Prototype

Para entender o que é prototype, é preciso conhecer um pouco de programação orientada à objetos. O prototype faz parte do mecanismo de criação de classes do ActionScript. Mas o que é uma classe?

A palavra classe, em linguagens de programação orientadas a objetos, tem o sentido de TIPO. Uma classe é um tipo de um objeto. Por exemplo, uma sopa seria um objeto da classe Sopa. Podemos ter várias sopas, que compartilham as características gerais de uma sopa, porém, cada uma delas teria suas propriedades diferentes (sopa de frango, sopa de legumes, sopa de designer ao molho MX 2004 etc).

Tudo no Flash é representado por uma classe. Movieclips são objetos da classe MovieClip. Campos de texto são objetos da classe TextField. Essas classes definem o compartamento padrão desses objetos. Por exemplo, o método play() está definido na classe MovieClip. Desta forma, todos os movieclips, mesmo tendo propriedades diferentes (altura, tamanho, forma etc) são capazes de executar o método play(). Mas o que são métodos? Estando associadas à um objeto, funções e variáveis passam a se chamar métodos e propriedades, respectivamente. Para que uma função se torne parte de uma classe, é preciso anexá-la ao objeto prototype. É por isso que muitas "funções" são iniciadas da seguinte forma:

```
MovieClip.prototype.funcao = function() {
```

O código da função estaria abaixo desta linha de código. Aqui, estamos anexando uma função ao objeto prototype da classe MovieClip. Sim, não pense que MovieClip deve ser substituído por um nome de instância de um movieclip qualquer. O que esta linha de código faz é simplesmente iniciar a declaração de uma função, que se tornará um método da classe MovieClip. E sendo um método desta classe, poderá ser executado diretamente a partir de qualquer movieclip.

Julgo interessante que você aprenda também que o termo prototype, quando usado para generalizar estes métodos que estendem classes, está errado. Normalmente chamamos "prototypes" de métodos, e especificamos as classess às quais eles pertencem. Porque podemos anexar não apenas funções mas objetos e propriedades ao objeto prototype de uma classe.

No ActionScript, uma classe é simplesmente uma função usada para criar objetos. Mas o que são objetos? Eu costumo dizer que arrays e objetos são como arroz e feijão. Eu pelo menos, não vivo sem eles. Objetos são como arrays, porém, ao invés de usar números para definir elementos, podemos usar qualquer nome. Quando criamos um objeto Date, por exemplo, com o new Date(), estamos na verdade criando um objeto simples, que tem acesso direto à algumas funções (métodos) que estão associados à classes Date.

Existem duas formas de se criar um objeto simples:

```
-------------------------------------------------------
umObjeto = new Object();
umObjeto.propriedade1 = "valor1";
umObjeto.propriedade2 = "valor2";
-------------------------------------------------------

-------------------------------------------------------
umObjeto = {
propriedade1:"valor1",
propriedade2:"valor2"
};
-------------------------------------------------------
```

Eu acho muito mais conveniente o segundo método, mesmo embora o primeiro ainda seja aplicável em alguns casos. Agora temos um objeto chamado umObjeto, e este contém duas propriedades: propriedade1 e propriedade2. Para acessar os valores dessas propriedades, utilizaríamos o seguinte:

```
-------------------------------------------------------
trace(umObjeto.propriedade1); // exibe "valor1"
trace(umObjeto.propriedade2); // exibe "valor2"
-------------------------------------------------------
```

Objetos podem ser úteis para agrupar informações co-relacionadas. Como já disse, um objeto é uma espécie de array sofisticada. Ao invés de user índices (números) para identificar os dados, usamos nomes personalizados. Agora imagine que em um projeto, você tenha que criar vários objetos iguais, mas com dados diferentes:

```
-------------------------------------------------------
pessoa1 = {
nome:"Jonas Galvez",
email:"jonas[at]jonasgalvez.com"
};
pessoa2 = {
nome:"Neto Leal",
email:"neto[at]netoleal.com.br"
};
pessoa3 = {
nome:"Anderson Arboleya",
email:"anderson[at]digitalmotion.com.br"
};
-------------------------------------------------------
```

Com este exemplo já é possível observar a utilidade dos objetos para encapsular dados. Mas criar vários objetos deste jeito seria uma tarefa realmente chata, não?

Neste caso, você pode criar uma classe para lhe ajudar a definir estes objetos. Uma classe, no ActionScript, nada mais é que uma simples função, que é usada para definir um objeto através do operador new. Veja o exemplo:

```
------------------------------------------------------------
function Pessoa(nome, email) {
this.nome = nome;
this.email = email;
};
------------------------------------------------------------
```

Agora definimos a classe Pessoa, que irá definir um objeto com as propriedades nome e email. Utilizando esta classe, podemos criar objetos mais facilmente, veja:

```
------------------------------------------------------------
pessoa1 = new Pessoa("Jonas Galvez", "jonas[at]jonasgalvez.com");
pessoa2 = new Pessoa("Neto Leal", "neto[at]netoleal.com.br");
pessoa3 = new Pessoa("Anderson Arboleya", "anderson[at]digitalmotion.com.br");
------------------------------------------------------------
```

O operador new é o responsável pela mágica. Ele faz com que a função retorne um objeto automáticamente, sendo que as propriedades definidas dentro da função (variáveis precedidas de "this") se tornam propriedades do objeto retornado). Este operador evita que tenhamos que retornar um objeto explicitamente:

```
------------------------------------------------------------
function Pessoa(nome, email) {
var objeto = new Object();
objeto.nome = nome;
objeto.email = email;
return objeto;
};
------------------------------------------------------------
```

Um objeto pode (e normalmente deve) executar certas tarefas, no caso funções. Uma função que é subordinada à um objeto é chamada método. Podemos adicionar um MÉTODO à classe Pessoa, para mostrar o nome na janela Output, veja:

```
------------------------------------------------------------
function Pessoa(nome, email) {
this.nome = nome;
this.email = email;
this.mostrarNome = function() {
trace(this.nome);
};
```

```
};
-------------------------------------------------------
```

Note que uma função (método), neste caso, é definido da mesma forma que uma variável (ou seja, a declaração function é precedida pelo operador de atribuição "="). Ok, agora já temos uma classe mais sofisticada. Além de definir um objeto com as propriedades nome e email, ela também dá ao objeto um método chamado mostrarNome, que exibe o conteúdo da propriedade nome na janela output. Veja um exemplo:

```
-------------------------------------------------------
pessoa4 = new Pessoa("Marcelo Frias", "marcelo[at]digitalmotion.com.br");
pessoa4.mostrarNome();
-------------------------------------------------------
```

A segunda linha do script acima irá exibir o texto "Marcelo Frias" na janela Output. Agora, pare para pensar. Todo objeto criado através da classe Pessoa receberá uma cópia individual do método mostrarNome. Mas na verdade, o método mostrarNome é único, e uma única cópia dele poderia ser acessada por todas as instâncias da classe, não? É aí que o prototype entra. Toda função (uma função também se comporta como um objeto) tem uma propriedade chamada prototype. Essa propriedade está acessível à todos objetos, o que os permite executar um método sem ter que uma cópia individual do mesmo. Se você anexar o método mostrarNome ao objeto prototype da classe Pessoa (o objeto prototype da função Pessoa), todas as instâncias desta classe terão acesso à este método. Então:

```
-------------------------------------------------------
function Pessoa(nome, email) {
this.nome = nome;
this.email = email;
};
Pessoa.prototype.mostrarNome = function() {
trace(this.nome);
};
-------------------------------------------------------
```

Com isso, temos um aumento de performance no script. Se quisermos adicionar mais métodos para a classe Pessoa posteriormente, basta anexarmos as função ao objeto prototype da classe. É o que ocorre com a classe MovieClip. Podemos adicionar ou substituir métodos da classe MovieClip, de acordo com nossas necessidades.

```
-------------------------------------------------------
MovieClip.prototype.mostrarX = function() {

trace(this._x); // mostra a posição x do movieclip
```

```
};
```

O método mostrarX estará disponível para todos movieclips, sendo que você poderia executá-lo da seguinte forma:

```
qualquerMovieclip.mostrarX();
```

Espero que a coisa tenha ficado clara. No Flash MX 2004, esse tipo de extensão não é mais recomendado. Agora, o ideal é criar libraries de classes externas, que estendem a funcionalidade de uma classe específica. Isso melhora bastante a organização do código. Por exemplo, antigamente eu tinha vários métodos para manipulação de string (para tarefas como: remover espaços, adicionar letras maiúsculas no começo das palavras etc). Estes métodos estavam todos definidos em arquivos externos, da seguinte forma:

```
String.prototype.trim = function() {
// ....
}
```

Agora, com o MX 2004, eu posso definir uma classe, SuperString, para definir métodos adicionais para a classe String. Posso inclusive definir a classe em uma library externa (um package), no estilo do Java.

```
class com.jonasgalvez.SuperString extends String {

public function SuperString(str) {
super(str);
}

public function trim():String {
for(var i = 0; this.charCodeAt(i) < 33; i++);
for(var j = this.length-1; this.charCodeAt(j) < 33; j--);
return this.substring(i, j+1);
}

}
```

Depois, posso carregar essa library da seguinte forma:

```
import com.jonasgalvez.*; // import no estilo Java

var str:SuperString = new SuperString("palavra");
trace(str.trim());
------------------------------------------------------------
```