,0,true);
	//no final de efectuar o movimento chama a função doJump
	customMove.play(); //inicia
	}
private function doJump(evt:EffectEvent):void {
	if(moveDir=="Up") { //se estiver a executar para cima
	customMove.yTo=imagem.y+50; //anda 50px para baixo
	customMove.play(); //inicia o novo moviemnto
	moveDir="Down"; //que esta a mover para baixo
	}
	else
	{ //sinal que terminou de mover para baixo
	customMove.yTo=imagem.y-50; //movemos de novo para cima
	customMove.play(); //inicia o efeito de novo
	moveDir="Up";
	}
}
private function stopJump():void {
customMove.addEventListener(EffectEvent.EFFECT_END, doJump);
//retira o eventListener que fará com que a função de movimento e a repetição
dos move seja feita.
}
```

Esta função (customJump) fará com que a imagem esteja a fazer um efeito de salto repetivamente, o que não passa de movimentos repetidos +-5px na vertical.
Muito simples e com um aspecto bem agradável.

Dentro do package mx.effects encontram a class easing, que nada mais nada menos nos disponibiliza algumas funções de animação que introduzem efeitos durante a animação, vulgarmente chamadas easing Functions que podem ser adicionadas a qualquer elemento explicado em cima.



Se escreverem mx.effects.easing. Aparecer-vos-á uma lista das funções easing, como na imagem ao lado, funções estas que podem ser introduzidas nos nossos efeits, utilizando o parametro .easingFunction como vou passar a explicar a seguir num exemplo que vai simular a queda de uma imagem e fazer com que ela salte no «chão» ao tocar nele.

Utilizando o nosso move, faremos o seguinte:

```
public function nativeDropBounce():void {
                    imagem.y=0;//inicio do move y=0; x=100;
                    imagem.x=100;

                    var move:Move = new Move;
                    move.target=imagem;
                    move.duration=1500;
                    move.yTo=150; //definição do y final, «chão»
                    //função de animação
                    move.easingFunction=Bounce.easeOut;
                    move.play();

                    //Utilização das classes easing / funções de animação
                    //Estão disponiveis muitos efeitos acedidos pelo package
                    //mx.effects.easing. bastando escrever mx.effects.easing. e
                    //escolher da lista o efeito, algums exemplos:
                    /*Bounce.easeIn / easeInOut / easeOut
                     Elastic.easeIn / easeInOut / easeOut
                     Circular.easeIn / easeInOut / easeOut
                     Back.easeIn / easeInOut / easeOut
                     etc...
                    */
            }
```

Se executarem esta função terão um efeito de queda com se fosse uma bola a cair e a saltar até parar. Este Bounce é uma das funções Easing mostradas em cima, e pode ser substituido por qualquer uma das outras na lista, testem e vejam os outros efeitos.

6.5. Filtros; Fade & Parallel

Existem alguns outros efeitos, mais conhecidos como filtros, que alguns deles falaremos mais à frente noutra forma de uso, neste momento vamos falar de 3 efeitos muito usados, o Fade, Blur e Glow que nos permite adicionar um pouco mais de criatividade às nossas imagens / botões.

Vamos começar pelo Fade, já aplicado a um movimento da imagem, o objectivo é que a imagem comece invisivel e ao longo do movimento venha aparecendo até ficar totalmente visivel no final do efeito. Para que isto aconteca ao mesmo tempos temos que usar o Parallel que nos permite executar 2 efeitos ao mesmo tempo no mesmo objecto, vejam o seguinte código:

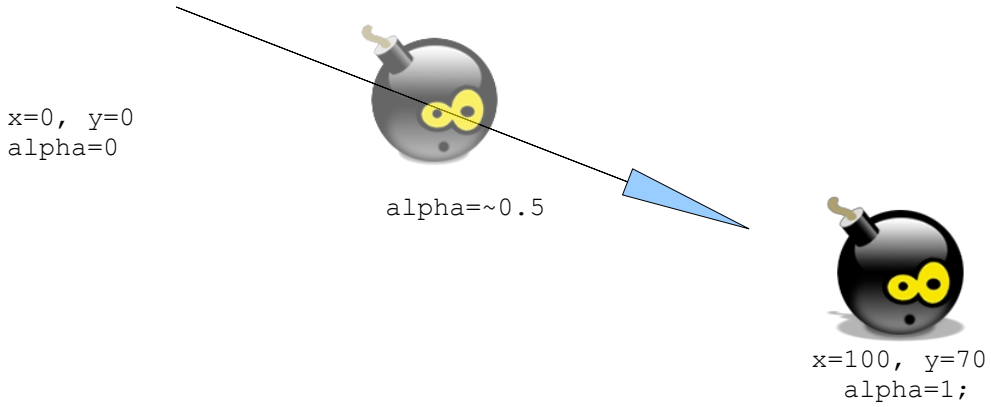
```
public function nativeFadeMove():void {
                    imagem.x=0;
                    imagem.y=0;
                    //o parallel como  já viram no exemplo do painel é uma
                    //instancia de efeitos que nos permite agrupar os efeitos que
                    //quisermos (addChild) e executar todos ao mesmo tempo.
                    var par:Parallel = new Parallel;
                    //ao adicionar o target do paralel como a imagem, em cada
                    //efeito nao necessitamos de o fazer.
                    par.target=imagem;
                    par.duration=1500;

                    //nosso movimento
                    var move:Move = new Move;
                    move.xTo=100;
                    move.yTo=70;
                    par.addChild(move); //adicionamos ao parallel
```

```
            //nosos filtro/efeito fade
            var fade:Fade = new Fade;
            fade.alphaFrom=0; //alpha inicial 0=invisivel, 1=visivel
            fade.alphaTo=1;
            par.addChild(fade); //adicionamos ao paralel

            par.play(); //iniciamos o nosso parallel.
        }
```

E finalmente está pronta a nossa animação que fará um movimento de x=0, y=0 até x=100, y=70, e fará também a transformação progressiva do alpha da imagem de 0 até 1, algo como a imagem em baixo ilustra.



6.6. Filtros; Glow

Agora vamos a um exemplo de Glow; A base deste efeito é um um contorno feito em volta da nossa imagem com alguns pixeis, preenchido de cor em que essa mesma cor é sujeita a um efeito de fade, que nos dará a impressao de estar a desaparecer essa mesma cor fanzendo com que a nossa imagem apresente um resultado bme interessante...

vejam o código:

```
public function nativeGlow():void {
            imagem.x=50;
            imagem.y=50;
            var glow:Glow = new Glow;
            glow.target=imagem;
            glow.alphaFrom=0; //o efeito inicia com alpha a 0
            glow.alphaTo=1; //até 1
            glow.blurXTo=30; //distancia/blur/XX
            glow.blurYTo=30; //distancia/blur/YY
            glow.color=0x43ADF0; //Cor do glow
            glow.play();
        }
```

Este código produziria o efeito seguinda na imagem:

Sem Glow

Com Glow

O efeito glow aparecerá gradualmente visto ser um efeito processado gradualmente.

6.6. Filtros; Blur

Outro efeito parecido é o Blur, que faz com que a nossa imagem fora um efeito de desfocagem como o exemplo a seguir mostrará:

```
public function nativeBlur():void {
                    imagem.x=50;
                    imagem.y=50;
                    var blur:Blur = new Blur;
                    blur.target=imagem;
                    blur.blurXTo=10; //distancia/distorção/XX
                    blur.blurYTo=10; //distancia/distorção/XX
                    blur.duration=100; //duração
                    blur.play();
            }
```

Que produziria o efeito seguinte:

Sem Blur

Com Blur

Estes são os filtros mais usados, no entant ficam outros para poderem pesquisar, tais como o Dissolve, Iris ou Zoom.

Para eliminarem os efeitos glow e blur da imagem façam:

imagem.filters=[];

Para restaurar a visibilidade retirada pela alpha:

imagem.alpha=1;

6.7. Filtros; Sequence

Existe também uma instancia Sequence, que nos permite fazer uma sequencia de efeitos e executa-los pela devida ordem, ao contrario do parallel, os efeitos são executados individualmente e por ordem (a mesma ordem de adição com o addChild) que nos permite fazer sequencias de efeitos. O que vou demonstrar é fazer um movimento de queda de uma imagem (y=0 -> y=100), de seguida fazer um Glow e depois um Fade. Vejam o seguinte código.

```
public function multipleEffects():void {

                    var seq:Sequence = new Sequence;
                    var move:Move = new Move;
                    var glow:Glow = new Glow;
                    var fade:Fade = new Fade;

                    imagem.y=0
                    imagem.x=50;

                    move.duration=1500;
                    move.yTo=(this.height-(originalSizeH));
                    //função de animação
                    move.easingFunction=Bounce.easeOut;

                    seq.addChild(move); //adicionamos à sequencia, ordem : 1

                    //depois de mover vamos fazer o glow
                    glow.alphaFrom=0;
                    glow.alphaTo=1;
                    glow.blurXTo=30;
                    glow.blurYTo=30;
                    glow.color=0x43ADF0;
                    glow.duration=1500;

                    seq.addChild(glow); //adicionamos à sequencia, ordem : 2

                    //finalmente o nosso fade;
                    fade.alphaFrom=1;
                    fade.alphaTo=0;
                    fade.duration=1500;

                    seq.addChild(fade); //adicionamos à sequencia, ordem : 3

                    seq.target=imagem; //definimos como alvo da seq a nossa imagem
                    seq.play(); //iniciamos a sequencia.

            }
```

Testem o exemplo e verão os 3 efeitos (Move, Glow e Fade) serem executados pela ordem que foram adicionados à sequencia, o efeito seguinte apenas se inicia quando o anterior estiver terminado.

6.7. Filtros & Efeitos; Exemplos

Já falamos de quase todos os efeitos principais e das suas implicações e funcionamento, vamos agora ver como a matemática (às vezes dificil de entender) nos pode ajudar a fazer um movimento circular infinito na nossa imagem, vejam o código:

```
private var angulo:int=0;  //guarda o angulo actual da posição da imagem
private var raio:int=50;  //raio do circulo que a imagem vai fazer
private var centroX:int;  //centro do circulo XX
private var centroY:int;  //centro do circulo YY
private var isMoving:Boolean=false; //definido para saber se existe movimento.
private var customMove:Move = new Move;  //variavel do nosso efeito


public function nativeCircularMove():void {
            var radians:Number= (angulo/180) * Math.PI; //angulo en rad.
            imagem.x=0;
            imagem.y=0;
            customMove.target=imagem;
            customMove.duration=5; //Movimento rápido, 5ms

            //faz o movimento da imagem de acordo com o circulo imaginário
            //o circulo é definido sempre com o centro fixo (centroX/Y) e
            //variando de acordo com os cosenos (XX) e senos (YY) do grau de
            //movimento, multiplicado pelo raio para que fique sempre à mesma
            //distancia do centro.
            customMove.xTo = centroX + Math.cos(radians) * raio;
            customMove.yTo = centroY + Math.sin(radians) * raio;

            //quando termina o movimento, chama o moveCircNext que se
            //encarregará de efectuar o proximo movimento que tornará este move
            //num ciclo infinito até o eventListner ser retirado.
            customMove.addEventListener(EffectEvent.EFFECT_END, moveCircNext);

            customMove.play(); //inicia o movimento.
            isMoving=true; //declara o movimento como iniciado.
      }


private function moveCircNext(evt:EffectEvent):void{
            angulo-=5; //variação que o efeito fará, vai mudar 5 graus de angulo
            angulo%= 360; //sem esquecer a referencia do circulo, 360 graus.

            //transformação en radianos.
            var radians:Number= (angulo/180) * Math.PI;

            //de novo o mesmo movimento.
            customMove.xTo = centroX + Math.cos(radians) * raio;
            customMove.yTo = centroY + Math.sin(radians) * raio;
            customMove.play();

            }
```

Este movimento será infinito já que o customMove têm um eventListner que fará que seja executado infinitamente, para o terminar, fazemos a função seguinte:

```
private function stopCircMove():void {
      customMove.removeEventListener(EffectEvent.EFFECT_END, moveCircNext);
      isMoving=false;
      }
```

E assim o nosso efeito circular será terminado, testem o código e vejam o poder da matematica aliado à programação..

Esta parte é a versão limitada da segunda parte do tutorial, esta segunda parte na sua totalidade engloba 59 paginas, falando desde a aplicação de efeitos e filtros num explorador, utilização e exploração da API Draw do AS3, Trabalho e exploração com documentos XML usando o E4X, chamadas a ficheiros externos e API's, Objectos remotos e ligações directas do AS3 a backend's usando também o protocolo AMF (incluido amfphp).

Na versão completa, todos estes códigos fontes serão fornecidos com o tutorial e estarão devidamente identificados e comentados.

A versão completa pode por enquanto não será divulgada gratuitamente pelo simples facto de existir uma necessidade de avaliar o mercado para uma possivel revisão e publicação editorial.

Quem estiver interessado em obter a versão completa, ela será disponibilizada a quem fizer uma doação ao blog. Esta doação terá um valor minimo a definir no blog.

Alguns dos motivos que alteraram a minha politica "gratuita" de divulgação deste tutorial foi em grande parte o enorme tempo dispendido na escrita deste mesmo tutorial, a actual necessidade de mudança de servidor do blog (já foi mudado), o qual actualmente recebe 25.000 visitas mensais gerando um trafego médio de 30 GB, muito superior ao limite que tinha disponivel o qual tive que suportar monetariamente antes da mudança de servidor. Por final como irão verificar o montante da doação não é de todo elevado, pelo que além de estarem a ajudar o blog recebem também a versão completa do tutorial e estarão na lista de prioridades de distribuição de conteudos directos e gratuitos do blog.

Além de terem a versão completa ficam com a certeza que me incentivarão a continuar a escrever e ajudar toda a comunidade flex & as3.

Futuramente talvez possa estar prevista a distribuição gradual e gratuita do resto do tutorial, mas não tão brevemente.

No entanto posso afirmar que serão distribuidas gratuitamente 5 cópias da versão completa num sorteio que será colocado no blog, por isso estejam atentos.

Por agora é tudo, qualquer critica, duvida ou sugestão:
http://www.msdevstudio.com
http://forum.msdevstudio.com
admin@msdevstudio.com
Até breve.
Mário Santos.