Sistemas Informáticos
Curso 2006-07

Animaciones de tipos abstractos de datos

Ana Colinas Lorenzo
Alejandro García Asenjo
Aída Mengual Castro

Dirigido por:
Isabel Pita Andreu
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice

Resumen

El objetivo principal de este trabajo es la realización de animaciones dinámicas e interactivas de tipos abstractos de datos mediante el uso de la herramienta de Macromedia Flash. Está dirigido principalmente a los alumnos que estén cursando asignaturas en las que se estudian tipos abstractos de datos.

En el trabajo se han realizado animaciones sobre un algoritmo de evaluación de expresiones postfijas, tres algoritmos de comprobación de cadenas palíndromas utilizando distintas estructuras de datos y por último una animación sobre la inserción y búsqueda de elementos en árboles 2-3-4.

El primer algoritmo completa el trabajo realizado el curso anterior sobre transformación de expresiones en forma infija a forma postfija. Los árboles 2-3-4 se suman a las animaciones de árboles de búsqueda y árboles equilibrados (rojo-negros) realizadas también el curso pasado.

A diferencia de los algoritmos desarrollados el curso anterior, las animaciones son completamente interactivas, permitiendo al usuario introducir los datos, esto se ha conseguido utilizando la herramienta ActionScript.

Adicionalmente se ha desarrollado un tutorial sobre cómo implementar animaciones en Flash utilizando ActionScript. Este manual será de gran utilidad para la continuidad del proyecto.

Summary

The main purpose of this project is to do dynamic and interactive abstract data type animations using Macromedia Flash. It is directed to those who are studying subjects in which abstract data types are being taught.

The animations that have been done are based on evaluating postfix expression algorithms, three algorithms about testing palindromes using different abstract data types and, finally, an animation about inserting and searching elements in 2-3-4 trees.

The first algorithm completes the work done last year about changing infix expressions into postfix expressions. The 2-3-4 trees are added to searching trees and balanced trees (red-black), made last year too.

The difference between the algorithms made last year and the ones done now is that animations are completely interactive allowing the user to insert data; this has been achieved thanks to Action Script.

An additional tutorial about how to implement animations in Flash using Action Script is been made. This manual ensures the continuity of the project.

1. Introducción

Los tipos abstractos de datos (TADs) son un elemento fundamental en la programación actual. Un TAD define un conjunto de valores junto con las operaciones que manejan dichos valores. Son por tanto piezas básicas de la programación modular, que permiten separar de forma estricta la implementación del TAD del uso que se hace de él en las aplicaciones. Para ello al definir un TAD se definen los valores del tipo y las operaciones con los respectivos interfaces. Normalmente se utiliza la técnica de la especificación algebraica para definir el comportamiento del tipo. Un tipo de datos suele permitir varias implementaciones, variando el coste de las operaciones de unas a otras. La implementación concreta seleccionada al utilizar un TAD en una aplicación debe ser transparente, esto es, se debe poder cambiar una implementación por otra sin modificar la aplicación.

El objetivo de este proyecto es realizar animaciones de algunos tipos abstractos de datos para facilitar su estudio a los alumnos. La animación puede referirse al comportamiento de la estructura, a una implementación concreta de ella, o al uso de la estructura en una aplicación. El trabajo continúa el realizado el año pasado por los alumnos R. Carrasco, J. Crespo y A. López en el proyecto de Sistemas Informáticos [1].

La principal aportación de este proyecto es la realización de animaciones dinámicas permitiendo a los usuarios introducir los datos que consideren oportunos para estudiar el comportamiento de las distintas funciones implementadas para los tipos de datos y estudiar el funcionamiento de éstos con todo el nivel de detalle que se quiera.

El proyecto está dirigido principalmente a los alumnos que están cursando asignaturas en las que se estudian tipos abstractos de datos y por ello, las animaciones han sido diseñadas teniendo en cuenta que pueden ser proyectadas en aulas para su visualización siendo éstas así ricas en contenidos visuales y de manejo muy intuitivo. Las animaciones también están pensadas para ser utilizadas por los alumnos para estudiar individualmente los TADs, para ello en cada animación se muestra los pasos detallados que va dando el algoritmo o la función además de la correspondiente animación.

En este proyecto se han realizado las siguientes animaciones:

- Evaluación de una expresión en forma postfija. La animación comprueba si la expresión dada está en forma postfija e informa del error si no lo está.
- Comprobación de si una cadena es o no palíndroma. Para ello se han empleado tres algoritmos distintos que emplean distintas estructuras de datos: dos pilas, una pila y una cola y una cola doble. Para cada algoritmo se ha producido una animación diferente lo que facilita la comparación de los algoritmos.
- Tipo de datos de los árboles 2-3-4, con las operaciones de inserción de nuevos elementos y búsqueda de elementos.

Para cada animación vemos, a la derecha de la pantalla la animación y a la izquierda los pasos del algoritmo que ésta va siguiendo.

Para el desarrollo de las animaciones hemos utilizado la herramienta Macromedia Flash utilizando la interfaz de programación Action Script, para poder ejecutar los algoritmos y dotar a las animaciones de un alto grado de interactividad con el usuario. El uso de ActionScript hace el desarrollo de las animaciones más estructurado y sencillo de comprender.

Todas las animaciones pueden ser visualizadas desde cualquier navegador que disponga del plugin Macromedia Flash Player, disponible en la siguiente dirección web: www.macromedia.org.

Punto de Partida

Este proyecto es la continuación del proyecto *Sistema multiusuario para la gestión universitaria vía Web, incluyendo el desarrollo de un interfaz para el diseño de aplicación de datos visuales* de R. Carrasco, A. J. López, J. J. Crespo [1].

Las animaciones del proyecto anterior tenían un interfaz atractivo pero un grave problema: la falta de interactividad con el usuario.
Esto es debido a que las animaciones estaban realizadas exclusivamente en Flash con interpolaciones de movimiento, sin la utilización de ActionScript.

Las nuevas animaciones deberían tener una apariencia lo mas parecida posible a la del proyecto anterior para mantener un estándar en estas aplicaciones de estudio para los alumnos. Sin embargo debíamos dotar a la herramienta de cierta interactividad con el usuario.

Objetivos y evolución

El proyecto ha tenido dos objetivos que se han desarrollado en paralelo por un lado el desarrollo de aplicaciones para la asignatura de Estructuras de Datos de la Información; y por otro lado el estudio del lenguaje ActionScript y sus posibilidades para resolver los problemas principales detectados el año anterior. Estos problemas son la falta de interactividad con el usuario y el coste de mantener aplicaciones realizadas en base a fotogramas.

En primer lugar el lenguaje ActionScript implica un cambio importante ya que aquí no sólo se programa de “arriba abajo” sino que también hay que considerar la programación en una línea de tiempo, lo que supone programar de “izquierda a derecha”.

El primer paso para resolver el problema de la interactividad consistió en permitir que el usuario pulsara botones para introducir los datos. Este tipo de interactividad es el utilizado en la animación de la evaluación de expresiones postfijas.

El segundo paso para conseguir la interactividad consistió en utilizar cuadros de texto que permiten introducir datos de teclado. Este es el tipo de interactividad utilizado en la comprobación de expresiones palíndromas y en los árboles 234.

Por último, para la implementación de los árboles 2-3-4, se definieron un conjunto de clase, permitiendo así una mejor organización de la información y el desarrollo de aplicaciones más complejas.

2. Animaciones

A continuación pasamos a tratar las animaciones que hemos realizado en este proyecto. Primero haremos una introducción de cada una de ellas y luego pasaremos a explicar su utilización y como hemos realizado su implementación con Flash.

Todas las animaciones son de manejo muy intuitivo y muy ricas en contenidos visuales por lo que son una buena herramienta de estudio para los alumnos y para su posible presentación en clase facilitando así la tarea al profesor. Para conseguir estos objetivos, las animaciones de *Evaluación de expresiones postfijas* y *Comprobación de expresiones palíndromas* disponen de una pantalla dividida en dos partes:

- A la izquierda se puede observar el código del algoritmo y la flecha blanca que indica el paso que se está realizando en cada momento.
- A la derecha se ve la ejecución, dependiendo de la animación de qué se trate aparecerán representadas pilas, colas o ambas, como en el caso de *Comprobación de expresiones palíndromas* con el algoritmo que utiliza una pila y una cola. Cada uno de los caracteres de las expresiones postfijas y palíndromas están representados por una bola de color azul claro. Todos estos elementos son los necesarios para realizar las operaciones requeridas por el algoritmo en cada caso.

La división de la pantalla permite al usuario relacionar el código con la ejecución, permitiéndole así entender mejor el funcionamiento del algoritmo. La pantalla de estas animaciones dispone, además, de dos botones (verdes) que sirven para cambiar la velocidad de los movimientos en caso de que sea necesario ver algún movimiento con algo más de detalle.

La animación de Árboles 2-3-4 varía en cuanto a su aspecto, sin dejar de ser por ello atractiva para el usuario. En este caso hemos optado por ofrecer en una misma pantalla lo necesario para interaccionar con el TAD y la muestra del resultado que proporcionan los algoritmos.

2.1. Evaluación de expresiones en forma postfija

En este proyecto continuamos con el trabajo realizado con las pilas en el proyecto anterior de Sistemas Informáticos *Sistema multiusuario para la gestión universitaria vía Web, incluyendo el desarrollo de un interfaz para el diseño de aplicación de datos visuales* de R. Carrasco, J. Crespo y A. López [1]. Pero además hemos introducido la posibilidad de evaluar las expresiones aritméticas en forma postfija de manera interactiva.

De esta forma, el usuario puede hacerse una idea más clara de cómo funciona la evaluación de este tipo de expresiones por medio de un algoritmo que utiliza el TAD de las pilas. Además, dicha interactividad le permite introducir todos los ejemplos de expresiones postfijas que considere oportunos obteniendo el valor de la expresión como resultado en caso de ser correcta o un mensaje de error en caso de que se trate de una expresión incorrecta.

Para realizar la evaluación de expresiones postfijas hemos utilizado un algoritmo basado en el TAD de las pilas. Una pila es una estructura de datos de tipo LIFO (*Last in, First out*) cuya característica principal es que el acceso a los elementos se realiza en orden inverso al de su almacenamiento. Este TAD tiene como ventaja que el acceso a los datos de la estructura se realiza a través de un solo punto, la cima de la pila, dando como resultado un algoritmo sencillo de implementar y a su vez fácil de comprender.

Se considera que una expresión aritmética construida con los operadores binarios +, -, *, / y constantes numéricas está en forma postfija si, o bien es una constante, o bien consiste en dos expresiones postfijas una tras otra, seguidas por un operador. A la vista de esta definición, una de las ventajas de estas expresiones es que no es necesario recurrir al uso de paréntesis para indicar la precedencia de los operadores. Esto es posible gracias a que las expresiones se evalúan de izquierda a derecha aplicando los operadores en el mismo orden en el que se presentan.

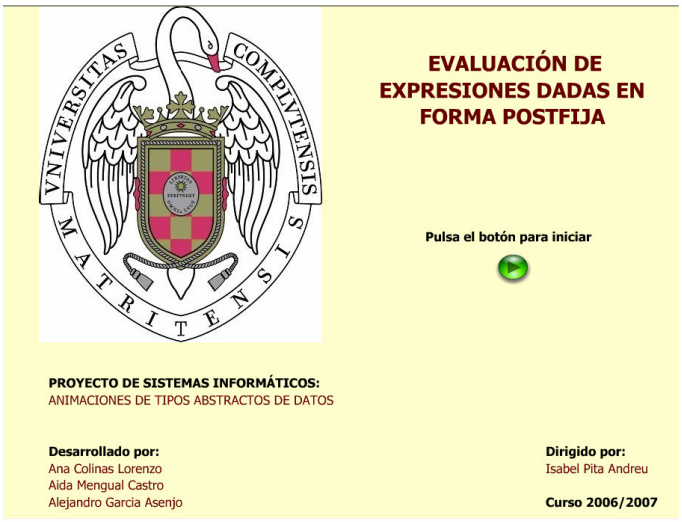
Teniendo en cuenta todo esto, el algoritmo empleado para realizar la animación en Flash es el siguiente: [2]

```
Proc Evaluar (exp: secuencia[Char], r: Nat);
        Var p:Pila[Nat];  x:Char;
        p := pila-vacia();
        reiniciar(exp);
                mientras not fin(exp) hacer
                x := actual(exp); avanza(exp);
                si es-constante?(x) entonces
                        apilar(valor(x), p);
                si no  {operador}
                        op2 := cima(p); desapilar(p);
                        op1 := cima(p); desapilar(p);
                        r := aplicar(x, op1, op2); apilar(r,p);
                fsi
                fmientras
                 r := cima(p); desapilar(p);
        fproc
```

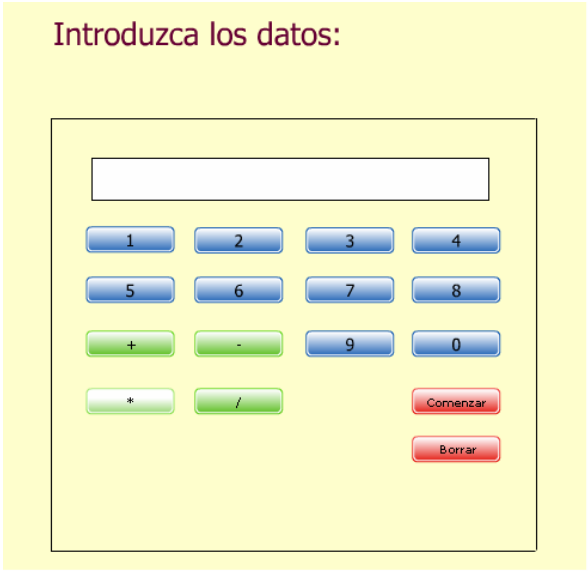
Se parte de una pila vacía en la que se va almacenando el valor de los operandos de la expresión y de los resultados intermedios. Se comienza el proceso con la pila vacía y se recorre de izquierda a derecha la secuencia de caracteres que forman la expresión. Cada vez que se encuentra una constante, se apila su valor. Si por el contrario, aparece un operador, se desapilan la cima y la subcima de la pila, se realiza la operación indicada por el operador y se apila el resultado. Al finalizar el proceso, la pila tendrá un único valor que será el resultado de la evaluación de la expresión postfija.

2.1.1.1. Animación paso a paso

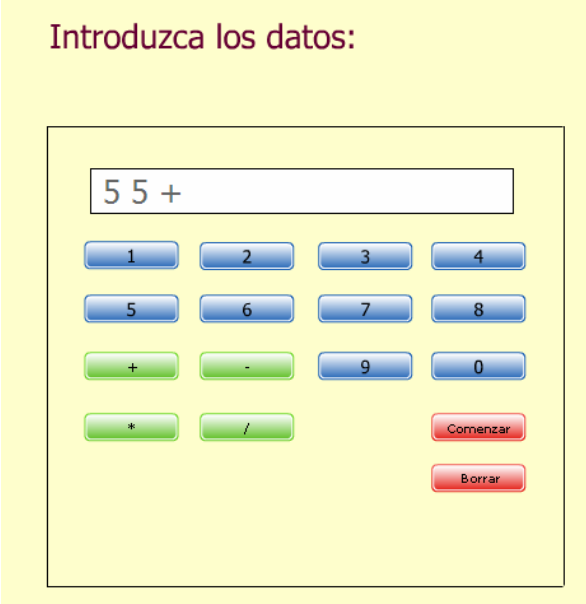
Lo primero que aparece al ejecutar la animación es la portada:



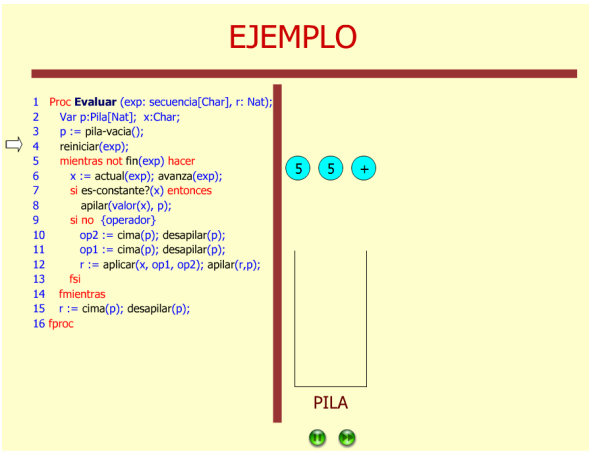
En ella aparece información general sobre el proyecto y se sitúa la animación dentro del mismo por medio del título. Para comenzar hay que pulsar el botón de inicio. A continuación se pide que se introduzcan los datos:



Esta pantalla dispone de los botones necesarios para introducir una expresión aritmética pulsando sobre ellos. Se pueden realizar las operaciones suma, resta, multiplicación y división con operandos enteros de 0 a 9. Además, es posible borrar y volver a empezar de nuevo en caso de que se necesite corregir algo, esto se hace por medio del botón *Borrar*. Una vez que se tiene la expresión deseada se pulsa el botón *Comenzar*. Veamos un ejemplo, la expresión postfija 55+.

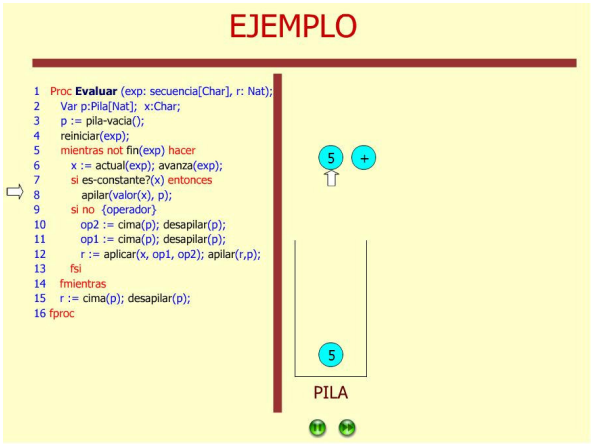


Una vez pulsado el botón *comenzar* aparece la pantalla en la que se ejecuta el algoritmo,

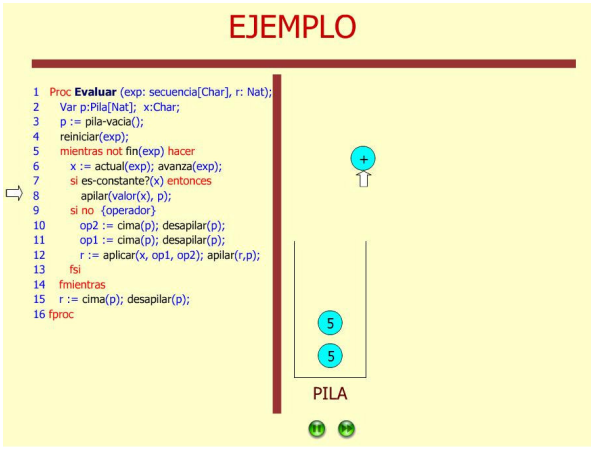


Veamos como se comporta la animación para el ejemplo elegido anteriormente,

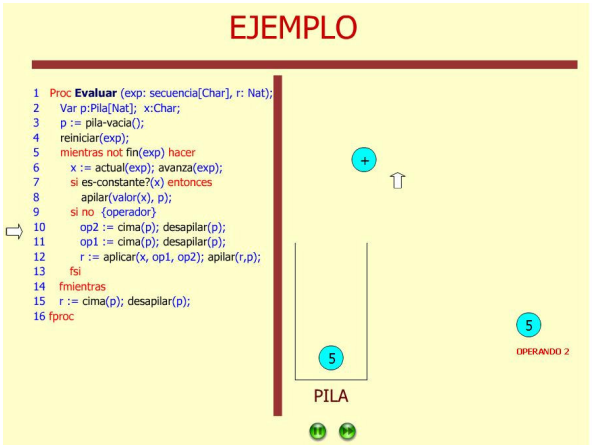
1.Se apila el primer operando



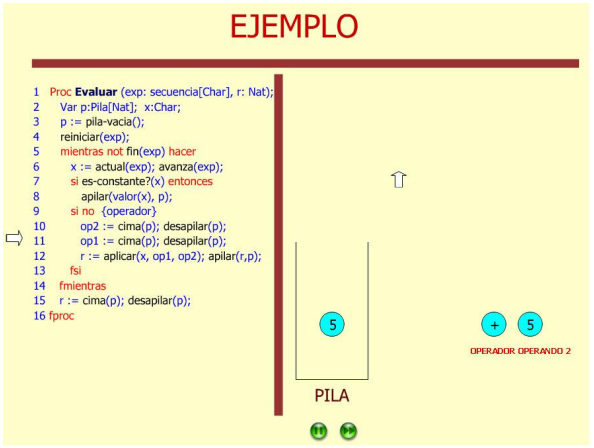
2. Se apila el segundo operando



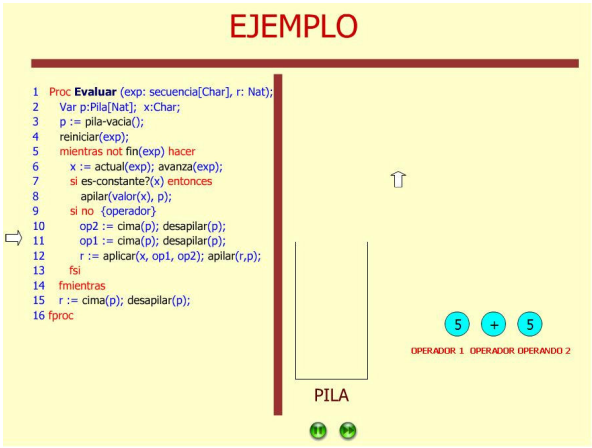
3. Al llegar al operador, se desapila el primer operando



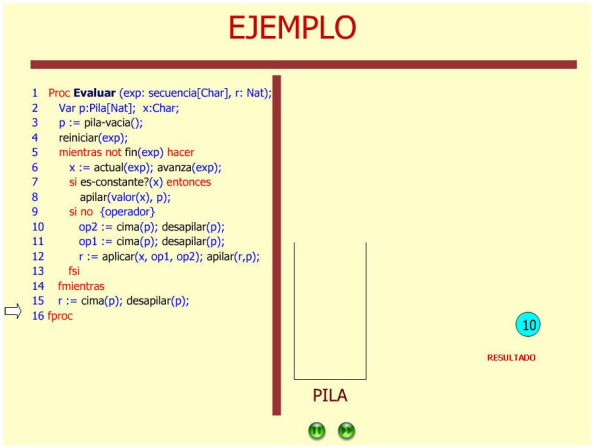
4. Se coloca el operador



5. Se desapila el segundo operando



6. Se obtiene el resultado



En este caso la ejecución termina correctamente devolviendo el valor correspondiente debido a que la expresión elegida está en forma postfija. En caso de que la expresión elegida no fuera postfija la evaluación de la expresión se detiene en el momento en el que se detecta un error y aparece un mensaje advirtiéndolo.

Si se desea realizar otra evaluación es necesario volver a iniciar la animación desde el principio siguiendo los pasos anteriormente señalados.

2.1.1.2. Implementación con Flash

Para realizar esta animación hemos utilizado un único archivo .fla en el que hemos introducido una división en capas para cada uno de los elementos que posteriormente aparecen en la pantalla y así facilitar su localización en caso de necesidad.

La capa principal se llama *acciones* y en ella se han programado en lenguaje Action Script los pasos que realiza el algoritmo, así como los movimientos y la creación de los elementos animados que se observan en la pantalla. Los fotogramas de esta capa contienen la siguiente funcionalidad:

- *Fotograma 1*: en este fotograma hemos incluido una parte fundamental de la animación, aquí definimos todas las variables que posteriormente utilizamos para programar el algoritmo y las funciones que nos sirven para realizar los movimientos de los objetos. Estas funciones se implementan utilizando la clase Tween de Action Script. Esta clase ofrece la posibilidad de programar los movimientos de los objetos sin necesidad de utilizar la línea de tiempo, lo que evita el uso de un número excesivo de fotogramas haciendo que la implementación y su posterior mantenimiento sean más sencillos. En las funciones implementadas utilizando esta clase lo único que hay que señalar es el punto de inicio, el punto donde se desea que finalice el movimiento y el tiempo que se requiere para dicho movimiento. Además, esta clase permite ciertos efectos visuales que pueden resultar atractivos para el usuario, como por ejemplo los cambios de velocidad del objeto en movimiento o el rebote, éste último utilizado para esta animación. En nuestro caso, hemos definido dos funciones que utilizan la clase *Tween* para realizar movimientos verticales, horizontales y diagonales:

```
//Mueve la bola en un sentido (horizontal o vertical)
function tweenBall(easeType, bola,begin,end,time,propiedad,salto) {
    //mueve la bola en sentido horizontal o vertical, con el efecto
    //esaseType desde begin hasta end en un tiempo time
    ballTween = new mx.transitions.Tween(bola,propiedad, easeType, begin, end, time);
    stop();
    //cuando termina el movimiento salta al fotograma salto
    ballTween.onMotionFinished = function() {
            gotoAndPlay(salto);
    };
}

//Mueve la bola en diagonal
function tweenBall2(easeType,
bola,begin1,end1,begin2,end2,time,propiedad1,propiedad2,salto) {
    //mueve la bola en el sentido que indique propiedad1
```

```
    ballTween = new mx.transitions.Tween(bola, propiedad1, easeType, begin1, end1,
time);
    //mueve la bola en el sentido que indique propiedad2
    ballTween = new mx.transitions.Tween(bola, propiedad2, easeType, begin2, end2,
time);
    //la unión de los dos movimientos da como resultado un
    //movimiento diagonal
    stop();
    //cuando termina el movimiento salta al fotograma salto
    ballTween.onMotionFinished = function() {
        gotoAndPlay(salto);
    };
}
```

- *Fotograma 4*: este fotograma se encarga de manejar la pantalla de introducción de datos que aparece a continuación de la portada de la animación. En ella se controlan los botones y se gestiona la introducción de caracteres mediante una variable de texto dinámico. Esta variable está identificada por un nombre a través del cuál accedemos a la propiedad *.text* que contiene la secuencia de caracteres que forman la expresión. Con lo que cada vez que se pulsa un botón en la pantalla, se añade un carácter a la variable de texto dinámico. A su vez, también se crea un objeto de tipo MovieClip (llamado *bolaX,* siendo X el lugar que ocupa el carácter en la expresión) que se utilizará mas adelante en la animación para visualizar cada uno de los caracteres de la expresión. Todos los objetos MovieClip que se crean, así como el valor que representan, se almacenan ordenadamente en dos variables de tipo array, a las que posteriormente accederá el algoritmo para llevar a cabo la evaluación de la expresión postfija. Como muestra, veamos el código del botón número1:

```
btn1.onRelease = function() {
        i = i+1;//contador de caracteres
        v[i] = 1;//array donde se guarda el valor del caracter
        td1.text = td1.text+" 1";//aumentar la variable de texto
        initObjeto = new Object();//inicialización del objeto
        initObjeto._y = -50;
        initObjeto._x = 400+50*i;
        vari="caracter"+i;
        //creación del MovieClip bola que representa el caracter
        w[i]=_root.attachMovie("bola"+i,vari,i,initObjeto);
        w[i].td.text="1"
};
```

- *Fotogramas 5 a 13*: en estos fotogramas se dan los primeros pasos del algoritmo. En el fotograma 5 se realiza el movimiento que hace aparecer las bolas que contienen los caracteres de la expresión y que fueron creadas anteriormente. A continuación, en los siguientes fotogramas, se crean los elementos que aparecen en la pantalla, como por ejemplo las flechas, que indican los pasos que se van dando, y la pila con la que van a interactuar las bolas en el algoritmo. Además, se muestra el funcionamiento del algoritmo.
- *Fotogramas 14 a 44*: esta es la parte principal del algoritmo, el bucle. Es necesario tener en cuenta que en Flash se trabaja en todo momento con la línea de tiempo y esto requiere cambiar la mentalidad a la hora de programar, es decir, es necesario llevar a cabo una programación que

podemos llamar “horizontal” cuando hay que implementar bucles de procesos que se repitan en el tiempo. Este tipo de programación hace que sea necesario volver hacia atrás en la línea de tiempo para poder hacer varias iteraciones del bucle del algoritmo y repetir así todas las comprobaciones y operaciones necesarias. Para realizar estos “saltos” en la línea de tiempo utilizamos principalmente la operación *gotoAndPlay()* que va al fotograma que se le indica como parámetro, ya sea hacia adelante o hacia atrás. Además, las funciones de movimiento llevan incorporado un salto que se realiza al terminar dicho movimiento.
En el fotograma 14 se detecta un error en caso de que al encontrar un operador en la secuencia de caracteres que se está evaluando, la pila este vacía o no haya suficientes operandos. En este caso se “salta” directamente al final (fotograma 46).
Veamos el contenido del fotograma 14 como ejemplo de uso de las funciones de “salto” y detección del error:

```
if (cont<v.length){//si quedan bolas
        //si el siguiente caracter es un operador suma, resta, multiplicación o división
        if (v[cont]=="+" || v[cont]=="-" || v[cont]=="*" || v[cont]=="/"){
                //detección del error: en la pila hay más de 2 bolas
                if(p>=2){
                        gotoAndPlay(21);//saltar al fotograma número 21
                }
                else{
                        gotoAndPlay(46);//saltar al fotograma número 46
                }
        }
}
else {
//no hay más bolas y se acaba el algoritmo, no sin antes comprobar el segundo posible
//motivo de error
        gotoAndPlay(44);//saltar al fotograma 44
}
```

- *Fotogramas 45 a 46*: aquí se hace otra comprobación de error. Puede ocurrir que la expresión postfija que se introdujo al principio no sea correcta por que el número de operandos es excesivo, es decir, sobran operandos y en la pila hay más bolas aparte de la bola resultado. En este caso se salta al fotograma 46 donde se detiene la aplicación y se muestra el error.
- *Fotogramas 47 a 51:* este es el final del algoritmo. Una vez que se ha comprobado que todo ha ido bien y se ha obtenido el resultado de la evaluación de la expresión postfija (que se encuentra en la pila), se procede a desapilar dicho resultado y termina la ejecución.

Otra de las capas que forman la animación es la capa *Botones*. Esta capa se encarga de controlar los botones de color verde que aparecen en la pantalla cuando se está ejecutando el algoritmo. Son los encargados de cambiar la velocidad a la que se mueven los objetos que aparecen en la pantalla. Esta funcionalidad la hemos implementado declarando dos variables de velocidad, que utilizamos a lo largo de todo el proceso y que, en caso de que se pulse alguno de los dos botones, cambian su valor.

La siguiente capa es la capa *editor*. Hay que acudir a esta capa si se quiere hacer alguna modificación de los elementos de la pantalla de introducción de datos. En esta capa se han introducido los botones que forman el panel de caracteres y también se ha incluido la variable de texto dinámico ya mencionada anteriormente. El único fotograma de esta capa que contiene código en Action Script es el fotograma 4, en el que se ha introducido un stop() que sirve para parar la animación y permitir al usuario introducir la expresión que desea evaluar. Cuando el usuario haya introducido la expresión postfija y esté preparado para continuar solo tiene que pulsar el botón *Comenzar* y se reanuda la animación.

A continuación aparece la capa *marcos*. Esta capa abarca únicamente los tres primeros fotogramas y controla la portada de la aplicación. En ella hemos introducido elementos como el texto y las imágenes. Además, hemos programado un efecto visual que resulta atractivo para el usuario. Al pulsar el botón de inicio que conduce a la pantalla de introducción de datos, la portada desaparece atenuando su color. Para ello hemos utilizado la propiedad ._alpha que permite jugar con la trasparencia de los objetos. Hemos implementado un bucle, de forma que en cada iteración el valor de ._alpha disminuye en una cantidad fija:

```
velocidad=8;//velocidad a la que desaparece la pantalla
if (mcUcm._alpha > 15){
        //se modifica la propiedad ._alpha de cada uno de los objetos que
        //aparecen en pantalla.
        //botón de inicio de la portada
        btn1._alpha= btn1._alpha - velocidad;
        //mcUcm es la foto del escudo de la ucm
        mcUcm._alpha= mcUcm._alpha -velocidad;
        //mcTxt1 es la variable de texto título de la animación
        mcTxt1._alpha= mcTxt1._alpha -velocidad;
        //mcTxt2 es la variable de texto instrucciones
        mcTxt2._alpha= mcTxt2._alpha -velocidad;
        //mcTxt3 es la variable de texto título del proyecto
        mcTxt3._alpha= mcTxt3._alpha -velocidad;
        //mcTxt4 es la variable de texto nombres de los autores
        mcTxt4._alpha= mcTxt4._alpha -velocidad;
        //mcTxt5 es la variable de texto directora del proyecto y año
        mcTxt5._alpha= mcTxt5._alpha -5;
        }
        else{ gotoAndPlay(4);//salto al fotograma 4}
```

De esta forma, la impresión que recibe el usuario es que la portada desaparece lentamente.

Por último, la capa *Algoritmo* contiene el objeto que aparece en la pantalla de ejecución del algoritmo y refleja el código del mismo.

2.2. *Comprobación de expresiones palíndromas*

Esta animación refleja como puede resolverse el problema de comprobar si una palabra es palíndroma utilizando diferentes estructuras de datos.
La animación tiene dos objetivos principales:

- Permitir al usuario profundizar su estudio de un TAD determinado, viendo como éste actúa para distintas expresiones de entrada, obteniendo en cada caso un mensaje de éxito o de fracaso dependiendo de si la expresión introducida es o no palíndroma.
- Comparar cómo actúan y los distintos costes, aunque en este caso serán todos ellos lineales, que tienen los distintos TADs en la resolución de un mismo problema, viendo así las ventajas e inconvenientes del uso de unos frente a otros.

Una expresión o frase palíndroma es aquella en la que la sucesión de sus caracteres, omitiendo posibles espacios en blanco y tildes, es la misma recorriéndola de izquierda a derecha que de derecha a izquierda. Así, por ejemplo, la frase: “Anita lava la tina” es palíndroma.

Los TADs empleados para la comprobación han sido los siguientes:

- Dos pilas
- Una pila y una cola
- Una cola doble.

A continuación pasaremos a estudiar en detalle cada uno de ellos.

2.2.1. **Algoritmo utilizando dos pilas**

En esta primera animación disponemos de dos pilas, en la primera de ellas vamos almacenando todos los caracteres no blancos de la expresión, quedando al final en orden inverso al de lectura. A continuación, se transfieren la mitad de ellos a la segunda pila, quedando éstos en el orden original.

Si la expresión tiene un número impar de caracteres, el carácter central ha de ser ignorado, puesto que se considera simétrico de si mismo.
Por último, se comparan las dos pilas, comparando uno a uno los caracteres extraídos de sus cimas, si son iguales todos ellos dos a dos, la frase de entrada es palíndroma y mostraremos un mensaje de éxito, de lo contrario mostraremos un mensaje de fracaso.

El algoritmo empleado para realizar la animación en Flash tiene el siguiente aspecto: [2]

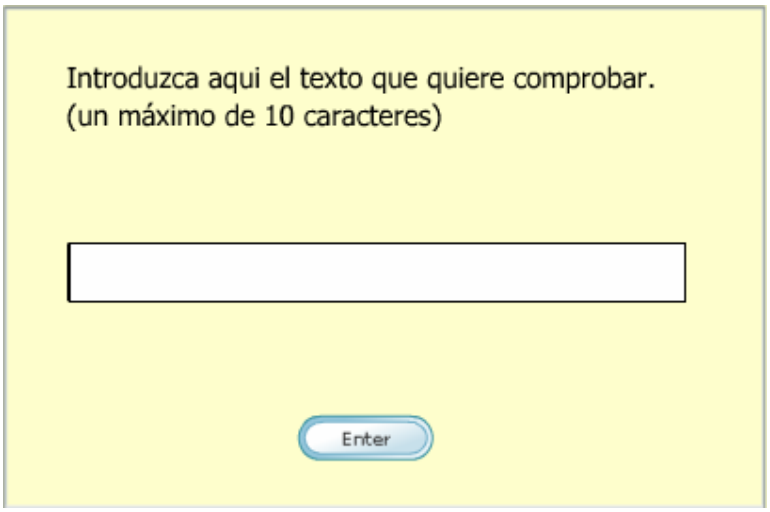
```
Procedimiento palindroma
var p1,p2: pila[car]
  p1:= pila-vacia();
  cont := 0; leer(x);
  mientras x <> fin hacer
        si x<> _ entonces  apilar(x,p);
                           cont:=cont+1;
       fsi
       leer(x);
  fmientras
  p2:= pila-vacia();
  para i=1 hasta (cont div 2) hacer
       x:= cima(p1); desapilar(p1);
       apilar(x,p2);
  fpara
 si es_impar(cont) entonces
    desapilar(p1) ;
 fsi
 b:= cierto
 mientras b and not esPilaVacia(p1) hacer
    b:=(cima(p1) = cima(p2));
    desapilar(p1); desapilar(p2);
 fmientras
 si b entonces imprimir(¡Es palíndroma!)
 si no    imprimir(¡No es palíndroma!);
          anularPila(p1);anularPila(p2);
 fsi
 fproc
```

En este algoritmo se hacen tres recorridos sucesivos de los caracteres de la cadena, cada uno de los cuales tiene un coste lineal sobre el número total de ellos, por lo que el coste en tiempo sigue siendo lineal. Como se utilizan dos pilas auxiliares, el coste en espacio también es lineal.

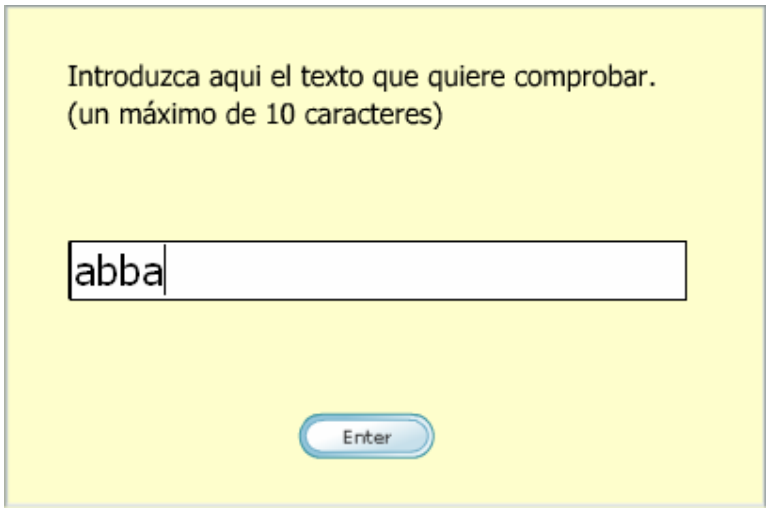
2.2.1.1. **Animación paso a paso**

Lo primero que aparece al ejecutar la animación, como en todas las realizadas, es la portada:

Para comenzar a ver la animación hay que pulsar el botón de inicio. A continuación se pide que se introduzcan los datos:

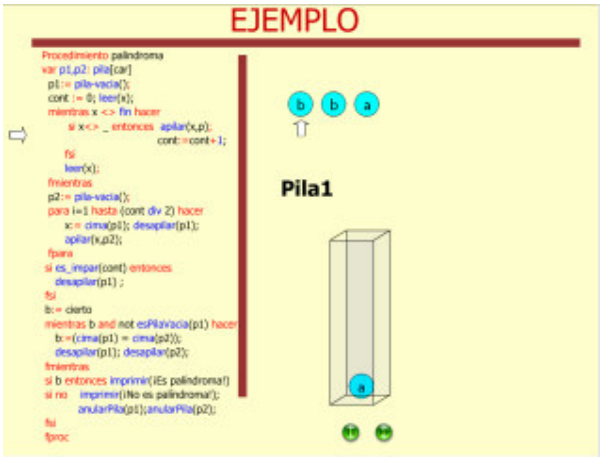


Aquí el usuario introducirá la cadena de caracteres que desee comprobar, deberá pulsar *Enter* para que los datos se guarden y comience la animación del algoritmo.
Por simplicidad, y para que la animación no se haga muy pesada y el dibujo muy engorroso, sólo se permiten cadenas de diez caracteres como máximo.
Supongamos que introducimos la cadena: “abba” y veamos qué ocurre:

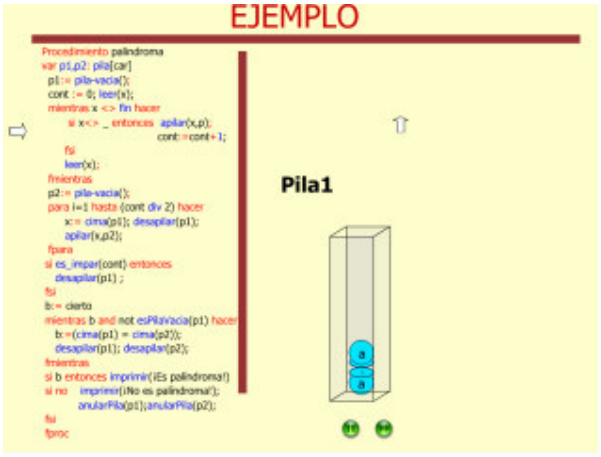


Pulsamos *Enter*, a continuación observamos que se empieza a ejecutar el algoritmo:

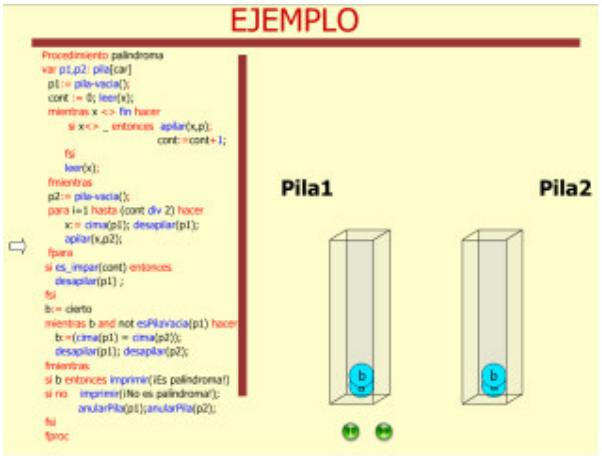
1.Se apila el primer caracter en Pila1



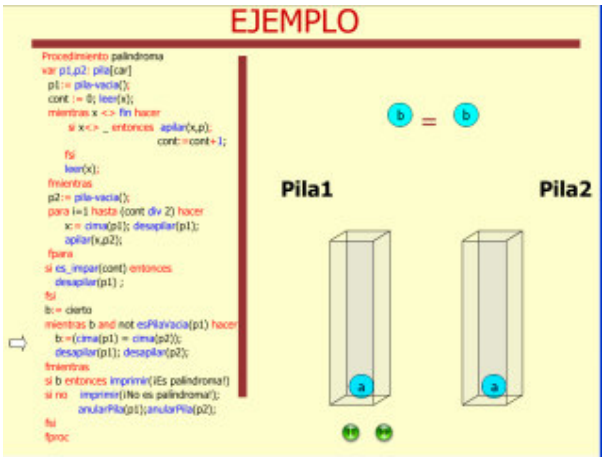
2. Se apilan todos los caracteres en Pila1



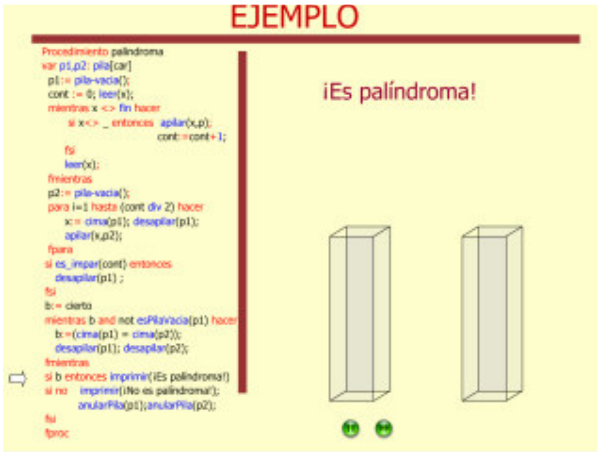
3. Se transfieren la mitad de los caracteres de Pila1 a Pila2



4. Se comparan las cimas dos a dos



5. Obtenemos el resultado



En este caso la ejecución termina correctamente y muestra un mensaje informando de que la cadena, efectivamente, es palíndroma. En caso de que la

expresión elegida no lo fuera, la ejecución se detiene en el momento en el que el primer par de elementos de las cimas de las dos pilas no coincidiese y aparece un mensaje advirtiéndolo.

Si se desea realizar otra evaluación es necesario volver a iniciar la animación desde el principio siguiendo los pasos anteriormente señalados.

2.2.1.2. Implementación con Flash

Esta animación está formada por un archivo .fla en el que se ha llevado a cabo la implementación del algoritmo que utiliza dos pilas para realizar la comprobación de expresiones palíndromas. Las ventajas que ofrece la estructuración de la animación en capas hacen que se siga utilizando en este caso.

En esta animación la mayor parte de la funcionalidad se encuentra en la capa *acciones*. Veamos el contenido de los fotogramas de esta capa:

- *Fotograma 1*: continuando con la misma estructura, en este fotograma declaramos todas las variables que se van a utilizar a lo largo del algoritmo, así como funciones que permiten implementar el movimiento de los objetos (las mismas que vimos en la sección 2.1.1.2. para evaluación de expresiones postfijas) y una función auxiliar que comprueba si un número dado es Impar (se utilizará más adelante en el algoritmo).
  En esta ocasión hemos introducido la novedad de mecanizar la desaparición de la portada. Hemos visto que la propiedad ._alpha de los objetos de tipo MovieClip también se puede variar implementando una función que utiliza la clase Tween de Action Script antes mencionada:

```
//cambio en la propiedad de un objeto
function tweenflecha(easeType, bola,begin,end,time,propiedad) {
   //varía el valor de propiedad del objeto bola
   ballTween = new mx.transitions.Tween(bola,propiedad, easeType, begin, end, time);
}
```

  De esta forma se ha centralizado el código de la animación en la capa acciones, el resto de capas sirven como auxiliares y contienen los objetos que aparecen en la pantalla. Esto hace que sea más fácil su localización.
- *Fotogramas 2 y 3*: aquí es donde se realiza el bucle que ejecuta la desaparición de la portada. Una vez más se aplica la característica de Flash de la que ya hemos hablado que hace que los bucles se programen avanzando y retrocediendo en la línea del tiempo. Además el fotograma 3 controla la pulsación del botón de inicio *Enter* de la pantalla de introducción de la expresión palíndroma a comprobar.
- *Fotograma 4*: este fotograma se encarga de parar la animación mediante la función *stop(),* de modo que cuando se llega a la pantalla en la que hay que introducir la expresión palíndroma, la animación no continúa hasta que la secuencia de caracteres se haya introducido y el usuario decida pulsar el botón *Enter*.
- *Fotogramas 5 a 7*: en el fotograma 5 se recoge la información introducida por el usuario de la siguiente manera:

```
//cadena es el texto que se introduce por pantalla
cadena = it1.text;
//cadena2 es la cadena dividida en caracteres
cadena2 = cadena.split("");
//contador de pila1
i = 0;
//recorre la lista de caracteres cadena2 creando dos bolas de cada caracter
while (i<cadena2.length) {
    initObjeto = new Object();//inicialización del objeto bola
    initObjeto._y = -50;
    initObjeto._x = 400+50*i;
    vari = "caracter"+i;//nombre del objeto
    vare = "carac"+i;
    //se crea el objeto MovieClip que aparece por pantalla y se introduce en la variable
    //que representa la pila 1
    p1[i] = _root.attachMovie("bola"+i, vari, this.getNextHighestDepth(), initObjeto);
    p1[i].td.text = cadena2[i];
    i++;
}
esImpar(i);//comprueba si la longitud de la palabra es impar
```

En la variable *cadena* se encuentra la secuencia de caracteres introducida por el usuario en el objeto de texto dinámico que aparece en la pantalla. Una vez que se han separado cada uno de los caracteres de la cadena, se crea una bola para cada uno de ellos que se apila en la primera de las pilas. Estas bolas son las que posteriormente mostrarán el funcionamiento del algoritmo. Además, se comprueba si la longitud de la palabra que se ha introducido es impar, lo cual será útil más adelante para hacer una de las comprobaciones que forman parte del algoritmo.
En los fotogramas siguientes (6 y 7) hemos programado el proceso de desaparición de esta pantalla y la aparición de la siguiente junto con las bolas que representan los caracteres.

- *Fotogramas 8 a 11*: en estos fotogramas se realizan los pasos iniciales del algoritmo. Se crean las flechas que guiarán al usuario a través de los pasos del algoritmo y la primera pila.
- *Fotogramas 12 a 17*: este algoritmo consta de tres bucles. El primero de todos ellos se ha programado en estos fotogramas. Este primer bucle apila cada una de las bolas (caracteres que forman la expresión) y para ello hemos incluido movimientos de los objetos *bola* y “saltos” en la línea del tiempo. Otro aspecto destacable, es que hemos incluido un nuevo efecto en las bolas a la hora de apilarlas. Debido al escaso espacio del que se dispone en la pantalla, hemos tenido que buscar una solución para poder permitir que las expresiones puedan tener diez caracteres y así dar más posibilidades cuando haya que introducir palíndromas:

```
//i es el número de caracteres que tiene la expresión
//p1 es la primera pila que aparece y que está representada por medio de un array
if (i>1 && i<p1.length){
    p1[i-1]._height=15;
}
```

Por medio de esta comparación simple, que se realiza cada vez que se apila una bola, conseguimos que si hay más de una en la pila y todavía quedan más por apilar, se cambie la propiedad ._heigh (altura) del objeto bola que se apila produciéndose un efecto visual de aplastamiento. Para

volver a la bola a su estado anterior (esto lo realizamos después al desapilar) lo único que hay que hacer es modificar de nuevo dicha propiedad.

- *Fotogramas 18 a 20*: en estos fotogramas se produce la transición al segundo bucle del algoritmo. Aquí aparece en pantalla la segunda pila de las que se utilizan y se producen los movimientos correspondientes de la flecha que sigue los pasos del algoritmo.
- *Fotogramas 21 a 27*: aquí se desarrolla el segundo bucle. Este bucle se encarga de desapilar la mitad de las bolas de la primera pila y apilarlas en la segunda, con lo cual lo único que hemos necesitado son movimientos de las bolas, "saltos" en la línea de tiempo y efectos de aplastamiento al igual que en los casos anteriores.
- *Fotogramas 28 a 31*: en estos fotogramas hemos implementado la sentencia if que aparece a continuación en el algoritmo. Si la expresión tiene un número impar de caracteres, se desapila de la primera pila la cima y desaparece la bola por medio del efecto de trasparencia que ya hemos mencionado anteriormente.
- *Fotogramas 32 a 36*: en este intervalo de fotogramas está el último bucle del algoritmo. Desapila la cima de cada una de las pilas que aparecen en la pantalla y compara los caracteres que contienen. En caso de que sean iguales, las bolas desaparecen lentamente atenuándose su color, de nuevo este efecto se consigue modificando el valor de la propiedad ._alpha. En caso de que no sean iguales, se realiza un "salto" hacia un fotograma más avanzado que detiene la animación y muestra el mensaje de error por pantalla.
- *Fotogramas 37 y 38*: Aquí se realiza la última comprobación del algoritmo, es decir, se realiza un "salto" al fotograma adecuado dependiendo de si la expresión ha resultado ser palíndroma o no.
- *Fotograma 39*: a este fotograma se llega como resultado de haber llegado a la conclusión de que la expresión que se evalúa no es palíndroma. En él se muestra el mensaje de error por pantalla y, mediante la función *unloadMovie()* de Action Script, se eliminan de la pantalla los objetos pila. Finaliza la animación y se detiene.
- *Fotograma 40* : realiza las mismas operaciones que el fotograma anterior, pero esta vez la expresión sí ha resultado ser palíndroma, con lo que se muestra por pantalla el mensaje adecuado y se detiene la animación.

Las siguientes capas que vamos a tratar son la capa *botones* y la capa *botonesgene*. En estas capas están todos los botones que aparecen en la animación, tanto el botón de inicio de la portada, como el botón *Enter* que sirve al usuario para introducir una expresión palíndroma y los botones verdes que sirven para cambiar la velocidad cuando se está ejecutando el algoritmo. Estas capas no tienen asociado código Action Script que maneje los botones porque de esto se encarga la capa *acciones* que es donde hemos implementado la funcionalidad de toda la aplicación.

Finalmente la capa *Capa1*, esta capa solamente contiene objetos al igual que las capas *botones* y *botonesgene.* En ella están incluidos todos los objetos de texto que aparecen fijos en cada una de las pantallas de la animación, como por ejemplo el escudo de la Universidad Complutense junto con todos los

textos informativos que aparecen en la portada. También, están incluidos el cuadro de introducción de expresiones palíndromas y el texto que contiene el algoritmo que se ejecuta en esta aplicación.

2.2.2. **Algoritmo utilizando una pila y una cola**

En esta segunda animación disponemos de una pila y una cola como estructuras auxiliares.
En este caso se trata de ir apilando los caracteres de la cadena al mismo tiempo en la pila (donde quedarán almacenados en orden inverso) y en la cola (donde quedarán almacenados en el orden de entrada). A continuación se hace un segundo recorrido comparando los elementos de la pila (que se extraen desde su cima) con los almacenados en la cola. La expresión será palíndroma cuando los elementos de ambas estructuras coincidan dos a dos.

El algoritmo empleado para realizar la animación en Flash tiene el siguiente aspecto: [2]

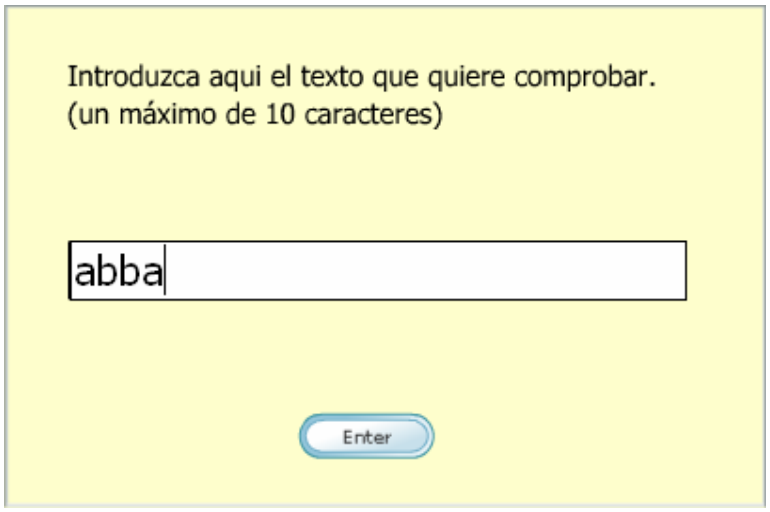
```
Procedimiento palindroma
var c: cola[car], p: pila[car]
  c:= cola-vacia();
  p:= pila-vacia();
  leer(x);
  mientras x <> fin hacer
   si x<> _ entonces pedir-vez(c,x);
                    apilar(x,p);
   fsi
   leer(x);
  fmientras
  b:= cierto
  mientras b or not es-cola-vacia(c) hacer
    b:= (primero(c)= cima(p))
    avanzar(c); desapilar(p);
  fmientras
  si b entonces imprime(¡Es palíndroma!);
  sino
    imprime(¡No es palíndroma!);
    anular-cola(c); anular-pila(p)
  fsi
fproc
```

En este algoritmo se hacen dos recorridos sucesivos, como cada uno de ellos es de coste lineal, el coste en tiempo total, sigue siendo lineal. Como ahora se utilizan una pila y una cola como estructuras auxiliares, el coste en espacio también será lineal.

2.2.2.1. Animación paso a paso

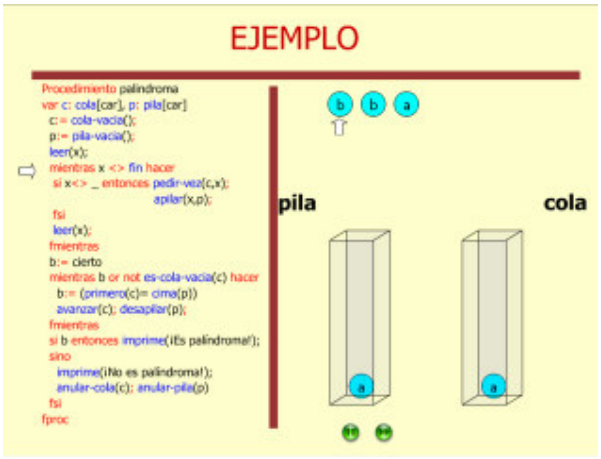
Lo primero que aparece al ejecutar la animación, como en todas las realizadas, es la portada:

Para comenzar a ver la animación hay que pulsar el botón de inicio. A continuación se pide que se introduzcan los datos, como en el ejemplo anterior del uso de dos pilas, introducimos la cadena “abba” y vemos qué ocurre:

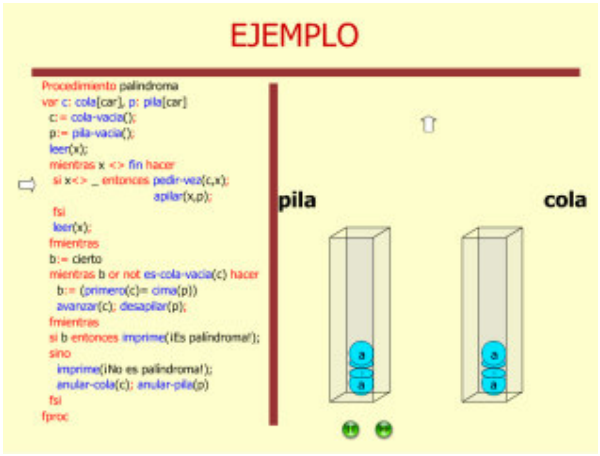


Pulsamos “Enter”, a continuación observamos que se empieza a ejecutar el algoritmo:

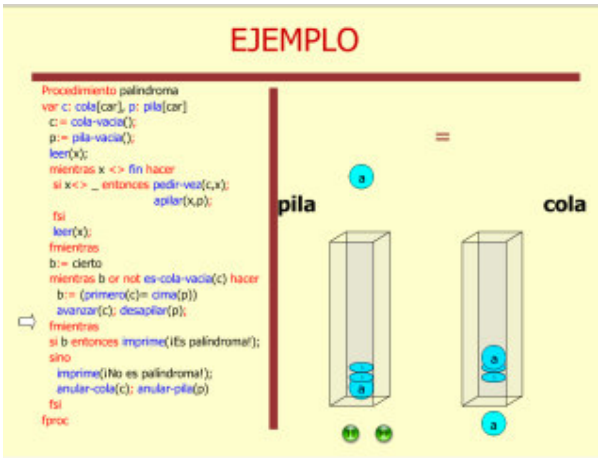
1.Se apila el primer carácter en la pila y en la cola



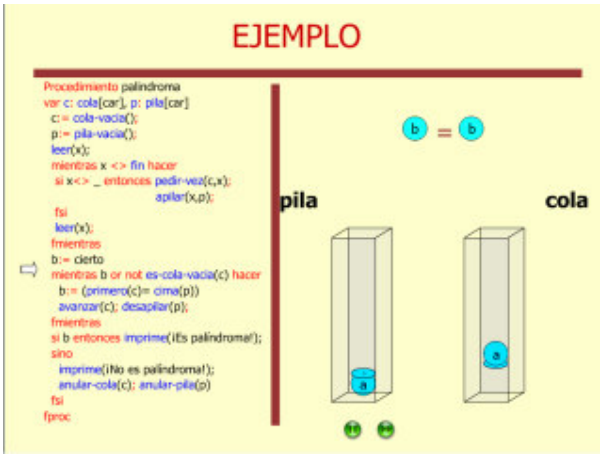
2. Se apilan todos los caracteres en la pila y en la cola



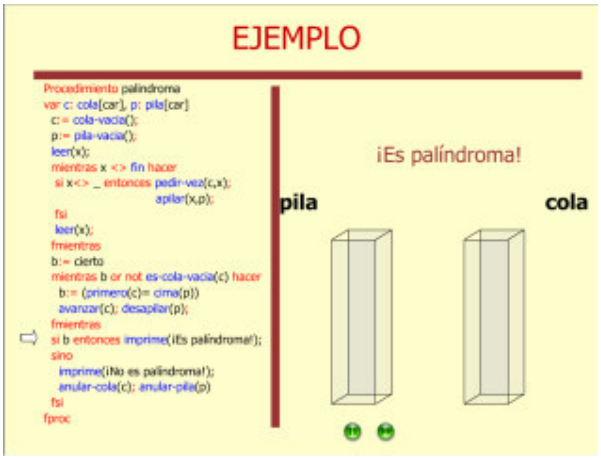
3. Sacamos los caracteres de la pila y de la cola (*)



4. Se comparan los caracteres dos a dos



5. Obtenemos el resultado



(*)Las colas suelen representarse en posición horizontal, sin embargo aquí están colocadas en posición vertical, esto es para permitir introducir cadenas lo suficientemente largas y permitir así que se vean claramente todos los caracteres de cada una de las estructuras.

En este caso la ejecución termina correctamente y muestra un mensaje informando de que la cadena, efectivamente, es palíndroma. En caso de que la expresión elegida no lo fuera, la ejecución se detiene en el momento en el que el primer par de elementos, el de la cima de la pila y el primero de la cola, no coincide y aparece un mensaje advirtiéndolo.

Si se desea realizar otra evaluación es necesario volver a iniciar la animación desde el principio siguiendo los pasos anteriormente señalados.

2.2.2.2. **Implementación con Flash**

Para realizar la implementación del algoritmo de comprobación de expresiones palíndromas hemos utilizado un archivo .fla al igual que en los casos anteriores. Este archivo contiene las mismas capas que el que hemos utilizado para el algoritmo con dos pilas, la diferencia es que es más corto, es decir, utiliza menos fotogramas porque en este caso sólo es necesario implementar dos bucles.

Veamos la distribución de los fotogramas de la capa *acciones* para este algoritmo que utiliza como TADs una pila y una cola:

- *Fotograma 1*: no hay ninguna novedad en cuanto a la estructura que seguimos en esta animación. En este fotograma declaramos todas las variables y funciones de movimiento necesarias para la ejecución del algoritmo (ver las funciones en la sección 2.1.1.2. de implementación evaluación de expresiones postfijas).
- *Fotograma 2*: en este fotograma hemos incluido la transición de pantalla, en concreto, la transición entre la portada y la pantalla de introducción de expresiones palíndromas. Como ocurría en las otras animaciones, aquí también está presente el efecto de trasparencia que se obtiene cambiando el valor del atributo ._alpha de cada uno de los objetos de la pantalla.
- *Fotogramas 3 y 4*: aquí se controla la parada de la animación hasta que se pulsa el botón *Enter* presente en la pantalla de introducción de palíndromas. Cuando el usuario pulsa este botón, la animación interpreta que ya ha introducido la palabra que deseaba y está listo para que comience el algoritmo.
- *Fotograma 5*: el código de este fotograma se ejecuta cuando el usuario ha pulsado el botón *Enter*. En él se guarda la secuencia de caracteres que forman la expresión palíndroma introducida por el usuario en la pantalla anterior:

```
//cadena obtenida de la pantalla de introducción de datos
cadena = it1.text;
//cadena con los caracteres separados
cadena2 = cadena.split("");
i = 0;//contador de caracteres de la palabra introducida.
while (i<cadena2.length) {
    initObjeto = new Object();//inicialización del objeto
    initObjeto._y = -50;
    initObjeto._x = 460+50*i;
    vari = "caracter"+i;
    vare = "carac"+i;
    //se hacen dos copias de la misma bola
    //una de ellas se introduce en p que es la pila
    p[i] = _root.attachMovie("bola"+i, vari, this.getNextHighestDepth(), initObjeto);
    p[i].td.text = cadena2[i];
    //la otra copia se introduce en c que es la variable que representa a la cola
    c[i] = _root.attachMovie("bola"+i, vare, this.getNextHighestDepth(), initObjeto);
    c[i].td.text = cadena2[i];
    i++;
}
```

En esta ocasión se hacen dos copias (se crean dos bolas) de cada uno de los caracteres de la expresión palíndroma que se almacena en la variable *cadena* y posteriormente se divide en caracteres guardándose en la variable *cadena2*. Una de las bolas se apila y la otra se introduce en la cola.

- *Fotograma 6*: este fotograma únicamente realiza la transición de la pantalla de introducción de expresiones palíndromas a la pantalla donde se visualiza la ejecución del algoritmo.
- *Fotogramas 7 A 10*: En estos fotogramas se llevan a cabo los primeros pasos de algoritmo. Se realizan movimientos de la flecha que indica el avance del algoritmo. También se crean la pila y la cola de la siguiente manera:

```
//se crea la pila (fotograma 9)
initObjeto = new Object();//inicialización del objeto
initObjeto._y = 350;
initObjeto._x = 740;
 _root.attachMovie("cola", colas,this.getNextHighestDepth(), initObjeto);
//movimiento de la flecha en sentido vertical
tweenball(mx.transitions.easing.None.easeNone, flecha, flecha._y, flecha._y+25,
velocidad2, "_y",10);

//se crea la cola (fotograma 10)
initObjeto = new Object();//inicialización del objeto
initObjeto._y = 350;
initObjeto._x = 540;
 _root.attachMovie("cola", pilas,this.getNextHighestDepth(), initObjeto);
 i=0;
//movimiento de la flecha en sentido vertical
tweenball(mx.transitions.easing.None.easeNone, flecha, flecha._y, flecha._y+50,
velocidad2, "_y",11);
```

Como se puede observar el procedimiento es el mismo que realizamos cuando creamos cada una de las bolas que contienen los caracteres. La diferencia es que en este caso no guardamos los objetos en ninguna

variable puesto que una vez creados no vamos a realizar operaciones con ellos, si no que van a mantenerse fijos en la pantalla.

- *Fotogramas 11 a 14*: entre estos dos fotogramas se desarrolla el primer bucle que se encarga de introducir en la pila y en la cola los caracteres que forman la expresión palíndroma. Para ello simplemente utilizamos movimientos que se realizan llamando a las funciones correspondientes declaradas en el primer fotograma e implementadas con la clase Tween como ya se ha explicado anteriormente. La cola está colocada en vertical y hemos convenido que los elementos se introduzcan por la parte superior y salgan por la parte inferior, mientras que en la pila ambos movimientos se realizan por la parte superior.
  En este caso, para solucionar el problema de la escasez de espacio en la pantalla donde se muestra el algoritmo también nos ha servido introducir el efecto de aplastamiento de las bolas.
- *Fotogramas 15 y 16*: estos son dos fotogramas de transición hacia el segundo bucle del algoritmo. Ya no necesitamos la flecha que indica el caracter de la secuencia que se está analizando por lo que eliminamos este objeto flecha de la pantalla utilizando la función *removeMovieClip()* que elimina el objeto MovieClip que se le pasa como parámetro.
- *Fotogramas 17 a 21*: en el fotograma 17 comienza el segundo bucle. Este bucle compara el caracter que obtiene de la cima de la pila con el primer caracter de la cola y así sucesivamente con todos los caracteres hasta que la pila y la cola se vacíen o hasta que detecte que la palabra no es palíndroma. En el caso de que los caracteres sean iguales, estos desaparecen con un efecto de trasparencia igual que sucedía cuando se implementó el algoritmo que utilizaba dos pilas.
- *Fotograma 23*: a este fotograma se llega solo si la expresión resulta ser palíndroma y lo único que hace es mostrar el mensaje por la pantalla y para la animación mediante la función *stop()*.
- *Fotograma 24*:aquí se hace lo contrario que en el fotograma 23, se llega sólo si la expresión no es palíndroma y se indica por medio de un mensaje que se muestra en la pantalla. Después la aplicación se para y finaliza.

Las capas *botones* y *botonesgen* contienen los botones que aparecen durante toda la ejecución de la animación.

La capa llamada *Capa1* contiene, como ya ocurría en la animación anterior, todos los objetos de texto que aparecen fijados en cada una de las pantallas de la animación.

2.2.3. Algoritmo utilizando una cola doble

En esta tercera animación disponemos de una única cola doble como estructura auxiliar.
En este caso los caracteres leídos se van almacenando en la cola doble. La cola doble es una estructura que permite el acceso a sus elementos por los dos extremos, por lo que en este caso la comprobación es más sencilla, ya que basta comparar si los extremos de la misma van coincidiendo dos a dos.
Si acabamos con todos sin encontrar ninguna discordancia, la expresión será palíndroma.

El algoritmo empleado para realizar la animación en Flash tiene el siguiente aspecto: [2]

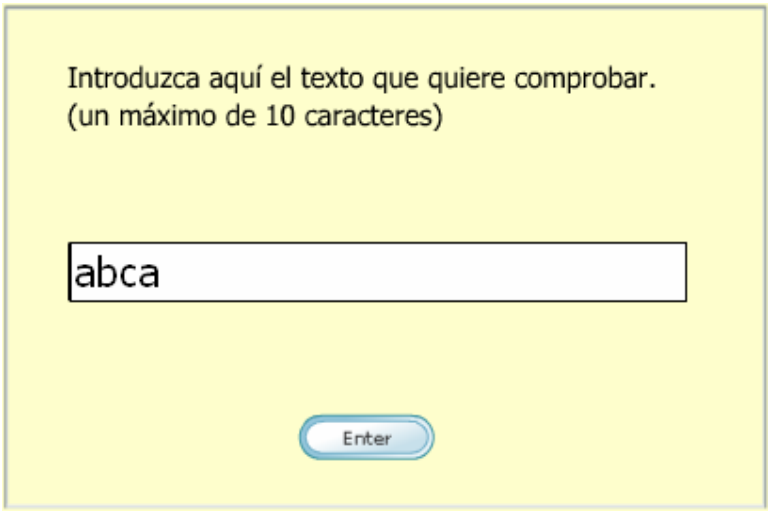
```
Procedimiento palindroma
var c: doble-cola[car];
  c:= dc-vacia();
  leer(x);
  mientras x not fin hacer
   si x not _ entonces poner-detrás(c,x);
   fsi
   leer(x);
  fmientras
  b:= cierto
mientras b or not es-dc-vacía(c) hacer
    b:= (primero(c)= último(c))
    quitar-prim(c);
   si not es-dc-vacía() entonces
        quitar-últ(c);
   fsi
  fmientras
  si b entonces imprime(Es palíndroma)
  sino
    imprime(No es palíndroma)
    anular-dc(c);
  fsi
fproc
```

En este caso se realizan dos recorridos sucesivos y el uso de una única estructura auxiliar no cambia los costes lineales en el caso peor, aunque si mejora la constante multiplicativa.

2.2.3.1. **Animación paso a paso**

Lo primero que aparece al ejecutar la animación, como en todas las realizadas, es la portada:

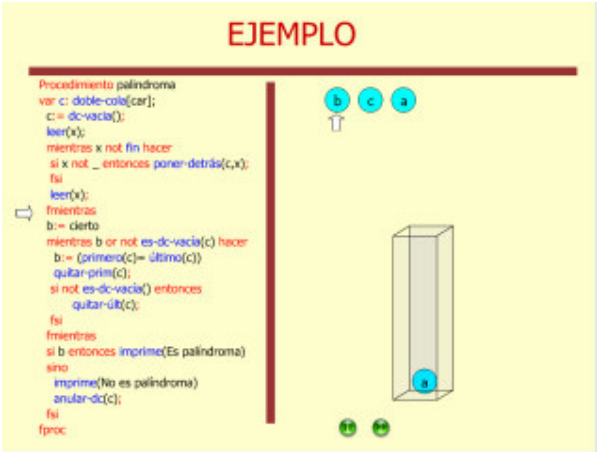
Para comenzar a ver la animación hay que pulsar el botón de inicio. A continuación se pide que se introduzcan los datos, ahora en este ejemplo, introduciremos la cadena “abca”, que no es palíndroma y nos informará del ello:



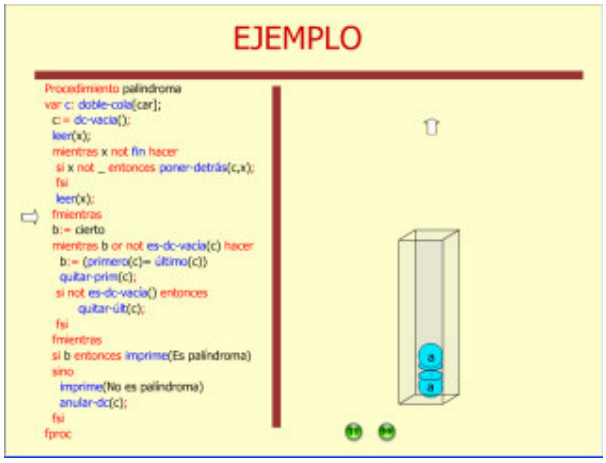
Pulsamos “Enter”, a continuación observamos que se empieza a ejecutar el algoritmo:

La estructura de la cola doble suele representarse en posición horizontal, sin embargo aquí para permitir una mejor visualización de los caracteres que se van introduciendo en ella si estos son un número elevado, hemos optado por representarla verticalmente.

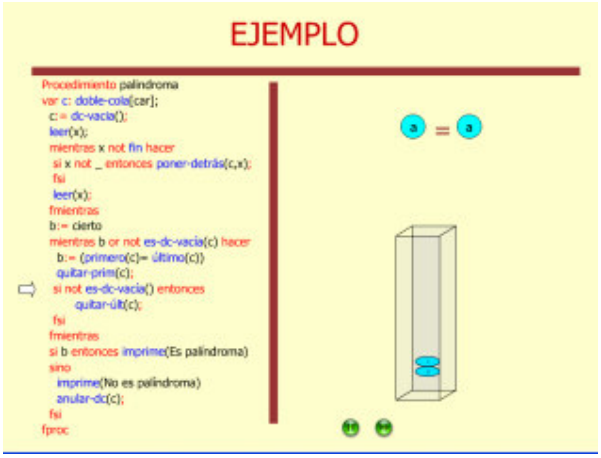
1. Introducimos el primer elemento en la doble cola



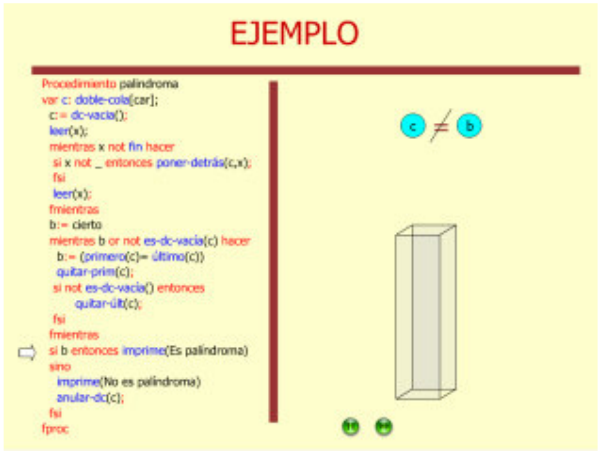
2. Se introducen todos los caracteres



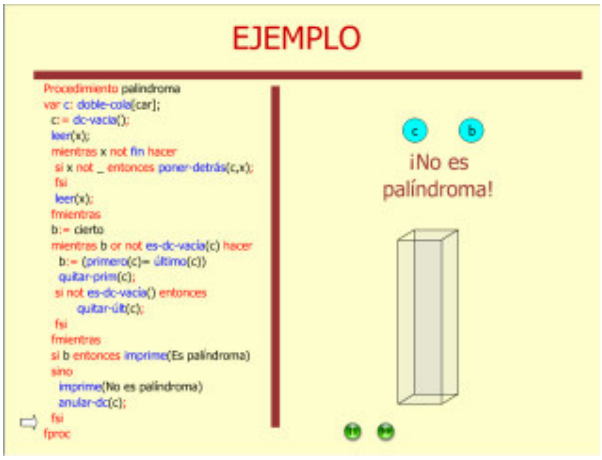
3. Se compara el primer par de caracteres, todo correcto



4. Se comparan los siguiente caracteres, no coinciden



5. Obtenemos resultado



En este caso la expresión elegida no es palíndroma, la animación se detiene en el momento en el que el primer par de elementos de las colas no coincide y aparece un mensaje advirtiéndolo.

Si se desea realizar otra evaluación es necesario volver a iniciar la animación desde el principio siguiendo los pasos anteriormente señalados.

2.2.3.2. Implementación con Flash

La implementación con Flash de esta animación es muy similar a las dos anteriores (apartados 2.2.1.2 y 2.2.2.2). También hemos realizado la implementación utilizando un único archivo .fla y la misma estructura en capas que en los casos anteriores.

En este caso el algoritmo utiliza una cola doble que hace que la implementación de la capa *acciones* esté estructurada de la siguiente manera:

- *Fotograma 1*: iniciamos la implementación de la misma forma que las anteriores. Declaramos todas las variables que se utilizan a lo largo de la animación para ejecutar el algoritmo que utiliza una cola doble para comprobar si una expresión es palíndroma o no.
- *Fotograma 2*: este fotograma hace la transición entre la portada y la pantalla de introducción de expresiones palíndromas aplicando el efecto de trasparencia.
- *Fotogramas 3 y 4*: se encargan de controlar el botón *Enter* que aparece en la pantalla de introducción de palíndromas. En cuanto se pulsa se inicia la ejecución del algoritmo.
- *Fotograma 5*: ahora se guarda la expresión palíndroma que ha introducido el usuario por pantalla de la siguiente manera:

```
cadena = it1.text;//cadena que se obtine a través del texto dinámico
cadena2 = cadena.split("");//cadena con los caracteres separados
i = 0;
while (i<cadena2.length) {
    initObjeto = new Object();//inicialización del objeto
    initObjeto._y = -50;
    initObjeto._x = 460+50*i;
    vare = "carac"+i;
    //se introduce la bola en la cola doble
    c[i] = _root.attachMovie("bola"+i, vare, this.getNextHighestDepth(), initObjeto);
    c[i].td.text = cadena2[i];
    i++;
}
```

  Es parecido a lo que se hacía en las animaciones anteriores. La variable *cadena2* contiene la expresión obtenida y se crea un objeto bola con el caracter correspondiente que se guarda en la cola doble representada por la variable *c*.
- *Fotogramas 6 a 10*: primero se realiza la transición de la pantalla de introducción de expresiones palíndromas hasta la pantalla donde aparece el algoritmo modificando la propiedad ._alpha de los objetos. Luego se crea el TAD cola doble que sirve como elemento auxiliar para hacer la comprobación con las expresiones.
- *Fotogramas 11 a 14*: en estos fotogramas hemos implementado el bucle que introduce en la cola doble cada uno de los caracteres de la expresión representados por una bola cada uno. Para ello recurrimos a las funciones de movimiento que situarán las bolas en el interior de la cola doble de una en una.
- *Fotogramas 15 y 16*: fotogramas que realizan los pasos previos al segundo de los bucles del algoritmo. Mueven la flecha que guía al usuario de los pasos del algoritmo y eliminan la flecha indicadora de caracteres de la expresión palíndroma porque ya no se va a utilizar más.
- *Fotogramas 17 a 21*: en estos fotogramas hemos hecho la implementación del segundo bucle. Este bucle saca de la cola doble un elemento de la parte superior, otro de la parte inferior y los compara. Repite este proceso hasta que la cola doble se quede vacía o haya alguna pareja de caracteres que no coincida. Utilizamos principalmente funciones de movimiento y "saltos" en la línea de tiempo.

- *Fotogramas 23 y 24*: en estos fotogramas termina la aplicación, en uno u otro según si la expresión que haya sido introducida es o no palíndroma, en cada uno de los dos fotogramas se muestra el mensaje correspondiente por pantalla y finaliza la animación.

Las capas *botones*, *botonesgene* y *Capa1* contienen los mismos elementos que las animaciones anteriores (apartados 2.2.1.2 y 2.2.2.2).

2.3. **Arboles 2-3-4**

En este proyecto hemos trabajado con el TAD de los árboles, en concreto con los árboles 234. Este tipo de árboles se caracteriza por ser vacío o tener nodos:
- De grado dos, tienen un elemento y dos hijos.
- De grado tres, tienen dos elementos y tres hijos.
- De grado cuatro, tienen tres elementos y cuatro hijos.

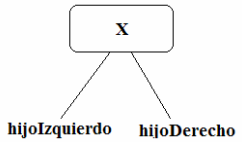
Estos elementos están ordenados en función del valor de sus claves.

En este proyecto trataremos dos aspectos fundamentales de este tipo de árboles: la inserción y la búsqueda.

2.3.1. **Inserción**

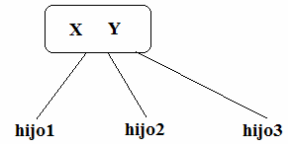
Primero vamos a tratar la inserción en árboles 234. Esta operación se caracteriza por insertar un elemento en el nodo que corresponda, según la ordenación del árbol, y tenga hueco libre. Además, al tratarse de un árbol ordenado, los elementos deben conservar el orden después de dicha inserción. Esto hace necesarias ciertas operaciones en el árbol para poder llevar a cabo la ordenación antes de la inserción del nuevo elemento.

Para realizar la inserción de nuevos elementos hemos utilizado un algoritmo recursivo [3]. Vamos a llamar T al árbol 234, y supongamos que queremos insertar el elemento N. La idea principal de este algoritmo es buscar en el árbol la hoja en la que debe insertarse N. Esto se realiza partiendo de la raíz y bajando por el árbol haciendo comparaciones con los elementos de los nodos y según los valores sean mayores o menores que N ir por uno u otro hijo (recordemos que cada nodo puede tener hasta cuatro hijos). Si la hoja en la que debería insertarse está llena, es decir, tiene tres valores, es necesario realizar fusiones para que haya hueco para el nuevo elemento. La inserción de N en T puede dar lugar los siguientes casos:
- Si T está vacío, se inserta el valor N en la raíz, quedando sus hijos vacíos.
- Si el árbol T tenía un solo valor X en la raíz y dos hijos:

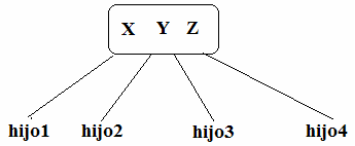


- Si T es hoja, se inserta N en el lugar que le corresponde del nodo, según sea mayor o menor que el valor X, y sus tres hijos quedan vacíos.
- Si T no es hoja, se inserta N en el hijo izquierdo si N<X o en el hijo derecho si N>X.

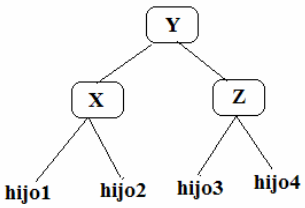
- Si T tenía dos valores X e Y (Y>X) en la raíz y tres hijos:



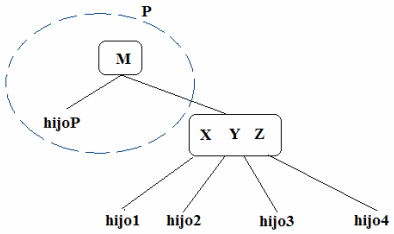
  - Si T es hoja, entonces se inserta N en el lugar que le corresponda según sea N<X o X<N<Y o X>Y y los hijos quedan vacíos.
  - Si T no es hoja, entonces se inserta N en el hijo que corresponda según sea N<X (se inserta en el hijo1) o X<N<Y (se inserta en el hijo2) o X>Y (se inserta en el hijo3)
- Si T tenía tres valores en la raíz y cuatro hijos:



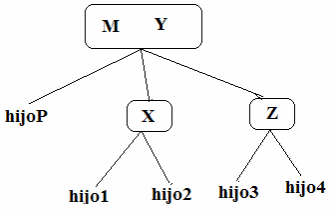
  - Si T no tiene árbol padre, entonces el nodo se tiene que dividir antes de insertar N. T queda de la siguiente manera:



  - Si tiene padre, llamémosle P, entonces T se fusiona con su padre antes de insertar N. Hay varios casos según el número de valores que tenga el padre:
    - Si tiene un valor:

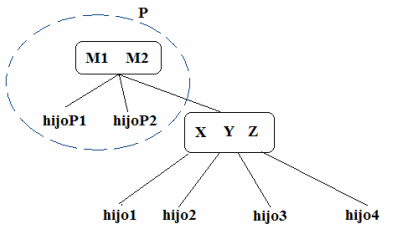


Entonces queda de la siguiente manera:

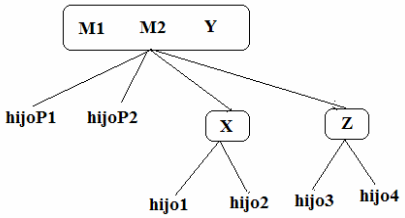


(el caso para el que T es hijo derecho es análogo)

- Si tiene dos valores:



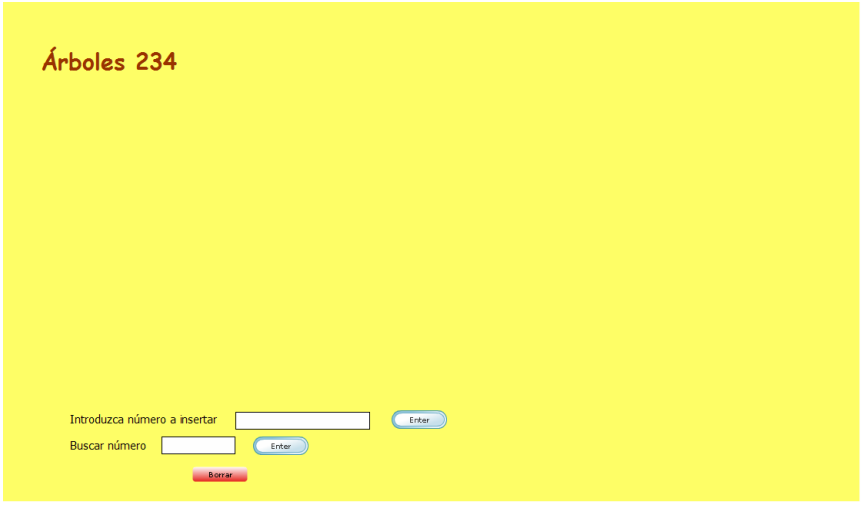
Entonces queda de la siguiente manera:



(el caso para el que T es hijo derecho e hijo intermedio es análogo)

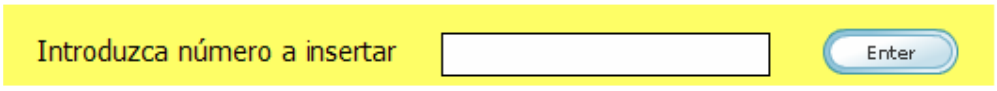
En esta animación se introducen novedades respecto de las anteriores, tanto en la visualización como en el manejo y la implementación del algoritmo.

2.3.1.1. Animación paso a paso



Como se puede observar, está estructurada de forma diferente que en las animaciones anteriores. Dispone de todo lo necesario para interactuar con el TAD árboles 234 insertando y realizando búsquedas. En apartado vamos a explicar el procedimiento para insertar elementos en el árbol.

Lo primero que hay que hacer es introducir



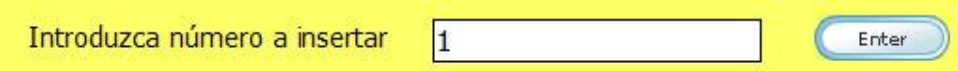
el número que se desea insertar en el árbol. Se pueden insertar números enteros desde el -99 al 99, en caso de que se inserte un número no adecuado

o uno que esté repetido, se advierte de ello al usuario mostrándole un mensaje por pantalla y se le pide que introduzca un número distinto.

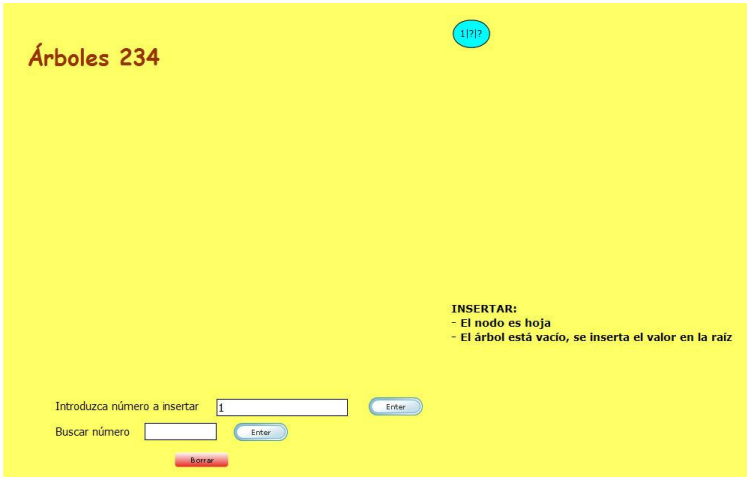
Después, para confirmar que el número introducido es el que se quiere insertar se pulsa el botón de la derecha *Enter*, con ello la animación ejecutará el algoritmo correspondiente y mostrará el resultado por pantalla.

En esta animación no se visualizan en movimiento los pasos que realiza el algoritmo para llegar a la solución, pero si se muestra la información detallada en formato de texto de como ha sido el proceso paso a paso. De esta manera el usuario puede observar como se ha desarrollado dicho algoritmo y el porqué de la solución mostrada.

Veamos un ejemplo, partimos de un árbol vacío e insertamos el número 1:



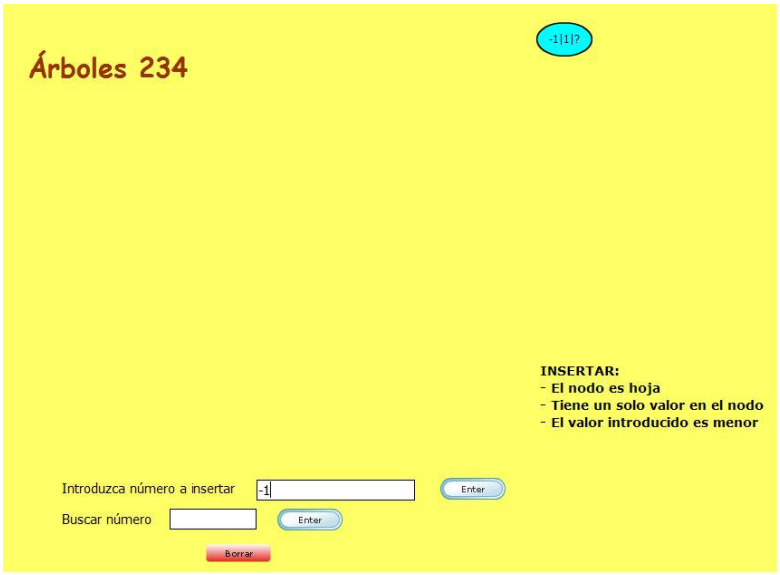
Pulsamos la tecla *Enter* y aparece automáticamente en la pantalla el resultado:



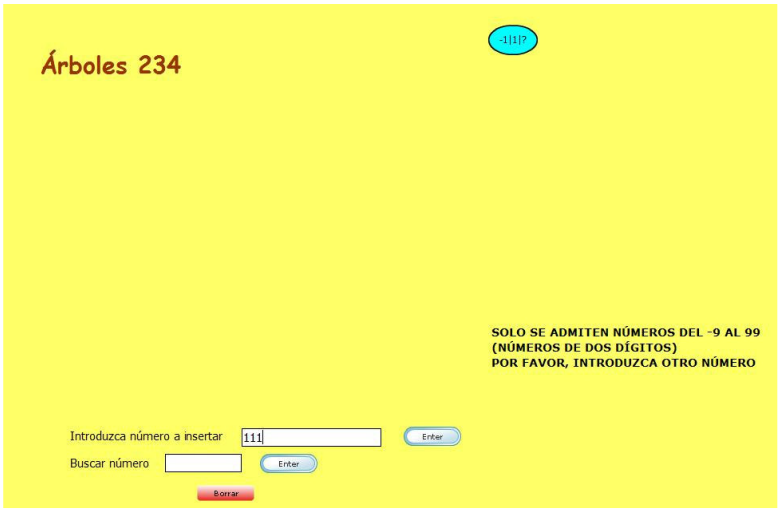
Como vemos aparece en la pantalla dibujada la raíz, que tiene como único valor el número 1. Además, se muestra el cuadro de texto con la secuencia de pasos que ha seguido el algoritmo para este caso concreto.

Veamos como va quedando el árbol al insertar varios números elegidos al azar:

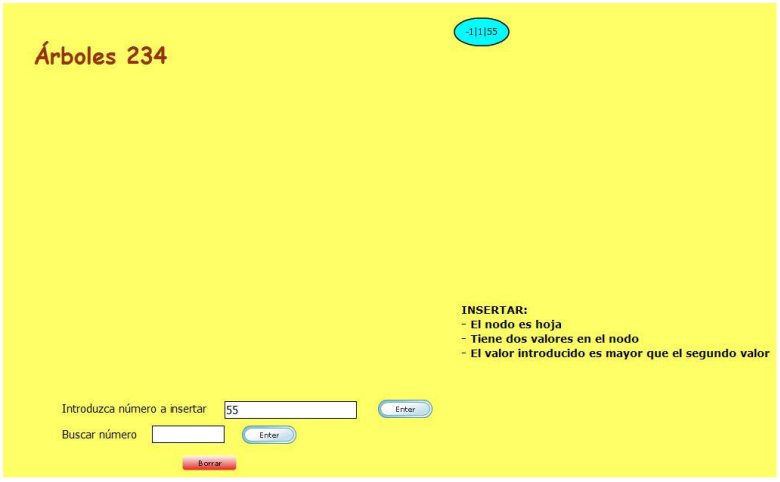
1. Introducimos el -1



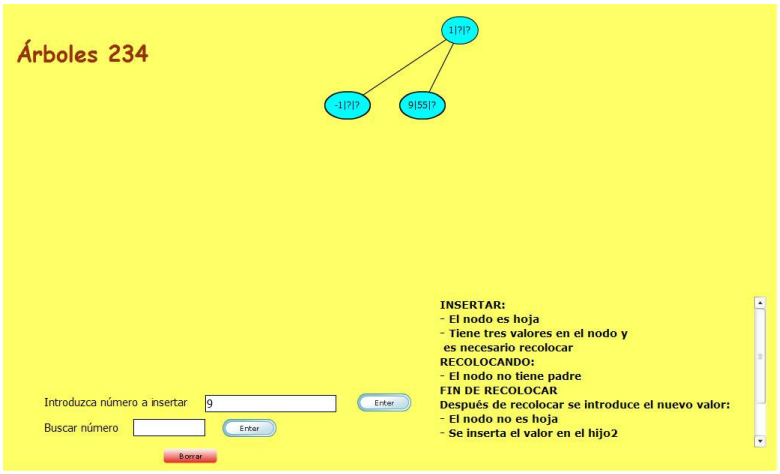
2. Introducimos el 111 (ejemplo de inserción de número incorrecto)



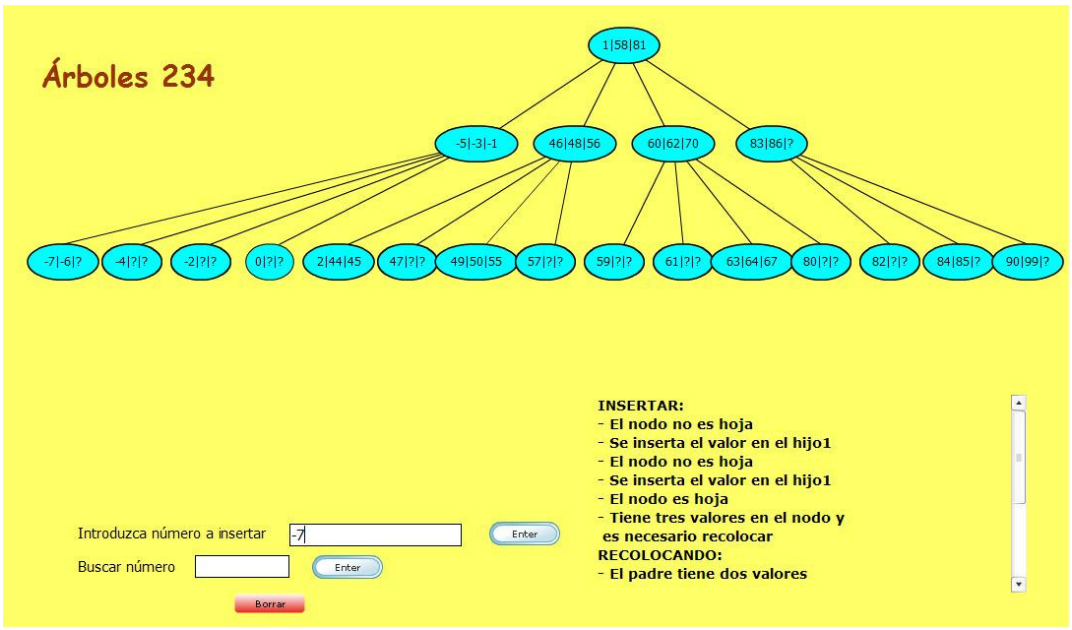
3. Introducimos el 55



4. Introducimos el 9



El árbol puede tener como máximo 3 niveles. Veamos como ejemplo un árbol de tamaño máximo para esta animación:



Si se desea empezar de nuevo, es decir, construir otro árbol nuevo, no es necesario ejecutar de nuevo la aplicación como ocurría en casos anteriores. Pulsando el botón *Borrar* se vacía el árbol y se puede comenzar a introducir valores de nuevo siguiendo los pasos ya señalados.

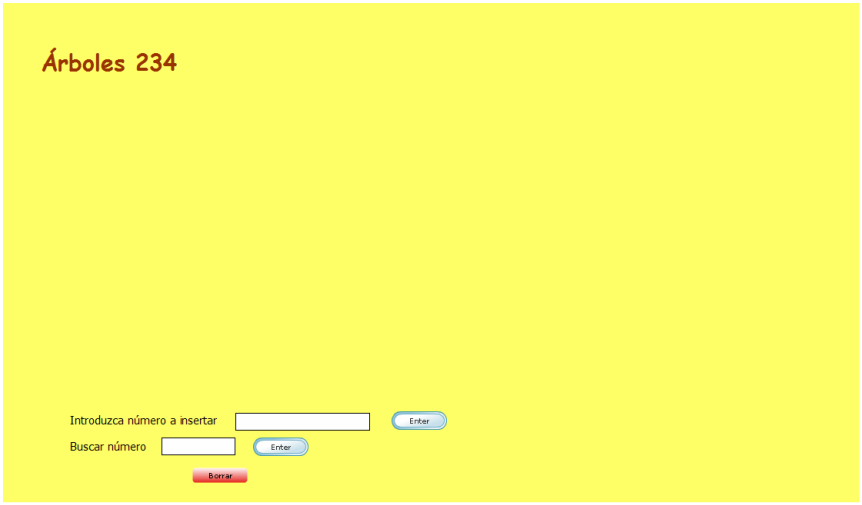
2.3.2. **Búsqueda**

Aquí tratamos la búsqueda de elementos en un árbol 234. Dicha búsqueda sigue un algoritmo recursivo que pasamos a describir brevemente a continuación:

- Un elemento no se encuentra en el árbol vacío.
- Si la clave del elemento a buscar coincide con alguna de las claves del nodo que estamos visitando, se ha encontrado el elemento buscado.
- Si la clave del elemento a buscar es menor que la clave del primer elemento de la raíz, se repite la búsqueda en el subárbol izquierdo de dicho nodo.
- Si no, si la clave del elemento es menor que la clave del segundo elemento de la raíz, se repite la búsqueda en el segundo subárbol de la raíz.
- Si no, si la clave del elemento a buscar es menor que la clave del tercer elemento de la raíz, se repite la búsqueda en el tercer subárbol de la raíz.
- Si no, se repite la búsqueda en el subárbol derecho de la raíz.

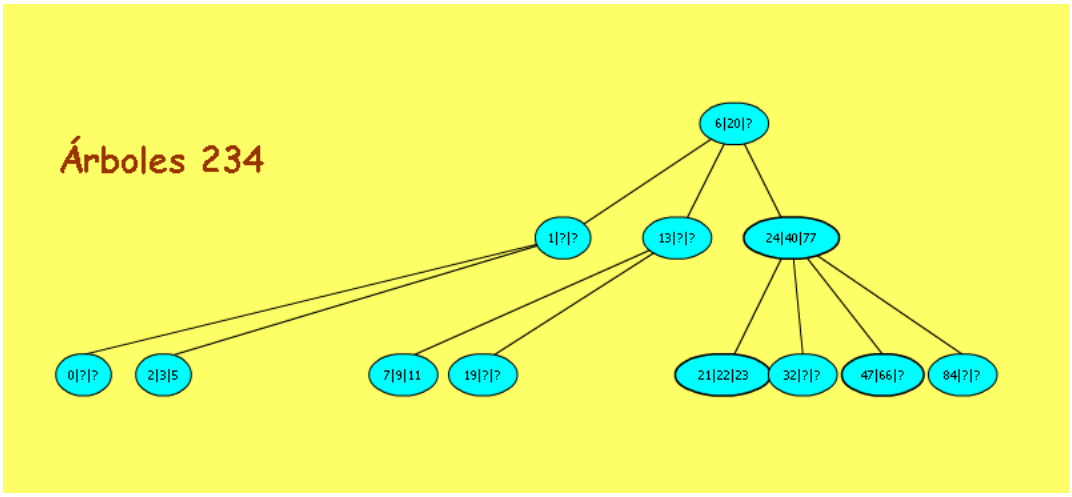
2.3.2.1 **Animación paso a paso**

A continuación mostraremos un ejemplo de ejecución de la animación.
Lo primero que nos encontramos es la siguiente pantalla



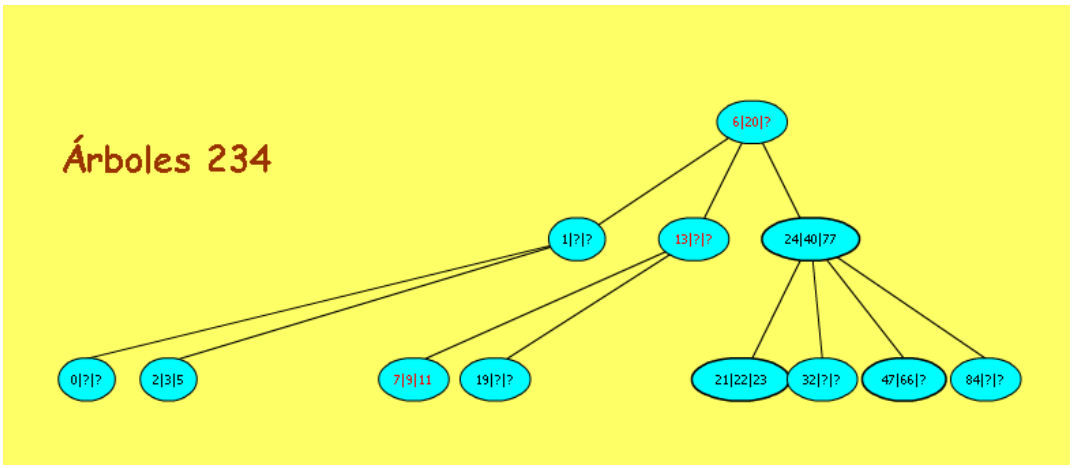
Donde se nos permite introducir elementos tanto para añadir como para buscar en el árbol.

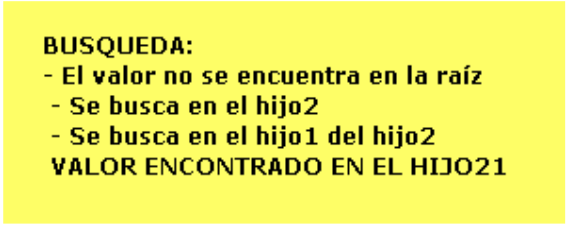
Supongamos que ya tenemos introducido el siguiente árbol:



Y que queremos buscar, por ejemplo, el elemento 9.
Para ello introducimos en el campo de texto “Buscar número” un 9 y pulsamos “Enter”:

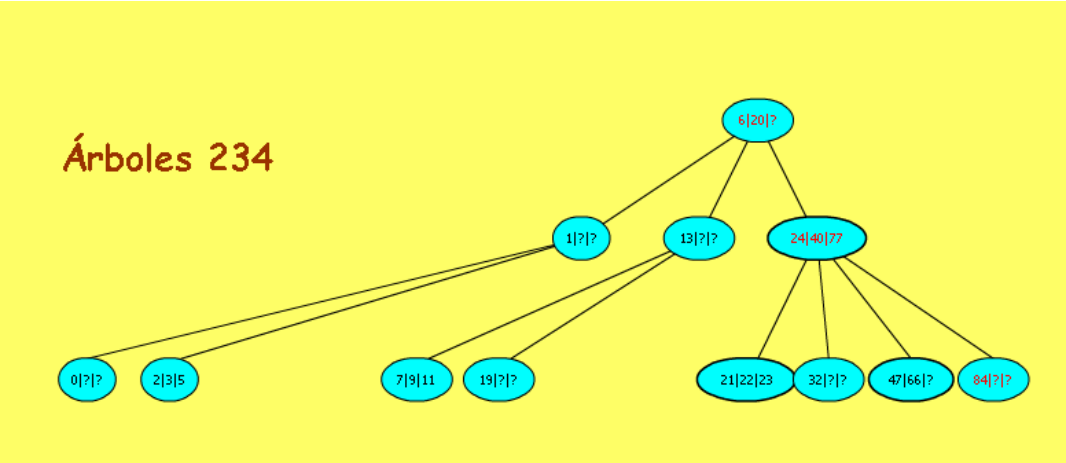
A continuación vemos señalados en rojo en el árbol aquellos nodos por los que va buscando el algoritmo y a la izquierda debajo de la pantalla una explicación de lo realizado:





Si, por ejemplo, introducimos un valor que no se encuentra en el árbol, por ejemplo, el número 100, la animación nos señalará en rojo los nodos por los

que ha ido buscando pero nos informará de que el nodo no ha sido encontrado. De esta forma:



2.3.3. Implementación en Flash

Para realizar esta animación, cómo se ha mencionado anteriormente, se han introducido cambios con respecto a las anteriores. Hemos utilizado una estructura distinta incorporando nuevos aspectos de ActionScript.

Anteriormente los tipos proporcionados por ActionScript habían sido suficientes para representar los TADs pilas y colas, en las que bastaba con el tipo Array. En este caso vimos la necesidad de crear objetos para representar árboles. Action Script ofrece esta posibilidad permitiendo crear archivos especiales de Action Script (*.as) que puede utilizar la animación principal creada en un archivo .fla. Además esto permite programar la animación en un solo fotograma.

Por todo ello, esta aplicación esta formada por tres archivos: Nodos.as, Arbol.as, Movimiento.as y Arboles234.fla (donde se encuentra la animación principal), necesarios para ejecutarla correctamente. A continuación explicamos uno a uno el contenido de los archivos y su funcionalidad dentro del conjunto de la aplicación.

2.3.3.1. Nodos.as

En Nodos.as es donde está implementado el objeto Nodo mediante una clase llamada de la misma forma. Un nodo está formado como máximo por tres

valores, tal y como hemos especificado antes, y operaciones para acceder o modificar dichos valores.

Veamos como queda el código de esta clase:

```
class Nodo {
	//defino el nodo del árbol que va a tener a lo sumo tres valores
	var valor1:Number;
	var valor2:Number;
	var valor3:Number;

	//constructor
	function Nodo() {
		valor1 = null;
		valor2 = null;
		valor3 = null;
		}

	//accesores y mutadores de la clase Nodo
	function getValor1():Number {
		return valor1;
	}
	function getValor2():Number {
		return valor2;
	}
	function getValor3():Number {
		return valor3;
	}
	function setValor1(numero):Void {
		valor1 = numero;
	}

	function setValor2(numero):Void {
		valor2 = numero;
	}
	function setValor3(numero):Void {
		valor3 = numero;
	}
}
```

Esta clase tiene como atributos los tres valores posibles que puede tener el nodo y los accesores y mutadores necesarios para acceder y modificar los atributos de la clase.

2.3.3.2. **Movimiento.as**

En este archivo hemos implementado la clase Movimiento. Esta clase se encarga de crear los nodos del árbol que se dibuja en la pantalla. Tiene como atributo un objeto MovieClip (que hemos llamado *texto*) que representa un nodo del árbol 234. De esta forma, cada vez que se crea un objeto de esta clase se está creando la representación de un nodo del árbol. Veamos que aspecto tiene la constructora de esta clase:

```
//constructora
function Movimiento(nombre,prof,x0,y0){
		var initObjeto = new Object();//inicialización del objeto
		initObjeto._y = y0;
		initObjeto._x = x0;
		//texto es la variable atributo de la clase Movimiento
		//en ella se guarda una bola de tipo MovieClip
		texto=_root.attachMovie("bola0",nombre,prof,initObjeto);
	}
```

Como podemos observar, para crear un objeto de tipo Movimiento es necesario indicar el nombre del objeto, la profundidad a la que se quiere colocar en la pantalla, la anchura y la altura. La constructora introduce en la variable atributo de la clase Movimiento un MovieClip *bola* que tiene las características que se han especificado.

Como toda clase de estas características, la clase Movimiento dispone de accesores y mutadores. Tenemos un único accesor al atributo *texto* y cuatro mutadores que lo modifican de diferente forma según si el nodo que se quiere modificar es de uno, dos, tres valores o simplemente se quiere vaciar un nodo:

```
//nodo con un valor
function ponTexto1(t){
	//vacía el texto
	texto.td.text="";
	//escribe el valor en el nodo
	texto.td.text=t+"|?|?";
	//cambia la anchura del nodo
	texto._width= (texto.td.text.length*12)-10;
	//._visible es una propiedad que permite ver o no ver el texto
	texto._visible=true;
}
//nodo con dos valores
function ponTexto2(t1,t2){
	//vacía el texto
	texto.td.text="";
	//escribe los dos valores en el nodo concatenándolos
	texto.td.text=t1+"|"+t2+"|?";
	//cambia la anchura del nodo
	texto._width= (texto.td.text.length*12)-10;
	//._visible es una propiedad que permite ver o no ver el texto
	texto._visible=true;
}
//nodo con tres valores
function ponTexto3(t1,t2,t3){
	//vacía el texto
	texto.td.text="";
	//escribe los tres valores en el nodo concatenándolos
	texto.td.text=t1+"|"+t2+"|"+t3;
	//cambia la anchura del nodo
	texto._width= (texto.td.text.length*12)-10;
	//._visible es una propiedad que permite ver o no ver el texto
	texto._visible=true;
}
//nodo sin valores
function textoVacio(){
	//vacía el texto
	texto.td.text="";		}
```

Vemos que las tres primeras funciones (*ponTexto1()*, *ponTexto2()* y *ponTexto3()*) sirven para modificar el nodo de forma que tenga los valores que se le pasan a estas funciones como parámetro. Para ello cambiamos el valor de las propiedades .text , ._width (anchura) y ._visible del campo de texto que contiene el MovieClip *bola*. Si por el contrario queremos vaciar un nodo, es decir, borrar todos los valores que contiene, recurrimos a la función *textoVacio()*.

Además tenemos dos funciones que nos sirven para cambiar el color del texto que aparece en los nodos del árbol,

```
//para ponerle un color rojo
function rojo (){
        texto.td.textColor=0xFF0000;
}
//para ponerle un color negro
function negro (){
        texto.td.textColor=0x000000;
}
```

las cuales nos resultan útiles a la hora de señalar los nodos por los que se realiza una búsqueda por el árbol y luego poder devolverles a su estado anterior. Lo único que hacemos es acceder a la propiedad .textColor del campo de texto del MovieClip para cambiar su valor, es necesario utilizar un valor numérico hexadecimal para indicar el color que queremos que aparezca.

Por último hemos añadido dos funciones auxiliares para dibujar las líneas que unen los nodos del árbol, imprescindibles para su representación:

```
//función que dibuja una línea
function linea(x0, y0, x1, y1) {
        _root.lineStyle(1, 0x000000, 100);
        _root.moveTo(x0, y0);
        _root.lineTo(x1, y1);
}

//función que borra todas las líneas de la pantalla
function borrar(){
        _root.clear();
}
```

La primera de ellas se encarga de dibujar una línea que empieza en (x0,y0) y termina en (x1,y1), siendo estas las coordenadas de la pantalla donde se sitúa la línea. La segunda función borra todas las líneas que haya dibujadas en la pantalla, esta función es muy útil, por ejemplo, cuando se desea borrar todo el árbol y empezar a crear otro nuevo.

2.3.3.3. **Arbol.as**

Arbol.as es la pieza clave de esta animación. En este archivo se ha creado la clase que representa el árbol 234 y las funciones que implementan los algoritmos de inserción y búsqueda.

En esta animación hemos limitado el número de niveles del árbol que dibujamos en la pantalla a tres para solucionar los diversos problemas de espacio que se nos presentaban debido al crecimiento excesivo del árbol.

Un objeto árbol está formado por:
- Un nodo de tipo *Nodo*, que puede tener hasta tres valores.
- Un padre de tipo *Arbol*.
- Cuatro hijos de tipo *Arbol*.

Además tiene una variable array, que hemos llamado M, que va a contener los nodos del árbol de tipo Movimiento ordenados, es decir, los objetos MovieClip para dibujar en la pantalla. Entonces la declaración de atributos de la clase Arbol queda de la siguiente manera:

```
//defino árbol tendrá el nodo el padre y como máximo cuatro hijos
var nodo:Nodo;
var padre:Arbol;//arbol padre
var lista1:Arbol;//hijo1
var lista2:Arbol;//hijo2
var lista3:Arbol;//hijo3
var lista4:Arbol;//hijo4
//para dibujar el árbol
var M:Array;
```

Cuando se crea un objeto de la clase *Arbol* la constructora inicializa todos los atributos que hemos visto. Si es necesario cambiar el valor de alguno de estos atributos hay que recurrir a las funciones accesoras y mutadoras que también hemos creado en este caso.

Para realizar la inserción de elementos hemos creado la función *insertar()* que es recursiva e implementa el algoritmo mencionado anteriormente (apartado 2.3.1). Tiene como parámetros el número que se quiere insertar, el árbol en el que se desea insertar y una variable llamada *texto* que se refiere al objeto de texto que aparece por pantalla y donde se muestran al final los pasos que ha seguido el algoritmo. Además, utiliza una función llamada *recolocar()* en el caso en el que se desea introducir un número en un nodo que está completo, es decir, tiene ya tres valores y es necesario realizar alguna rotación para poder introducir el nuevo elemento. La función *recolocar()* también es recursiva y recibe como parámetros el árbol y de nuevo la variable *texto*. Para colocar el árbol de manera que quede hueco para el nuevo elemento debe recorrer el árbol hacia arriba (hasta la raíz) reorganizando los valores de los nodos sin que se pierda el orden, para esto se utiliza el atributo *padre*.

La búsqueda de elementos en el árbol la hemos realizado mediante dos funciones. En un principio, lo ideal parecía una función recursiva para implementar el algoritmo (apartado 2.3.2) y para ello creamos la función *buscar()* que recibe como parámetros el árbol y el número que hay que buscar. Pero luego se nos presentaron problemas a la hora de representar en la pantalla la búsqueda ya que es complicado seguir los pasos que da la recursión. La solución que hemos encontrado y gracias a que hemos limitado el número de niveles del árbol a tres, es implementarla de manera directa, es decir, empezando en la raíz y teniendo en cuenta que los valores están ordenados, bajar por el árbol hasta encontrar el valor que buscamos. Para ello

hemos utilizado la función *dibujarBusqueda()* que a la vez que realiza el proceso de búsqueda modifica los nodos del árbol cambiando su color a rojo para que el usuario pueda apreciar el recorrido realizado. Además, dichos pasos se escriben en el objeto de texto de la pantalla para que quede aún más claro el proceso.

Para dibujar el árbol hemos recurrido a las siguientes funciones:

- *dibujaNodos()*: esta función se utiliza para crear en la pantalla todos los objetos *Movimiento* que representan los nodos y que luego van a formar el árbol. Todos estos objetos están introducidos ordenadamente en la variable array *M* antes mencionada y tienen en su propiedad ._visible un valor *false,* esto significa que no son visibles porque se supone que el árbol comienza siendo vacío.
- *dibujaArbol()*: esta función dibuja todo el árbol cuando se produce alguna modificación debido a la inserción de un elemento en el árbol. Lo primero que hace es borrar todas las líneas que había dibujadas en la pantalla por si alguna ya no se necesita. Después recorre el árbol viendo cuáles son los valores de los nodos, dibujando todas las líneas necesarias y al final hace visibles todos los nodos que tengan algún valor cambiando el valor de la propiedad ._visible a *true*.

El resto de funciones que podemos encontrar en este archivo son funciones auxiliares que nos han facilitado la implementación de los dos algoritmos principales. Veamos cuáles son:

- *grado()*: esta función tiene el siguiente aspecto:

```
//función de grado da el numero de valores que tiene el nodo
function grado(p1):Number {
        //si no tiene valores en el nodo
        if (p1.damenodo().valor1 == null) {
                return 0;
        } else {
                //si tiene un sólo valor en el nodo
                if (p1.damenodo().valor2 == null) {
                        return 1;
                } else {
                        //si tiene dos valores en el nodo
                        if (p1.damenodo().valor3 == null) {
                                return 2;
                        } else {//tiene tres valores en el nodo
                                return 3;
                        }
                }
        }
}
```

  Se encarga de averiguar el grado que tiene el nodo del árbol que estamos tratando. Lo cuál nos resulta útil cuando se desea insertar un valor en un nodo, sabremos que está completo si la función grado nos devuelve el valor tres.

- *borraNodos()* : Veamos cómo es:

```
//función que borra todo
function borraNodos():Void {
	var i:Number;
	for (var i = 0; i<=20; i++) {
		//borra las líneas de la pantalla
		M[i].borrar();
		//hace invisibles todos los nodos
		M[i].texto._visible = false;
		//vacía los nodos
		M[i].textoVacio();
	}
}
```

Esta función la utilizamos cuando se pulsa el botón *Borrar* que aparece en la pantalla. Nos sirve para eliminar de la pantalla el árbol que estaba dibujado: borrando las líneas, haciendo que los nodos se vuelvan invisibles y vaciándolos. De esta forma el usuario puede comenzar a introducir nuevos valores partiendo de un árbol vacío.

- *esHoja()* : es una función auxiliar que está definida de la siguiente manera:

```
//comprueba si tiene hijos
   function esHoja(p2):Boolean {
   if ((p2.lista1 == null) and (p2.lista2 == null) and (p2.lista3 == null) and (p2.lista4 == null))
   {//si todos los hijos están vacíos
		return true; //entonces sí es hoja
	} else {//si alguno de sus hijos tiene algún valor
		return false;//no es hoja
	}
   }
```

Se encarga de comprobar si el nodo que se está tratando es una hoja, es decir, si tiene hijos o no. Se utiliza en la función *insertar()*.

- *ponNegroNodos()* : esta función tiene el siguiente aspecto:

```
//función que pone a negro todos los números del árbol
function ponNegroNodos() {
	var i:Number;
	for (i=0; i<21; i++) {
		M[i].negro();
	}
}
```

Esta función se aplica cuando se quiere que todos los nodos vuelvan a su estado anterior cuando tras una búsqueda algunos de ellos han adquirido el color rojo. Se llama a esta función antes de realizar una operación de inserción o una operación de búsqueda.

2.3.3.4. **Áboles234.fla**

Finalmente llegamos al archivo que tiene la animación y que utiliza los tres anteriores para ejecutarla. Debido a esta nueva estructuración del código Action Script, tenemos la ventaja de que solo utilizamos un fotograma de la línea de tiempo. Esto hace que sea más sencilla la introducción de alguna nueva operación o cualquier modificación que se quiera realizar en el diseño.

Tenemos dos capas, la capa *acciones* que maneja la animación utilizando todas las clases que se han implementado en los archivos .as especificados anteriormente; y la capa *texto* que simplemente contiene el objeto de texto donde aparecen escritos los pasos que dan los algoritmos de inserción y búsqueda.

Vamos a analizar el contenido del único fotograma que tiene la capa *acciones*. Lo primero que hay que hacer para que la animación pueda utilizar los objetos *Arbol* y *Movimiento* es importarlos. Después ya podemos declarar una variable *Arbol* que va a representar al árbol que aparece en pantalla. Toda la funcionalidad se encuentra en los tres botones que aparecen en pantalla:

- El botón de inserción tiene el siguiente aspecto:

```
btn.onRelease = function() {
    comienzo.ponNegroNodos();
    //si el número tiene más de dos cifras pedimos que se vuelva a introducir otro número
    if (Number(it1.text)>99 || Number(it1.text)<-9) {
            barra.visible = false;
            textoArbol.text = "SOLO SE ADMITEN NÚMEROS DEL -9 AL 99 \n(NÚMEROS
DE DOS DÍGITOS)\nPOR FAVOR, INTRODUZCA OTRO NÚMERO";
    } else {
            //Si hay números repetidos, pedimos que vuelva a introducir número
            if (numeroRepetido(it1.text, nums)) {
                    barra.visible = false;//hacer desaparecer la barra de texto
                    textoArbol.text = "NÚMERO REPETIDO, POR FAVOR, INTRODUZCA
OTRO";
            } else {
                    //introducir el número a insertar en la lista de números del árbol
                    nums[i] = Number(it1.text);
                    textoArbol.text = "INSERTAR:\n";
                    //insertar el número introducido en el árbol
                    comienzo.inserta(Number(it1.text), comienzo, textoArbol);
                    //hacer visible la barra de desplazamiento de texto si es necesario
                    if (textoArbol.length>=250) {
                            barra.visible = true;
                    } else {
                            barra.visible = false;
                    }
                    i++;//contador de la lista de números del árbol
                    //dibujar el árbol con el nuevo valor insertado
                    comienzo.dibujaArbol(comienzo);
                    return comienzo;
            }
    }
};
```

Este botón se encarga de ejecutar la inserción del número introducido por el usuario. Antes de introducir el número hace dos comprobaciones. La primera es si el número que se introduce está dentro del rango adecuado, si no es así muestra un mensaje por pantalla indicándolo y pidiéndole al usuario que introduzca otro número. La segunda comprobación es si el número introducido está repetido, esto lo hace comprobando si el número se encuentra en la lista de números ya introducidos. Si está repetido, se muestra el mensaje correspondiente por pantalla.
Una vez insertado el número, se actualiza el dibujo del árbol que aparece por pantalla y se muestra el texto con los pasos que se han realizado.

- El botón de búsqueda se define de la siguiente manera:

```
btn2.onRelease = function() {
    //cambia a negro el color de todos los nodos
    comienzo.ponNegroNodos();
    numero = Number(it2.text);
    textoArbol.text = "BUSQUEDA: \n";
    //busca en el árbol el número introducido por el usuario
    comienzo.dibujarBusqueda(comienzo, numero, textoArbol);
    //hace visible la barra de desplazamiento de texto si es necesario
    if (textoArbol.length>=250) {
            barra.visible = true;
    } else {
            barra.visible = false;
    }
};
```

Este botón realiza la búsqueda del número en el árbol. Para ello llama a la función *dibujarBusqueda()* que a su vez también actualiza el árbol que se muestra por pantalla indicando en color rojo el camino que se ha seguido en el árbol para llevar a cabo la búsqueda. Si no encontrara el número en el árbol, lo indica mediante un mensaje de texto en el objeto habilitado para ello en la pantalla.

- El botón *Borrar* se implementa así:

```
borrar.onRelease = function() {
    textoArbol.text="EL ÁRBOL SE HA BORRADO, PUEDE VOLVER A COMENZAR";
    //borra los nodos del árbol
    comienzo.borraNodos();
    //crea un nuevo árbol
    comienzo=new Arbol();
    comienzo.dibujaNodos(comienzo);
    //vacía la lista de números introducidos
    nums = [];
    i=0;
}
```

El botón de borrado tiene la finalidad de poder empezar de nuevo, para ello es necesario reiniciarlo todo. Se borran los nodos antiguos y se crea un nuevo árbol vacío. También es necesario vaciar la lista de números introducidos para poder insertar los valores que se deseen.

Para comprobar si los números están repetidos utilizamos una función auxiliar que también declaramos en el primer fotograma de la capa *acciones*:

```
//función que comprueba si un número está repetido
function numeroRepetido(numero, nums):Boolean {
        var b:Boolean;
        b = false;
        for (i=0; i<nums.length; i++) {//recorre la lista de números introducidos
                if (numero == nums[i]) {//si el número está en la lista
                        b = true;
                }
        }
        return b;
}
```

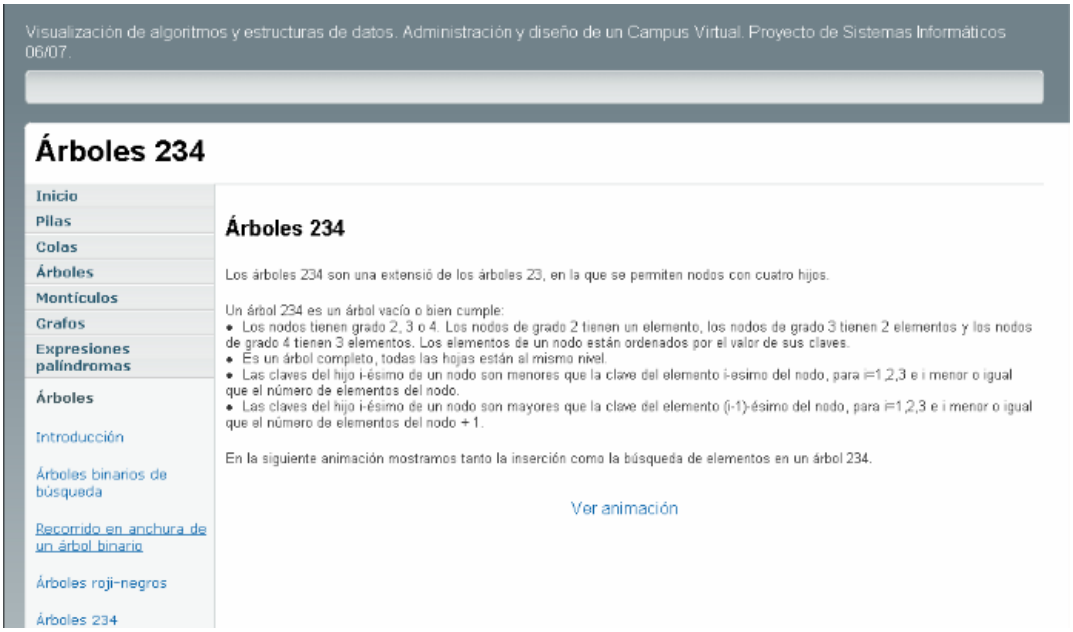
Esta función recorre la lista de números que se han introducido (que guardamos en la lista *nums* cada vez que se produce una inserción) y comprueba que ninguno de los números que hay en la lista es igual al que vamos a insertar.

El resultado es una aplicación completamente interactiva y de fácil manejo y mantenimiento.

3. **Aplicación web**

Hemos realizado también un entorno web donde mostramos todas las animaciones realizadas junto con una breve explicación de las mismas.
Se ha mantenido el diseño que realizaron los alumnos R. Carrasco, J. Crespo y A. López [1] durante el proyecto anterior. El diseño usado es intuitivo, modular y flexible puesto que emplea hojas de estilo, permitiendo cambiar el estilo de todo el entorno de forma sencilla. Sobre este entorno citado, se han añadido las nuevas páginas junto con las nuevas animaciones realizadas durante este proyecto.

A continuación mostramos una captura de pantalla de una de las páginas:



En la captura anterior se muestra el aspecto común de todas las páginas:

- A la izquierda observamos un menú donde se muestran los distintos tipos abstractos de datos realizados tanto en este proyecto como en el anterior [1]. Pinchando en cualquiera de ellos, aparecerá desplegado los distintos algoritmos u operaciones realizados para el tipo en cuestión.
- En la parte central se observa una breve introducción a la animación y un enlace para acceder a la animación flash en cuestión. Dicha animación se mostrará en una nueva ventana.
- El entorno dispone de un buen sistema de navegación, intuitivo y fácil, mediante una serie de enlaces se permite navegar de unas navegaciones a otras de manera sencilla.
- Disponemos también de un enlace al manual detallado de ActionScript, para resolver cualquier posible duda que se tenga de la realización de cualquiera de las animaciones.

4. Tutorial de ActionScript

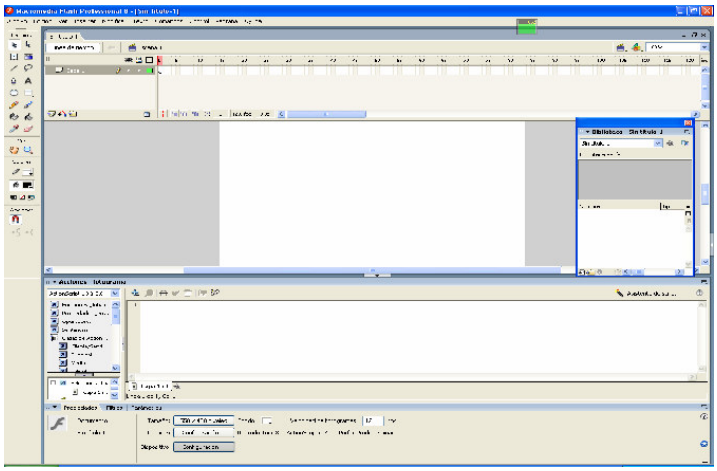
4.1. *Manual flash:*

4.1.1. Por qué utilizar Flash para realizar animaciones de TADs

Flash nos proporciona un entorno adecuado para realizar animaciones de TADs al permitir programar los algoritmos en su lenguaje Action Script, interactuar con el usuario de una manera extremadamente sencilla utilizando textos dinámicos, botones y una amplia gama de componentes suministrados por Flash y por último y posiblemente el mayor aliciente que puede hacer que nos decantemos por su utilización con este fin, el potencial de dibujo de Flash que nos permitirá realizar un entorno muy atractivo para el usuario de una manera relativamente sencilla y rápida. Además, con las últimas versiones de este producto se nos facilita en gran manera la interpolación de movimiento consiguiendo así animaciones mucho más útiles.

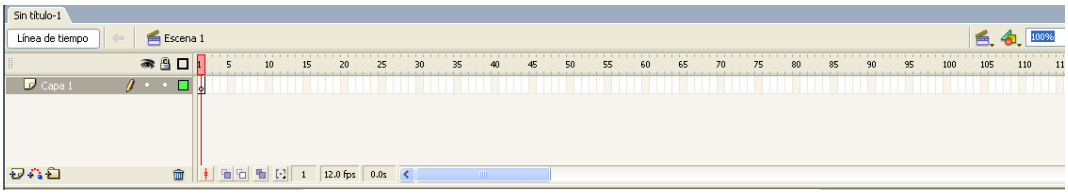
4.1.1.1. Introducción al entorno

Al iniciar una nueva aplicación Flash nuestra pantalla tendrá un aspecto muy similar a éste:



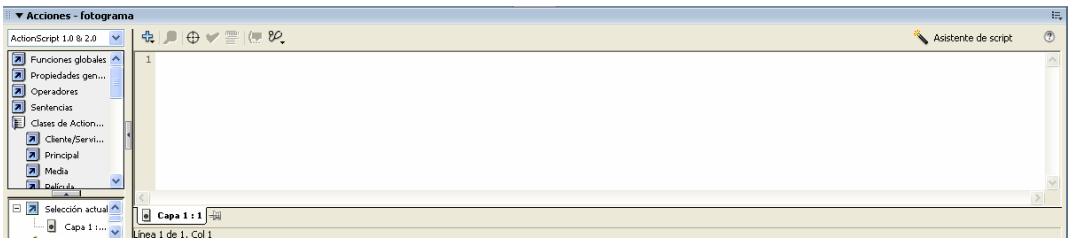
Pasemos a desglosar esta pantalla:

1. El stage (escenario) situado en el centro de la pantalla es la pantalla en blanco donde vamos a realizar la animación.
2. La línea de tiempo situada en la parte superior de la pantalla está compuesta por Frames (cuadros) y Layers (capas). Un frame representa cada una de las viñetas que van a aparecer en una animación. Las características de una viñeta se pueden dividir en varias capas de forma que la suma de los frames de las distintas capas en un instante de tiempo muestran lo que el usuario visualiza.

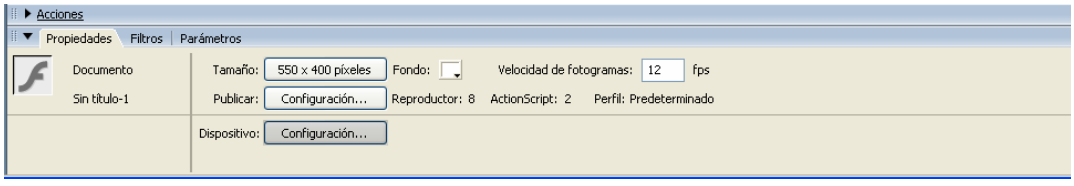


3. En la parte izquierda de la ventana podemos observar la paleta de herramientas que vamos a utilizar para dibujar, no merece extendernos en ella ya que es muy similar a cualquier otra que encontremos en el mercado.

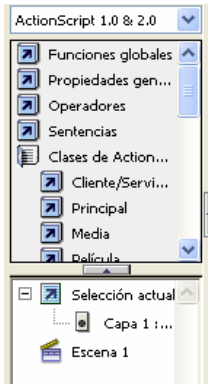
4. Bajo el Stage, podemos encontrar acciones, aquí es donde se podrán programar los objetos que se introduzcan en el stage. Hay que tener cuidado porque si se selecciona un objeto del stage, la programación que se haga aquí será relativa a ese objeto, por ejemplo si se selecciona un botón para programar su evento onRelease sólo se tendrá que poner onRelease {}, sin embargo si se hace sin tener seleccionado nada se tendrá que especificar para qué botón se desea programar. Esta programación también es susceptible a la capa que se tenga seleccionada, los eventos que se programen cuando esté seleccionada una capa serán visibles para esa propia capa y para las que estén situadas en una posición inferior a ésta en la línea de tiempo. Por este motivo se aconseja siempre poner una primera capa superior a todas denominada acciones para programar ahí todos los eventos que se deseen.



5. Tras el cuadro acciones podemos encontrar otro cuadro con las pestañas propiedades, filtros… éstas dependerán de la versión de Flash de que dispongamos pero en cualquier caso el menú más importante será el de propiedades, que es común a todas las versiones, donde podremos configurar si no hay nada seleccionado en el stage características como el tamaño de nuestra aplicación, la velocidad en frames, etc. Si se tiene algún objeto seleccionado se podrán modificar las propiedades de éste como ancho, alto, posición, etc.



6. Por último, comentar que a la izquierda del cuadro acciones tenemos el manual Action Script. No nos vamos a detener en explicarlo ya que es muy extenso pero queremos mencionar que suele resultar de bastante ayuda y que merece la pena perder unos minutos en curiosearlo.



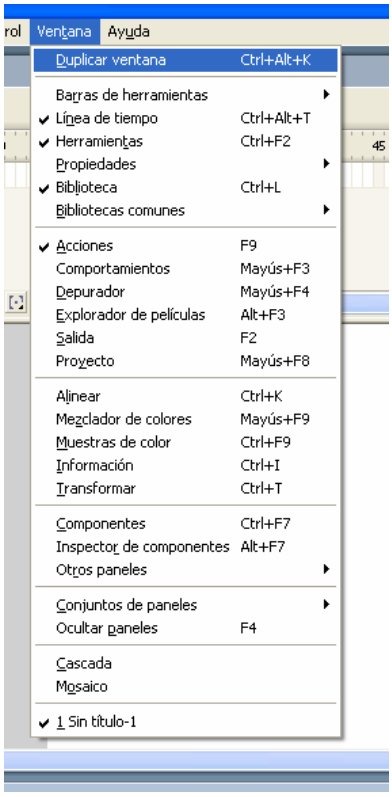
En las opciones del menú encontramos:

1. En el menú Archivo, además de las opciones básicas podemos encontrar algunas útiles como:
   a) Importar: permite importar objetos a la biblioteca, estos objetos pueden ser cualquier imagen o dibujo que descargues de Internet o creado por ti mismo. También permite importar videos.
   b) Publicar: Permite ver la ejecución de la animación en una Web.
   c) En Edición destacamos:
      - Línea de tiempo: aquí podemos encontrar los comandos básicos para trabajar con la línea de tiempo. Permite cortar, pegar, etc. fotogramas.
      - Preferencias: esta opción será necesaria en cuanto comencemos a incorporar clases ya que aquí se debe poner la ruta donde se buscarán las clases.

   d) En insertar cabe destacar las opciones de la línea de tiempo:
      - Capa: para introducir una nueva capa en nuestra película.
      - Guía de movimiento: más adelante expondremos un ejemplo de esta utilidad.
      - Insertar un fotograma más F5.
      - Insertar un fotograma clave más F6.
      - Insertar un fotograma clave vació más F7.

   Cabe destacar que las más importantes de estas opciones se pueden encontrar en los botones bajo el nombre de las capas en la línea de tiempo.

   e) En Control se tienen los comandos para reproducir la película.
   f) Por último, comentaremos las opciones mas utilizadas del menú ventana:



- Es importante tener en cuenta la biblioteca común ya que ahí vamos a poder encontrar una gran variedad de botones predefinidos.
- En componentes se encuentran objetos como:
    - TextArea: un lugar ideal para cargar texto.
    - TextInput: se utiliza para interactuar con el usuario, lo utilizamos para que el usuario pueda introducir datos.
    - DataGrid: componente típico para crear menús desplegables.

Línea de Tiempo:

La línea de tiempo nos sirve para darle el tiempo necesario a la animación que vamos a diseñar. El sumatorio de frames equivale a los segundos que queremos que dure nuestra animación. De forma predeterminada Flash viene con una velocidad de 12 cuadros por segundo.
Esta configuración se puede cambiar aumentando el número de frames por segundo para dar la sensación de mayor velocidad o disminuirlo para conseguir el efecto contrario.

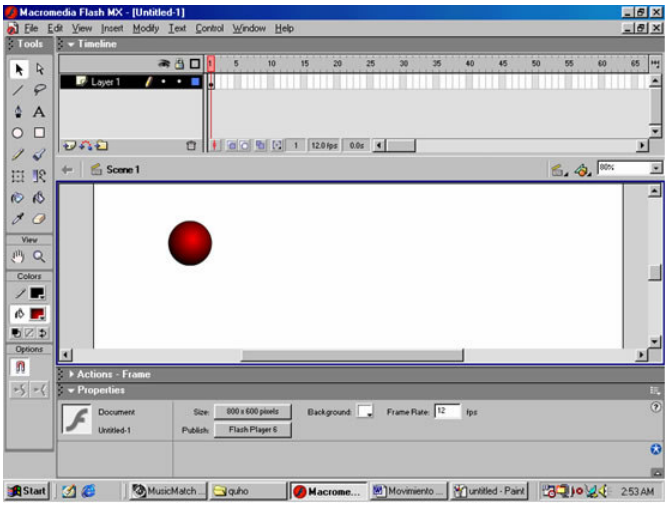
4.1.1.2. Cómo realizar un movimiento básico en Flash

Flash permite realizar dos clases de movimiento las cuales llamaremos frame to frame (cuadro a cuadro), y motion tween (interpolación de movimiento).
A continuación pasaremos a explicar estos dos tipos de movimiento.

- Animación cuadro a cuadro:

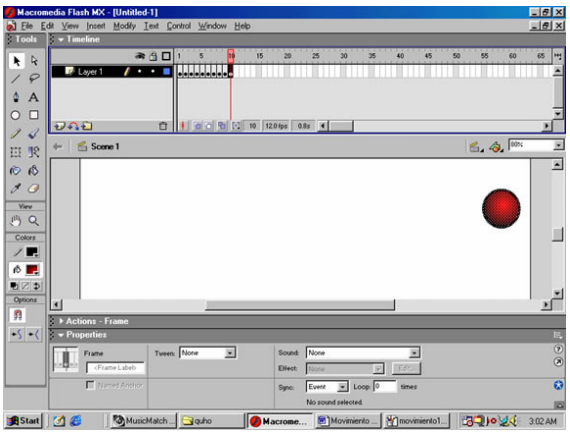
Al utilizar esta técnica, el usuario dibuja todos los cuadros y el movimiento se obtiene al mostrar los cuadros muy rápidamente.

Veamos un ejemplo:
Dibujamos un círculo en la pantalla utilizando la herramienta de óvalo, como se muestra en la siguiente imagen.



A continuación insertaremos un fotograma clave utilizando la línea de tiempo:

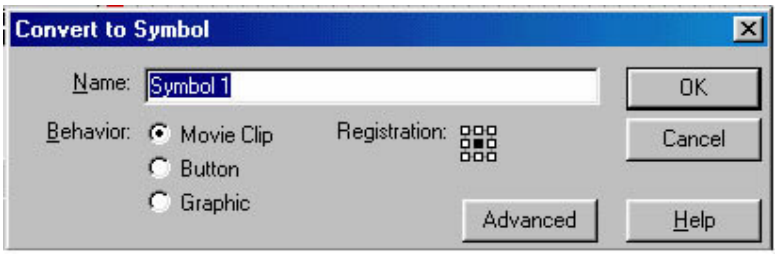
- Seleccionar el fotograma que se acaba de crear haciendo clic en el segundo fotograma clave de la línea de tiempo.
- Mover un poco la bola a la derecha y repetir el proceso hasta que la bola quede al otro lado del escenario.



Ahora ya tenemos nuestro primer movimiento cuadro a cuadro, a continuación lo que hay que hacer es ejecutar la animación.

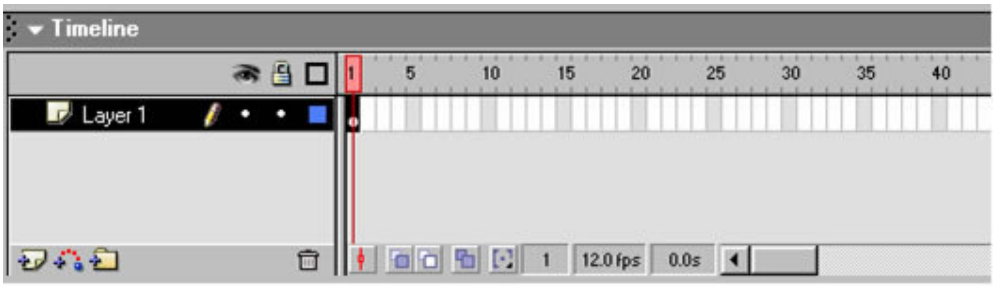
- Interpolación de Movimiento

  La interpolación de movimiento se realiza sobre símbolos, no sobre objetos. Para crear un símbolo existen dos maneras, la primera: ir al menú en la parte de arriba, en la sección de Insertar / Insert, escoger la opción de Convertir en símbolo / Convert to symbol, y la segunda: una vez seleccionado el objeto presionar F8.
  Automáticamente después de haber creado el objeto de cualquiera de las dos formas anteriormente mencionadas veremos la siguiente ventana:

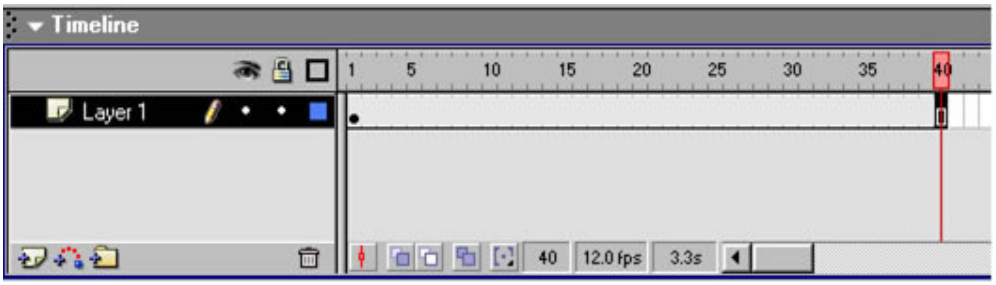


Aquí es donde le daremos el nombre a nuestro símbolo, en el campo de texto de la etiqueta “Name”, colocaremos el nombre de nuestro símbolo, que este caso será “bola”, dejamos el “Behavior” como está, en Movie Clip (de esto dependen las propiedades que va a tener nuestro símbolo, pero eso lo veremos más adelante), “Registration” (sirve para modificar el tamaño de tu símbolo a escala determinando hacia donde va a crecer) también lo dejamos como está, ya que en este caso no lo vamos a utilizar.

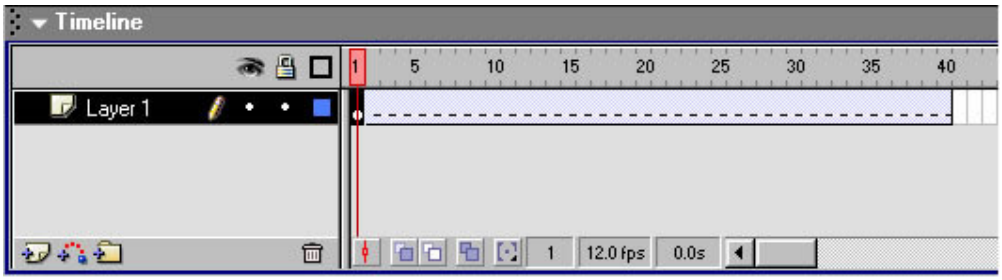
Muy bien, ahora que ya tenemos nuestro símbolo, vamos a ver la interpolación de movimiento:
En este momento tenemos la línea de tiempo de esta manera:



A continuación hacemos clic derecho en el frame (fotograma) número 40 de nuestra línea de tiempo, y en el menú contextual que sale, hacemos clic en la opción de “insert frame (insertar fotograma)”, quedando nuestra línea de tiempo así:

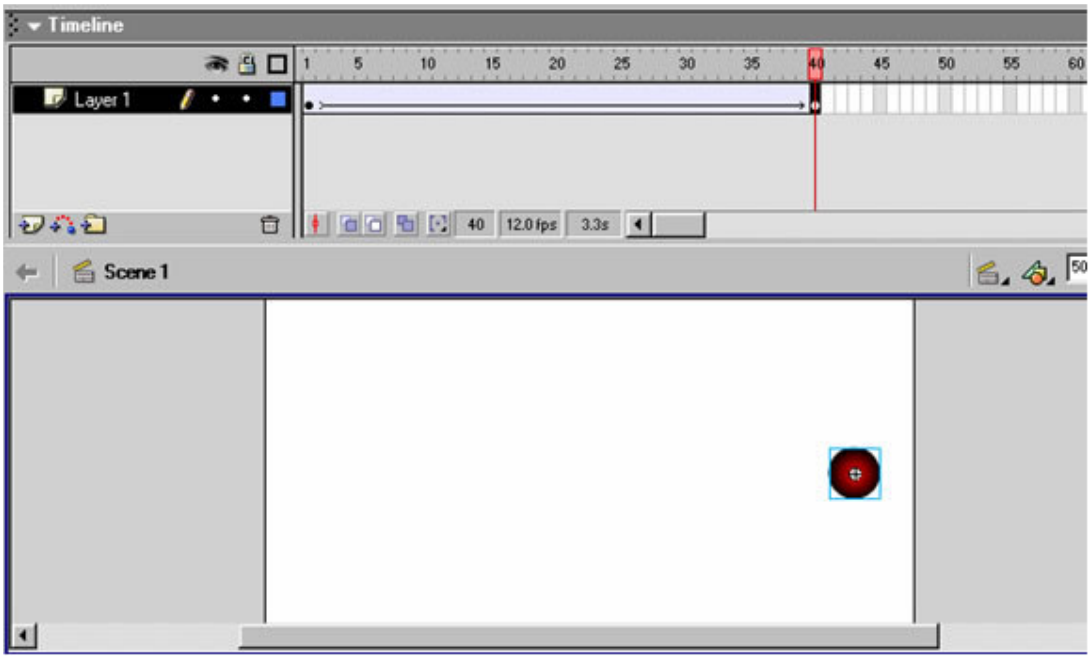


Dirigimos el puntero del ratón al primer frame (fotograma), y hacemos clic derecho sobre él, en el menú contextual que emerge seleccionaremos la opción de “create motion tween” (crear interpolación de movimiento) y la línea del tiempo se rellenará con una línea punteada, así:



Ahora que ya tenemos nuestra interpolación de movimiento, sólo tenemos que indicar una cosa, “el movimiento”, y esto se hace de la siguiente manera:

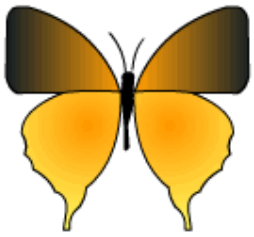
- Hacemos clic en el frame (fotograma) número 40, de modo que quede seleccionado (esto lo sabemos porque queda sombreado, de color negro).
- Ahora nos dirigimos al stage (escenario), y tomamos la bola con el puntero del ratón, la arrastramos hasta el otro extremo del stage (escenario), hasta que quede de esta forma:



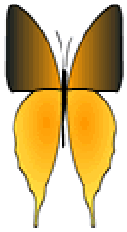
Como se puede ver, en la línea de tiempo se creó una flecha desde el frame (fotograma) número uno, hasta el número 40. Esta flecha indica interpolación de movimiento: nosotros solamente indicamos el principio y el fin de nuestro símbolo, y flash crea todos los frames / fotogramas intermedios.

4.1.1.3. Movimiento básico utilizando una guía

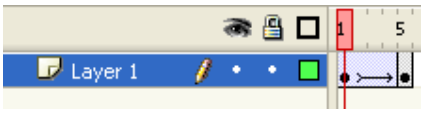
Para este ejemplo vamos a realizar una animación de una mariposa volando, para iniciar dibujamos una mariposa.



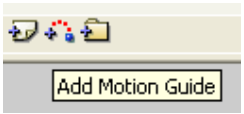
Luego se convierte en un Movie Clip (F8), una vez dentro de éste, se hace una pequeña animación para dar el efecto del movimiento de alas. Para ello insertamos un KeyFrame (fotograma clave) más o menos en el frame 5 (las mariposas baten las alas rápidamente y por eso la animación debe ser corta) y allí cambiamos el tamaño horizontal de la mariposa para que quede de la siguiente forma:



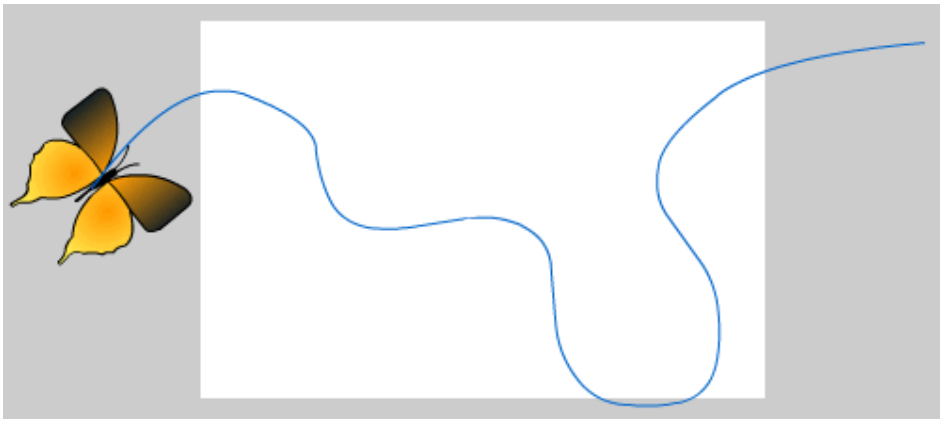
Y la línea del tiempo se ve así:



Después de tener listo el Movie Clip, vamos a la escena, allí lo que hacemos es crear una línea guía para que la mariposa se desplace por ella, esto se logra haciendo clic en el icono de Add Motion Guide ubicado en la parte inferior del panel de los layers.



Una vez allí dibujamos una línea por la cual se desplazará la mariposa.



Para hacer la animación, en el primer frame ubicamos la mariposa al comienzo de la línea, y en el último KeyFrame (fotograma clave), ubicamos la mariposa al final de la línea, finalmente al hacer el Motion Tween y listo.

4.2. *Tutorial ActionScript:*

4.2.1. Cómo obtener interactividad y movimiento con Action Script

Este tutorial pretende explicar de manera sencilla el lenguaje Action Script, siendo a su vez lo más general posible sin centrarse en ninguna aplicación concreta.

ActionScript 1.0 fue el primer lenguaje sobre el que se basó Action Script y aunque tenía un sistema "pseudo" basado en objetos, en realidad no implementaba la POO como debía ser y la hacía un poco abstracta para el desarrollador.
La nueva versión de ActionScript, 2.0, ha cambiado este panorama, rigiéndose con el estándar ECMA (estándar al que se acogen lenguajes como JavaScript o C#) y con un toque de similaridad a Java. Esto ha hecho mucho más fácil a los experimentados saltar de cualquier otro lenguaje a ActionScript y a los principiantes aprenderlo de una manera más sencilla.

4.2.1.1. Programación orientada a objetos con ActionScript 2.0

Action Script tiene bastantes clases definidas, aunque no esperemos encontrar grandes cosas como en JAVA, en ellas podremos encontrar cosas ciertamente útiles como la clase tween que más adelante comentaremos.
Las acepciones de propiedades y miembros que se hacen en este tutorial son las mismas que para cualquier lenguaje orientado a objetos.
En este corto ejemplo sólo usamos clases incluidas dentro de Flash: *TextField*, *MovieClip* y *Date*.
Pasemos a ver cómo se utilizan estas clases. Para ello vamos a ir comentando ejemplos sencillos.

Comencemos con un ejemplo sencillo donde iremos comentando las propiedades, métodos e instanciación de un objeto:

Empecemos con las *propiedades*. Comencemos modificando la propiedad "text" de un campo de texto:

```
area_txt.text = "Programación orientada a objetos";
```

Con esto hemos conseguido poner el campo de texto de la variable área el valor “Programación orientada a objetos”.

Ahora veamos los *métodos*, para esto, vamos a invocar al método "gotoAndPlay" de un Movieclip

```
pelicula_mc.gotoAndPlay (50);
```

Los métodos se manejan de la misma manera que las funciones: en este caso, ésta tenía un parámetro "frame" cuyo valor pasamos como 50. Este método nos permite pasar a ejecutar directamente el frame número 50.

Veamos la instanciación de objetos, vamos a crear un nuevo objeto "Date"

```
var fecha: Date = new Date();
```

Para ello colocamos la palabra “var” a continuación el nombre de la variable, dos puntos y el nombre de la clase de la que esta variable es instancia. Por ultimo, por medio de "new" asigno memoria, diciéndole que cree dentro de sí un objeto "Date". Usamos el "()" como en las funciones para invocar su constructor, concepto que veremos más adelante.

4.2.1.2. 1r Movimiento: Archivos externos .as, primera clase

Lo primero que debemos tener en cuenta es que cuando estemos programando clases, estas deben ir en archivos externos .as, no pueden ser incluidas como parte del código dentro de la película.

Más adelante trataremos el tema de los paquetes de clases y los namespace, por el momento, sólo mantendremos los archivos .as en la misma carpeta de nuestro .fla y .swf.

Las clases se pueden escribir utilizando cualquier editor de texto. Si se quiere hacer desde el mismo editor de Flash, ve al menu File -> New y en el cuadro de diálogo emergente elige "Actionscript File". Aparecerá ante nosotros una vista de código sólo, sin escenario, que será donde podremos programar nuestra clase.

Aquí empezaremos, creando nuestra primera clase, la clase Persona. Hemos utilizado esta clase porque es ideal para comprender los conceptos de herencia, abstracción, etc.

Coloca este código dentro de Flash, en el modo anterior de "Actionscript File" :

```
//Inicio declaración de la clase
class Persona {
  //Variable nombre, miembro de la clase
  var nombre:String;
  //Función constructora;
  function Persona() {
          /*El uso del apuntador "this" indica que nos estamos
          refiriendo a los elementos de la clase. En este caso,
          modificaremos la variable "nombre" mencionada arriba*/
          this.nombre = "";
  }
}
```

Ahora guardemos este archivo como "Persona.as" en una carpeta que usaremos para guardarlo todo.
Importante: El nombre del archivo .as que contenga el código de las clases debe tener el mismo nombre que la clase.
Solamente puede ser declarada *una* clase por archivo.
Ahora, creamos una nueva película, guardémosla con cualquier nombre en la misma carpeta donde colocamos el archivo "Persona.as" y coloquemos este código en el primer keyFrame (fotograma clave).

```
//Palabra clave que trae a nuestro código la clase que creamos
import Persona;
//Creo una nueva variable de tipo "Persona"
var freddie:Persona;
//Asigno memoria a la variable "freddie" y la convierto en una
instancia de la clase "Persona"
freddie = new Persona();
```

Así hemos iniciado la programación orientada a objetos con nuestra primera clase: **Persona**.

4.2.1.3. 2o Movimiento: Miembros y Abstracción

Ahora definiremos los miembros de la clase, que no son más que las funciones (métodos) y variables (propiedades) que integrarán nuestra clase.

Si enumeramos funciones básicas de una persona esta debe

1. Hablar
2. Oír
3. Tocar
4. Mover

Y si hablamos de sus propiedades básicas

1. Nombre
2. Color de Ojos
3. Color de Piel

Se que dejamos muchísimas fuera, pero esto es en pro de hacer mas sencillo el ejemplo.
Así que, las funciones básicas serán métodos de nuestra clase y las propiedades lo mismo.

Vamos a modificar de esta manera nuestro código

```
//Inicio declaración de la clase
class Persona {
        /* PROPIEDADES */
        //Variable nombre, miembro de la clase
        var nombre:String;
        var colorOjos:String;
        var colorPiel:String;
        //Función constructora;
        function Persona() {
                this.nombre = "";
                //Esto simplemente para avisar en la ventana del output
                que la clase fue creada
                trace("Una nueva Persona ha nacido");
        }
        /* METODOS */
        //Método "hablar", recibe por parámetro la frase que dirá
        //devuelve una frase procesada
        function hablar(frase:String):String {

                var mensaje:String = this.nombre+" dice: "+frase;
                //Esta frase aparecerá en el Output
                trace(mensaje);
                //Y así mismo, la función la retornara
                return mensaje;
        }
        //Método "ver"
        function ver():Void {
                trace(this.nombre+" esta mirando alrededor");
        }
        //Método "tocar"; recibe por parámetro, otra persona para tocar
        function tocar(personaTocada:Persona):Void {
```

```
                trace(this.nombre+" ha tocado a
"+personaTocada.nombre);
        }
        //Método "mover"; recibe por parámetro el lugar a moverse
        function mover(lugar:String):Void {
                trace(this.nombre+" se mueve a "+lugar);
        }
}
```

Hagamos que dos "Personas" interactúen, modifiquen el código de la película .FLA a esto:

```
//Trae a nuestro código la clase que creamos; nótese que no he
colocado la extensión ".as"
import Persona;
//Creo una nueva variable llamada "freddie" de tipo "Persona"
var freddie:Persona;
//Asigno memoria a la variable "freddie" y la convierto en una
instancia de la clase "Persona"
freddie = new Persona();
//Creo otra persona
var sandra:Persona = new Persona();
//Les asigno nombres
freddie.nombre = "Freddie®";
sandra.nombre = "Shala";
//Empezamos a llamar a sus métodos
freddie.mover("donde esta la otra persona");
sandra.ver();
freddie.hablar("Hola, ¿como te llamas?");
sandra.hablar(sandra.nombre+" y tu?");
freddie.hablar("Yo soy "+freddie.nombre);
freddie.ver();
freddie.tocar(sandra);
```

Lo que mostrara la siguiente salida en la ventana del **trace**:

```
Una nueva Persona ha nacido
Una nueva Persona ha nacido
Freddie® se mueve a donde esta la otra persona
Shala esta mirando alrededor
Freddie® dice: Hola, ¿como te llamas?
Shala dice: Shala y tu?
Freddie® dice: Yo soy Freddie®
Freddie® esta mirando alrededor
Freddie® ha tocado a Shala
```

Aquí se empieza a ver que estas clases podrían ser útiles (aparte de realizarlas por diversión)

Nota: Esto podría aplicarse a la creación de TADs por ejemplo para crear una clase bola que se pudiese mover por el escenario.

4.2.1.4. 3r Movimiento: Encapsulación, getters y setters

En nuestro movimiento anterior creamos propiedades y métodos "miembro" de la clase, como el color de ojos, hablar, ver, etc. Ahora vamos a encapsular todos estos datos, algo esencial en la programación.

La encapsulación nos permitirá esconder del "mundo exterior" de la clase las variables y funciones internas que no llamaremos desde una instancia. De hecho, la teoría dice que nosotros *no debemos* manipular directamente las propiedades (variables) de una clase. Para ello usaremos un tipo de funciones llamadas *getters* y *setters*.
Nota: Los *getters* y *setters* son funciones que nos permiten manipular la asignación y llamado de variables dentro de nuestro programa. Como su nombre indica son funciones para asignar un dato a una variable (**set)** o para obtener el dato de una variable (**get)**. Una vez creadas, cada vez que asignemos un dato a una variable será llamada su función **set** y cuando pidamos el dato de una variable veremos su **get**.

Así que lo próximo que vamos a hacer es:

1. Hacer públicas las funciones o propiedades a las que accederemos desde el exterior.
2. Hacer privadas las funciones o miembros que no queremos/debemos acceder desde el exterior.
3. Crear funciones getter/setter para las propiedades que ahora son privadas y queremos modificar.

```
//Inicio declaración de la clase
class Persona {
        /* PROPIEDADES */
        //Variable nombre, miembro de la clase
        public var nombre: String;
        private var colorOjos:String;
        private var colorPiel:String;
        //Función constructora
        function Persona() {
                this.nombre = "";
                //Esto para avisar el output que la clase fue creada
                trace("Una nueva Persona ha nacido");
        }
        /* METODOS */
        //Método "hablar", recibe por parámetro la frase que dirá
        //devuelve una frase procesada
        public function hablar(frase:String):String {

                var mensaje:String = this.nombre+" dice: "+frase;
                //Esta frase aparecerá en el Output
                trace(mensaje);
                //Y así mismo, la función la retornara
                return mensaje;
        }
        //Método "ver"
```

```
        public function ver():Void {
                trace(this.nombre+" esta mirando alrededor");
        }
        //Método "tocar"; recibe por parámetro, otra persona para tocar
        public function tocar(personaTocada:Persona):Void {
                trace(this.nombre+" ha tocado a
                "+personaTocada.nombre);
        }
        //Método "mover"; recibe por parámetro el lugar a moverse
        public function mover (lugar:String):Void {
                trace(this.nombre+" se mueve a "+lugar);
        }
        //GETTERS Y SETTERS (ACCESORES Y MUTADORES)
        //Funciones "get" y "set" para el color de ojos;
        Public function set ojos (color:String):Void {
                trace(this.nombre+" tiene ojos de color "+color);
                this.colorOjos = color;
        }
        public function get ojos():String {
                return this.colorOjos;
        }
        //Funciones "get" y "set" para el color de piel
        public function set piel(color:String):Void {
                trace(this.nombre+" tiene piel "+color);
                this.colorPiel = color;
        }
        public function get piel():String {
                return this.colorPiel;
        }
}
```

Como se puede observar se han usado las palabras clave **public** y **private** del lenguaje ActionScript 2.0 para declarar algunas funciones, variables públicas y otras privadas. Las privadas solo podrán ser llamadas desde el código dentro de la clase, mientras que las públicas se podrán llamar desde cualquier lugar del código.

NOTA: toda función **set** debe asignarle un valor a alguna variable miembro de la clase y toda función **get** debe devolver algún valor.

Para ver un poco más la utilidad de lo que acabamos de hacer y hacer uso de los getters y setters vamos a modificar y reescribir algunas partes del código en la película .FLA:

```
import Persona;
//Creo una nueva variable llamada "freddie" de tipo "Persona"
var freddie:Persona;
freddie = new Persona();
//Creo otra persona
var sandra:Persona = new Persona();
//Les asigno nombres
freddie.nombre = "Freddie";
sandra.nombre = "Shala";
//Les asigno color de ojos y piel
freddie.ojos = "negro";
sandra.ojos = "azul";
freddie.piel = "blanca";
sandra.piel = "morena";
```

```
//Empezamos a llamar a sus métodos
freddie.mover("donde esta la otra persona");
sandra.ver();
freddie.hablar("Hola, ¿como te llamas?");
sandra.hablar(sandra.nombre+" y tu?");
freddie.hablar("Yo soy "+freddie.nombre);
freddie.ver();
freddie.hablar("Me gustan tus ojos color "+sandra.ojos+" y tu piel
"+sandra.piel);
sandra.hablar("Gracias ");
freddie.tocar(sandra);
sandra.hablar("...");
freddie.hablar("...");
```

Lo que nos dará también un cambio en la ventana del trace:

```
Una nueva Persona ha nacido
Una nueva Persona ha nacido
Freddie tiene ojos de color negro
Shala tiene ojos de color azul
Freddie tiene piel blanca
Shala tiene piel morena
Freddie se mueve a donde esta la otra persona
Shala esta mirando alrededor
Freddie dice: Hola, ¿como te llamas?
Shala dice: Shala y tu?
Freddie dice: Yo soy Freddie®
Freddie esta mirando alrededor
Freddie dice: Me gustan tus ojos color azul y tu piel morena
Shala dice: Gracias
Freddie ha tocado a Shala
Shala dice: ...
Freddie dice: ..
```

4.2.1.5. 4o movimiento: Clases abstractas, Herencia y súper

Para explicar la herencia hemos pensado en un ordenador, hay dos tipos de ordenadores los pc's y los Mac.

Con Herencia, podemos hacer que Ordenador sea una clase abstracta (entiéndase por clase abstracta una clase que no se puede instanciar, solo heredar) y crear dos nuevas clases, *Mac* y *PC*, que hereden de ordenador todas sus propiedades y métodos e implementen ellas mismas los suyos propios. ¿Por qué es importante que Ordenador sea una clase abstracta? Dado que ordenador tiene los métodos y propiedades genéricas, pero no los específicos, si un programador crea una instancia de la clase Ordenador sería un poco inútil y se saltaría toda la teoría de la encapsulación del código.

Implementemos las tres clases, una clase "Computador" en un archivo Computador.as, una clase "PC" en PC.as y una clase Mac en un archivo Mac.as. Todas en la misma carpeta junto a un archivo FLA de cualquier nombre donde instanciaremos los objetos. Iniciemos con el archivo Computador.as :

```
//Esta será nuestra clase Computador, de la que heredaran las otras
//dos
class Computador {
        /*Función constructora; este constructor se ejecutara primero
        que el constructor de la clase heredada*/
        function Computador() {
                trace("Creado un nuevo computador");
        }
        public function encender():Void {
                trace("El computador esta iniciando");
        }
        public function apagar():Void {
                trace("El computador ha sido apagado");
        }
}
```

Ahora el archivo PC.as

```
//La palabra clave "extends" es la que le indica a la clase que
//herede todas sus propiedades y métodos de una superclase, en
//este caso "Computador"
class PC extends Computador {
        //Esta propiedad nos devolverá el sistema operativo
        private var SO:String;
        //Constructor del PC
        function PC () {
                trace("Has creado un PC");
        }
        //Función setter para colocar un sistema operativo
        public function set sistema(SSOO:String):Void {
                this.SO = SSOO;
        }
        //Función que me dirá que sistema tengo instalado
        public function sistemaInstalado(){
                if(this.SO == "windows"){
                        trace("Tienes instalado Windows X_X");
```

```
                } else if (this.SO == "linux"){
                        trace("Tienes instalado Linux ^_^");
                } else {
                        trace("No se que tienes instalado O_o");
                }
        }
}
```

Luego nuestra clase Mac en *Mac.as*

```
//Aquí es lo mismo, solo que en la clase Mac
class Mac extends Computador {
        //Constructor del PC
        function Mac() {
                trace("Has creado una Mac :-)");
        }
        //Como diferencia al PC, aquí instalamos Mac
        public function instalarMac():Void {
                trace("Has instalado MacOSX");
        }
        //Las Mac tienen "Rendezvous" que les permite configurarse
solas
        //en cualquier entorno de red; demosle esa capacidad
        public function rendezvous():Void {
                trace("Detectando redes cercanas ...");
                trace("Red configurada ^^");
        }
        //Y una funcioncilla para navegar con Safari alguna web
        public function safari(URL:String):Void {
                trace("Estas navegando "+URL);
        }
}
```

Por último, en el archivo FLA vamos a "jugar" creando una *Mac* y una *PC*, haciendo cosas que hacen ambas (por medio de la herencia de la clase *Computador*) y haciendo cosas que solo están implementadas y son exclusivas de cada clase:

```
//Importo las clases PC y Mac, estas al heredar de la clase Computador
//importan automáticamente la misma
import PC;
import Mac;
//Iniciemos con el PC
//Le asigno memoria,
var Sunny:PC = new PC();
//Llamo a una función de la clase Computador que hereda la clase PC
Sunny.encender();
//Llamo a un setter miembro de la clase PC nada mas
Sunny.sistema = "linux";
//Llamo a un método propio de la clase PC
Sunny.sistemaInstalado();
//Llamo al método apagar, de la clase heredada Computador
Sunny.apagar();
/**/
//Sigamos con el Mac
trace("**** BLUESHOCK *****");
//Le asigno el tipo "Mac" y creo la instancia de la clase con new
var Blueshock:Mac = new Mac();
//Llamo a un método de la clase Computador, heredado a Mac, por ende
//aparece en la clase Mac y PC
Blueshock.encender();
//Llamo al método propio de Mac "instalarMac", este no aparece en la
//clase PC
Blueshock.instalarMac();
```

```
//Otro método miembro de la clase Mac
Blueshock.rendezvous();
//Llamo al metodo Apagar, heredado de "Computador"
Blueshock.apagar();
```

Al pulsar CONTROL+ENTER en nuestra película encontraremos que el Output nos muestra el proceso donde se evidencia que cada vez que hacemos un **new Mac()** o un **new PC()** se ejecuta primero el constructor de la clase *Computador*, y luego el de la clase que lo heredó.

```
Creado un nuevo computador
Has creado un PC
El computador esta iniciando
Tienes instalado Linux ^_^
El computador ha sido apagado
**** BLUESHOCK *****
Creado un nuevo computador
Has creado una Mac
El computador esta iniciando
Has instalado MacOSX
Detectando redes cercanas ...
Red configurada ^^
El computador ha sido apagado
```

Así de una manera sencilla hemos implementado herencia en nuestras clases de Flash.

MX 2004.
Estos componentes, como se puede ver, tienen similitudes, por ejemplo, casi todos tienen el evento *change* o *click* y en todos hace lo mismo y se implementa de la misma manera. Todos tienen el evento *setStyle* que permite cambiar el estilo del componente y así la mayoría tiene cosas similares. Esto es porque todos los componentes heredan de una superclase llamada *UIObject* que contiene las propiedades y métodos básicos para todos los componentes, así no tenemos que duplicar esfuerzos recreando cada propiedad o método común, solo heredamos.

Nota: Una SuperClase es el nombre común que se le da a las clases abstractas o clases que son heredadas por otras. En este ejemplo, las clases normales eran **PC** y **Mac** y la SuperClase era **Computador .**

4.2.1.6. 5o Movimiento: Interfaces

Una interfaz es, básicamente, una declaración de propiedades y métodos de una clase ¡¡**sin código**!!, es decir, una clase que solo tiene los nombres, los tipos, los parámetros de los métodos y las propiedades, pero no las implementa. Es muy útil para establecer reglas en equipos de trabajo y similar a la herencia.

Esta vez usaremos animales para nuestro ejemplo, un pato y un gato. Todos los animales comen, pero el gato come con el hocico y el pato con el pico. Al ser procesos de igual nombre pero distintos, lo ideal sería una interfaz que implemente el método comer y luego ellos "implementen esa interfaz" y le pongan su código respectivo. Así mismo ambos caminan, pero uno lo hace con dos patas y el otro lo hace con cuatro.

Lo primero que haremos entonces será crear la interfaz de Animal;

```
//Declaramos nuestra interface "Animal"
interface Animal {
	//Como ven, se declara la función "comer", con un parámetro
	//"comida" de tipo "String" y es una función "Void" que no
	//retorna nada, pero
	//No se coloca código alguno en la función, solo se declara
	function comer (comida:String):Void;
	function caminar(destino:String):Void;
}
```

Ahora vamos a crear las clases que implementaran esa interface, empezando por *Gato.as* :

```
//La palabra clave "implements" es la que me permite asignarle una
//interface a una clase
class Gato implements Animal {
	//El constructor de toda la vida
	function Gato() {
		trace("miau");
	}
	//Ahora la implementación de las funciones de la interface
	//Debe escribirse exactamente igual a como esta en la interfaz
	public function comer(comida:String):Void {
		trace("El gato, con su hocico esta comiendo "+comida);
	}
	//Igual la de caminar
	public function caminar(destino:String):Void {
		trace("El gato, camina en sus cuatro patas a
"+destino);
	}
}
```

Pasemos a la clase *Pato*, en *Pato.as*

```
class Pato implements Animal {
	//Lo mismo, un constructor para el pato
	function Pato() {
		trace("quack!");
	}
	//De nuevo implementamos los métodos descritos en la interface
	//Aquí con un código distinto
	public function comer(comida:String):Void {
		trace("El pato come "+comida+" usando su pico");
```

```
        }
        //Igual la de caminar
        public function caminar(destino:String):Void {
                trace("Hacia "+destino+" camino en dos patas el pato;
lol");
        }
}
```

Para terminar, como siempre, creamos un FLA en la misma carpeta de los .AS, que es donde crearemos las variables y el código que le da utilidad a nuestras clases, en él colocamos (Primer keyFrame –punto clave):

```
//Importo las clases Gato y Pato, que implementan la interface Animal
import Gato;
import Pato;
//Les asigno tipo de objeto y memoria
var garfield:Gato = new Gato();
var lucas:Pato = new Pato();
//Llamo a la función comer, declarada en la interface y escrita en la
clase
garfield.comer("lasaña");
//Igual con la de caminar
garfield.caminar("el sofá");
//Si notan, se llama igual, pero su código y función es diferente
lucas.comer("maíz");
lucas.caminar("el fin del mundo");
```

Y si pulsamos CONTROL+ENTER, podemos ver su comportamiento reflejado en el Output. Las funciones fueron implementadas con mismo nombre, tipo y argumentos, pero con distinto código que genera resultados diferentes :

```
miau
quack!
El gato, con su hocico esta comiendo lasaña
El gato, camina en sus cuatro patas a el sofá
El pato come maíz usando su pico
Hacia el fin del mundo camino en dos patas el pato
```

Así hemos implementado un sistema simple de interfaces entre dos clases.

La manera tradicional de asociar, vincular o hacer que un MovieClip herede las propiedades de una clase que hemos hecho, es a través de la vinculación en el panel biblioteca. En el campo Clase de AS 2.0: deberíamos colocar el nombre de la clase.
Pero hay una manera más práctica y más limpia que permite muchas más posibilidades. Es con el siguiente código:

```
Object.registerClass(”miClip”, miClase);
```

De esta manera tenemos vinculado un MovieClip a una clase. Esta clase hay que extenderla a la clase MovieClip de esta manera:

```
class miClase extends MovieClip
```

Conclusiones

Este tutorial trata de explicar cómo realizar animaciones de forma que el lector aprenda a crearlas él mismo, en ningún caso hemos pretendido explicar exactamente cómo han sido diseñadas las nuestras. Hemos tratado de explicar la mayor parte de aspectos técnicos de la implementación de clases en ActionScript.

4.2.1.7. Tutorial de la clase tween

Lejos quedan ya las versiones de Flash en las que para mover un objeto uno tenía que hacerlo mediante interpolaciones de movimiento entre dos situaciones distintas en fotogramas distintos. Ahora podemos hacer uso de la clase Tween.
La clase Tween permite mover, cambiar el tamaño y desvanecer un clip de película fácilmente en el escenario. La sintaxis es:

```
function Tween(obj, prop, func, begin, finish, duration, useSeconds) {
// código...
}
```

Donde:

- **obj** es el objeto de clip de película al que se refiere la instancia de Tween.
- **prop** es un nombre de cadena de una propiedad de obj cuyos valores se interpolarán.
- **func** es el método de suavizado que calcula un efecto de suavizado para los valores de propiedad del objeto interpolado. Es decir, lo que conocíamos por 'easing'.
- **begin** es un número que indica el valor inicial de prop (propiedad del objeto de destino que se interpolará).
- **finish** es un número que indica el valor final de prop (propiedad del objeto de destino que se interpolará).
- **duration** es un número que indica la duración del movimiento de interpolación. Si se omite, se establece la duración en infinito de forma predeterminada.
- **useSeconds** es un valor booleano relacionado con el valor especificado en el parámetro *duration*, que indica que se deben utilizar segundos si se establece como *true* o fotogramas si se establece como *false*.

La clase Tween está en el paquete the mx.transitions, por lo que lo más habitual es que tengamos que importarla antes de usarla:

```
import mx.transitions.Tween;
```

Por ejemplo, el siguiente código mueve un clip denominado "miClip" de 0 a 550 en el eje x en 10 segundos:

```
import mx.transitions.Tween;
var twMoveClip: Tween = new Tween(miClip, "_x", null, 0, 550,
10, true);
```

Para especificar una función de 'easing', deberemos especificar cuál es su nombre. Dentro de este paquete disponemos de 6 métodos de suavizado o 'easing':

- **Back:** Extiende la animación cuando se supera el límite de la transición en uno o ambos extremos para producir el efecto de ser empujado hacia atrás desde el límite.
- **Bounce:** Añade un efecto de rebote dentro del rango de la transición en uno o ambos extremos. El número de rebotes depende de la duración: cuanto mayor sea la duración, producirá un mayor número de rebotes.
- **Elastic:** Añade un efecto elástico que está fuera del rango de la transición en uno o ambos extremos. La duración no afecta al grado de elasticidad.
- **Regular:** Añade un movimiento más lento en uno o ambos extremos. Esta función permite añadir un efecto de aceleración, desaceleración o ambos.

- **Strong:** Añade un movimiento más lento en uno o ambos extremos. Este efecto es similar al suavizado Regular, pero más pronunciado.
- **None:** Añade un movimiento uniforme de principio a fin sin efectos, ralentización ni aceleración. Esta transición también se denomina transición lineal.

Cada una de estas seis clases de cálculo de suavizado tiene tres métodos de suavizado que indican la parte de la animación en la que se aplicará el efecto de suavizado:

- **easeIn :** Proporciona el efecto de suavizado al principio de la transición.
- **easeOut :** Proporciona el efecto de suavizado al final de la transición.
- **easeInOut :** Proporciona el efecto de suavizado al principio y al final de la transición.
- **easeNone :** Indica que no se va a utilizar ningún cálculo de suavizado. Sólo se proporciona en la clase de suavizado None.

Ejemplo:

```
1. import mx.transitions.Tween;
2. import mx.transitions.easing.*;
3. miClip.onPress = function():Void {
4.      var twMove:Tween = new Tween(this, "_x", Back.easeOut, 0,
   350, 2, true);
5. };
```

Asociados a la clase Tween existen diversos métodos que podemos utilizar:

- **Tween.yoyo ():** Indica a la animación interpolada que continúe desde su valor actual hasta un nuevo valor.
- **Tween.fforward ():** Avanza la animación interpolada directamente al final de la animación.
- **Tween.nextFrame ():** Avanza la animación interpolada al siguiente fotograma.
- **Tween.prevFrame ():** Dirige la animación interpolada al fotograma anterior al actual.
- **Tween.resume ():** Reanuda una animación interpolada a partir del punto en que se ha detenido en la animación.
- **Tween.rewind ():** Retrocede una animación interpolada hasta el principio de la misma.
- **Tween.start ():** Inicia la animación interpolada desde el principio.
- **Tween.stop ():** Detiene la animación interpolada en su posición actual.
- **Tween.toString ():** Devuelve el nombre de clase, "[Tween]".
- **Tween.yoyo ():** Ordena a la animación interpolada que se reproduzca a la inversa desde el último sentido de incrementos de la propiedad de interpolación.

4.2.1.8. **Action Script con Java Script**

Este tutorial esta pensado para futuras ampliaciones del proyecto, ya que creemos que podría ser interesante.
Muchos de los que no solo programamos en Flash sino en otros lenguajes como Visual Basic o aquellos diseñadores y desarrolladores Web, han querido alguna vez que sus scripts se comuniquen e intercambien datos con Flash.
Flash se ha ido desarrollando en el tiempo volviéndose muy dinámico; enriqueciendo su ActionScript, con conexiones a objetos XML, carga de variables por URL y por envío GET y POST, integración con lenguajes dinámicos y muchísimas cosas mas
Pero existe algo que fue desarrollado desde que se publico en Internet el plug-in de Flash, con lo cual podemos intercambiar información entre Flash y otros lenguajes de una manera muy sencilla y ágil, las funciones del Componente ActiveX Shockwave Flash
La clave solo esta en una función de Flash... el FSCommand
Iniciemos con la integración en paginas HTML de Flash
El primer paso es tener una pagina Web HTML lista con el código estándar para colocar una película en nuestra pagina Web.
Cree una película de flash con un botón, y a ese botón asígnele el siguiente código:

```
on (release) {
fscommand("comando", "argumentos");
}
```

Guarde la película con el nombre "Pelicula.fla"
Luego vaya al Menú File>Publish Settings y ahí, escoja publicar Flash y HTML y clickee el botón "Publish", se generaran los archivos SWF de la película y el archivo HTML para ponerlo en la web, el código generado del archivo HTML será así:

```
<HTML>
<HEAD>
<TITLE>Pelicula</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/
shockwave/cabs/flash/swflash.cab
#version=6,0,0,0" WIDTH="550" HEIGHT="400" id="Pelicula" ALIGN="">
<PARAM NAME=movie VALUE="Pelicula.swf"> <PARAM NAME=menu VALUE=false> <PARAM NAME=quality
VALUE=best> <PARAM NAME=wmode VALUE=transparent> <PARAM NAME=bgcolor VALUE=#FFFFFF>
<EMBED src="Pelicula.swf" menu=false quality=best wmode=transparent bgcolor=#FFFFFF WIDTH="550"
HEIGHT="400" NAME="Pelicula" ALIGN=""
TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer"></EMBED>
</OBJECT>
</BODY>
</HTML>
```

Este es el código normal para la incrustación d una película, resalto el pedazo en el que encontramos en la etiqueta OBJECT el atributo ID con el valor "Película", este atributo será el que nos sirva para manipular la película mas adelante en nuestro código.
Ahora lo interesante, hacer que Flash y JavaScript se integren y compartan datos
Vamos a incluir el primer código de JavaScript necesario para que nuestra película funcione; incluyan este código después de que se la etiqueta BODY

```
<SCRIPT LANGUAGE="JavaScript">
<!--
var InternetExplorer = navigator.appName.indexOf("Microsoft") != -1;
//Maneja todos los mensajes de FSCommands en una película
function Pelicula_DoFSCommand(command, args) {
var PeliculaObj = InternetExplorer ? Pelicula: document.Pelicula;
//Aquí va a ir nuestro código de manejo
}

if (navigator.appName && navigator.appName.indexOf ("Microsoft") != -1 &&
navigator.userAgent.indexOf("Windows") != -1 && navigator.userAgent.indexOf("Windows 3.1") == -1)
{
document.write ('<SCRIPT LANGUAGE=VBScript\> \n');
document.write ('on error resume next \n');
document.write ('Sub Pelicula_FSCommand (ByVal command, ByVal args)\n');
document.write (' call Pelicula_DoFSCommand (command, args)\n');
document.write ('end sub\n');
document.write ('</SCRIPT\> \n');
}
!-->
</SCRIPT>
```

Bien, ahora miremos que es lo que hace nuestro código paso a paso.
La línea 1 declara una variable llamada Internet Explorer donde se almacenara un valor true o false, dependiendo si la película se esta o no ejecutando en Internet Explorer
La línea 3 declara la función que va a manejar los eventos de nuestra película, aquí parare un momento a explicar lo que desencadena los eventos de la película
Un evento es la forma en la que puede reaccionar un programa a una acción realizada por el usuario; en nuestro caso, esta función se llama Pelicula_DoFSCommand(command, args) esta función se desencadenara cada vez que un FSCommand sea generado dentro de nuestra película, vemos que la función inicia con el nombre "Película" porque ese es el nombre que se le asigno en el atributo ID de la etiqueta OBJECT; el evento se llama "DOFSCommand" que es cuando ocurre un FSCommand en nuestra película, esta función tiene como argumentos (command, args) que son los dos valores que puede tomar la instrucción fscommand en nuestra película, y estos son los valores que manipularemos mas tarde.
La línea 4 es por compatibilidad, lo que hace es que se pueda manejar de igual manera la película así el navegador sea Internet Explorer y Netscape, de la siguiente manera
Si el navegador es Internet Explorer
PeliculaObj=Película
Ya que el ID de la etiqueta OBJECT es Película, para Internet Explorer se puede manejar así
Si no es Internet Explorer
PeliculaObj=document.Pelicula
Ya que esto funciona en todos los navegadores y no habría que preocuparnos por que navegador esta viendo la pagina, siempre manejaremos el objeto por medio de la variable PeliculaObj
Las líneas 8 a la 16 son líneas de compatibilidad para que el manejador de eventos funcione en Internet Explorer, este gestiona los eventos de diferente manera que el resto de los navegadores, ya que para eso creo el Visual Basic Script, lo que hace ese código es que en caso de que el navegador sea el de Microsoft, escribirá en el documento las líneas necesarias para que VBScript gestione los FSComands, y le pase el control al código JavaScript de arriba.

Bien, ya estamos listos para hacer algo interesante con ese código y poder comprobar si existe una comunicación con Flash y JavaScript
Vamos a hacer que aparezca un mensaje de alerta al usuario al darle clic a un botón de Flash, dejemos como quedo el código de nuestro botón en la película y modifiquemos esto en el código HTML

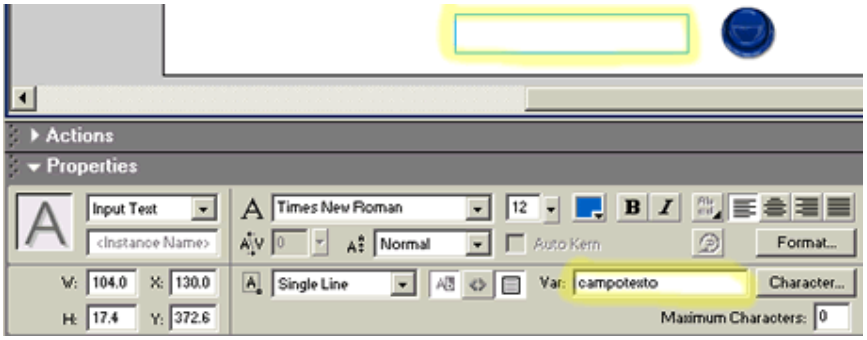
```
<SCRIPT LANGUAGE="JavaScript">
<!--
var InternetExplorer = navigator.appName.indexOf("Microsoft") != -1;
//Maneja todos los mensajes de FSCommands en una película
function Pelicula_DoFSCommand (command, args) {>
var PeliculaObj = InternetExplorer? Pelicula: document.Pelicula;
//Aquí va a ir nuestro código de manejo
alert("El comando pasado fue : "+command+" y el argumento fue : "+args);
}

if (navigator.appName && navigator.appName.indexOf("Microsoft") != -1 &&
navigator.userAgent.indexOf("Windows") != -1 && navigator.userAgent.indexOf("Windows 3.1") == -1)
{
document.write ('<SCRIPT LANGUAGE=VBScript\> \n');
document.write ('on error resume next \n');
document.write ('Sub Pelicula_FSCommand (ByVal command, ByVal args)\n');
document.write (' call Pelicula_DoFSCommand (command, args)\n');
document.write ('end sub\n');
document.write ('</SCRIPT\> \n');
}
</SCRIPT>
```

El código que fue añadido en la línea 6 genera simplemente mensaje para el usuario que le informa que valores fueron pasados por el comando FSCommand en nuestra película
Este código siempre será ejecutado cuando en la película se ejecute una instrucción fscommand.
De esta manera es posible enviarle información de todo tipo a JavaScript desde flash para que este las procese y se las devuelva procesadas... pero ... y ¿como hacemos para que JavaScript le envíe información a Flash?
Para esto deberemos usar una de los métodos de objeto ActiveX Shockwave Flash. El de asignación de variables desde afuera de la película, de la siguiente manera:

Pelicula.SetVariable (‘variable’, valor);

Película: Es el nombre del objeto Película asignado, recuerden que por compatibilidad deben usar en este caso la variable declarada en la función como "PeliculaObj"
SetVariable: Es la instrucción que asigna un valor a una variable en la película, en caso de que la variable no haya sido declarada aun dentro de la película, esta instrucción la creara.
Recibe dos parámetros esta instrucción
Variable: Es el nombre de la variable, debe ir entre comillas.
Valor: Es el valor que le será asignado a la variable; puede ser de tipo String (Cadena o texto) o de tipo Numérico.
Vamos a hacer un corto ejemplo para comprobar que esta instrucción funciona...
En la película vamos a insertar un campo de texto de tipo "Input Text", al cual le vamos a asignar la variable "campotexto" de esta manera:



Como ven en la grafica, existe un campo de texto al que le fue asignado la variable "campotexto"; también esta el botón que habíamos creado antes que contiene la instrucción:

```
on (release) {
fscommand ("comando", "argumentos");
}
```

Mas adelante cambiaremos esa instrucción.

Ahora, vamos a hacer que un botón nos cambie el contenido de ese campo de texto, mediante HTML y JavaScript

En el código HTML, dentro de la etiqueta BODY inserte este código:

```
<P><INPUT id ="texto"><INPUT id ="boton"type=button value ="Click Aqui" OnClick
="javascript: ValorEnTexto (texto.value) ;"></ P>
```

Y luego, en la etiqueta HEAD inserte este pequeño JavaScript

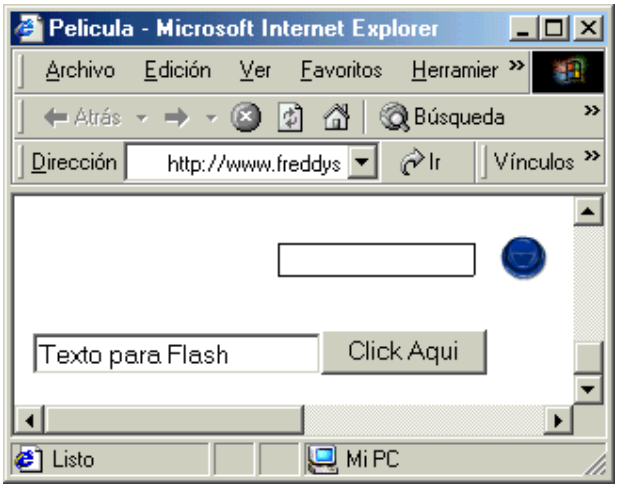
```
<SCRIPT LANGUAGE="JavaScript">
<!--Esconde el código de los navegadores que no lo entienden
function ValorEnTexto (valor){
var PeliculaObj = InternetExplorer? Pelicula: document.Pelicula;
PeliculaObj.SetVariable ("campotexto",valor);
}
-->
</SCRIPT>
```

Este ultimo pedazo de código JavaScript lo que hace es pasarle al campo de texto de Flash el valor almacenado en el campo de texto de HTML.

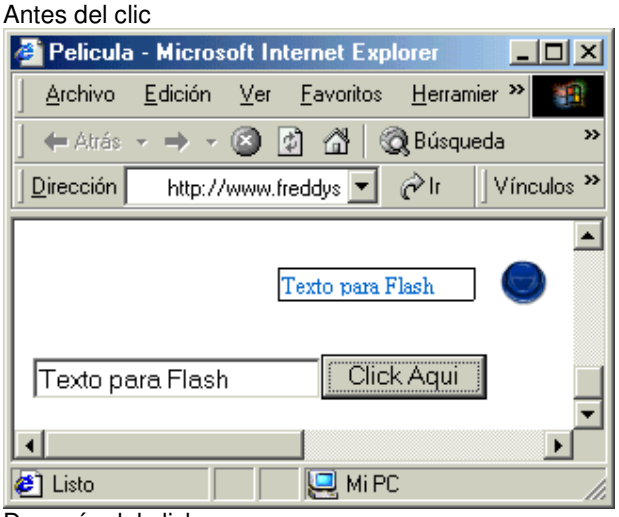
Utilizando el método del objeto Shockwave Flash "SetVariable".

Observen con cuidado que en esta función también declaramos la variable PeliculaObj, recuerden que esta es la manera en la que nos podrá funcionar tanto en Internet Explorer como en Netscape

Prueben la página en el navegador y verán algo así:



Antes del clic



Después del click

En este punto, el código completo HTML de su página debe parecerse a esto :

```
<HTML>
<HEAD>
<TITLE>Pelicula</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var InternetExplorer = navigator.appName.indexOf("Microsoft") != -1;
//Maneja todos los mensajes de FSCommands en una película
function Pelicula_DoFSCommand(command, args) {
var PeliculaObj = InternetExplorer? Pelicula : document.Pelicula;
//Aquí va a ir nuestro código de manejo
alert ("El comando pasado fue : "+command+" y el argumento fue : "+args);
}
if (navigator.appName && navigator.appName.indexOf("Microsoft") != -1 &&
navigator.userAgent.indexOf("Windows") != -1 && navigator.userAgent.indexOf("Windows 3.1") == -1)
{
document.write('<SCRIPT LANGUAGE=VBScript\> \n');
document.write('on error resume next \n');
document.write('Sub Pelicula_FSCommand(ByVal command, ByVal args)\n');
document.write(' call Pelicula_DoFSCommand(command, args)\n');
document.write('end sub\n');
document.write('</SCRIPT\> \n');
}
-->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
<!--Esconde el código de los navegadores que no lo entienden
function ValorEnTexto(valor){
var PeliculaObj = InternetExplorer? Pelicula: document.Pelicula;
PeliculaObj.SetVariable ("campotexto",valor);
```

```
}
-->
</SCRIPT>
</HEAD>
<BODY>
<OBJECT id=Pelicula
codeBase=http://download.macromedia.com/pub/
shockwave/cabs/flash/swflash.cab#version=6,0,0,0}
classid=clsid:D27CDB6E-AE6D-11cf-96B8-444553540000 width=550 height=400><PARAM NAME="_cx"
VALUE="14552">
<PARAM NAME="_cy" VALUE="10583">
<PARAM NAME="FlashVars" VALUE="14552"><PARAM NAME="Movie" VALUE="Pelicula.swf">
<PARAM NAME="Src" VALUE="Pelicula.swf"><PARAM NAME="WMode" VALUE="Transparent">
<PARAM NAME="Play" VALUE="-1"><PARAM NAME=" Loop " VALUE="-1">
<PARAM NAME="Quality" VALUE="High"><PARAM NAME="SAlign" VALUE="">
<PARAM NAME="Menu" VALUE="0"><PARAM NAME="Base" VALUE="">
<PARAM NAME="AllowScriptAccess" VALUE="always"><PARAM NAME="Scale" VALUE="ShowAll">
<PARAM NAME="DeviceFont" VALUE="0"><PARAM NAME="EmbedMovie" VALUE="0">
<PARAM NAME="BGColor" VALUE="FFFFFF"><PARAM NAME="SWRemote" VALUE="">
<EMBED src="Pelicula.swf" menu=false quality=best wmode=transparent bgcolor=#FFFFFF WIDTH="550"
HEIGHT="400" NAME="Pelicula" ALIGN="" TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">
</EMBED></OBJECT>
<P><INPUT id="texto"><INPUT id="boton" type=button value="Click Aqui"
OnClick="javascript:ValorEnTexto(texto.value);"></P>
</BODY>
</HTML>
```

De esta manera podemos comunicar a JavaScript con flash fácilmente, pero hagamos algo más; ahora vamos a hacer que al dar click al botón de Flash El valor que estaba en el campo de texto de Flash se pase como el texto que tiene el botón HTML asignado.

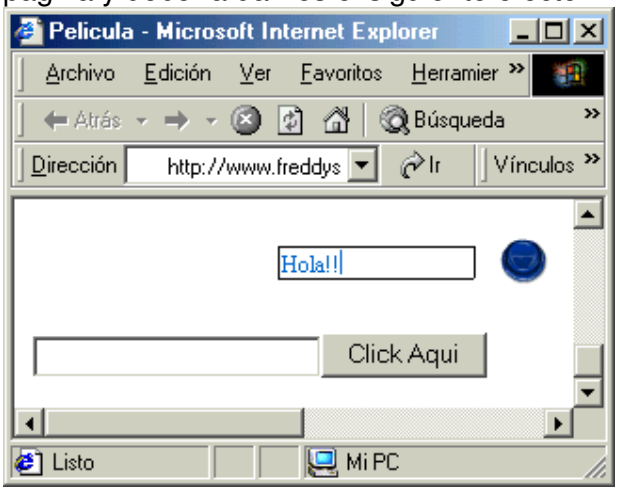
Para esto, inserten este código dentro de Flash en el botón:

```
on (release) {
fscommand ("comando", campotexto);
}
```

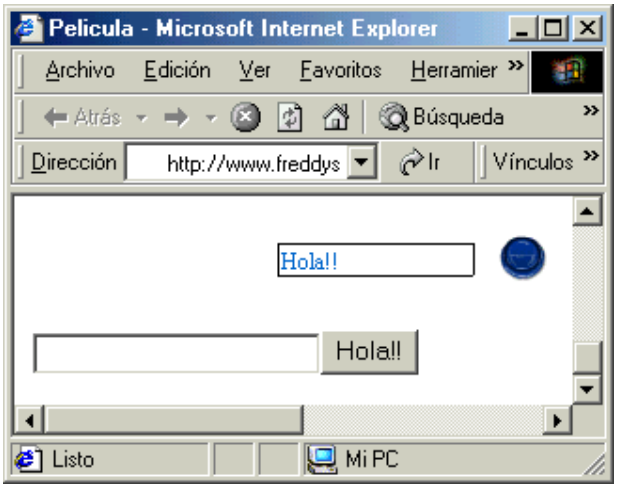
Y luego en la página HTML inserten este código en la función "Pelicula_DoFSCommand":

```
function Pelicula_DoFSCommand (command, args) {
var PeliculaObj = InternetExplorer? Pelicula:document.Pelicula;
//Aquí va a ir nuestro código de manejo
boton.value=args;
}
```

La instrucción "boton.value=args;" lo que hace es asignar el parámetro args de el FSCommand como el texto que aparece en el botón, vamos a probar la pagina y debería darnos el siguiente efecto:



Antes del click al botón Flash (El de color Azul en este caso)



Después del click (Fíjense que el texto del botón HTML (el gris), cambio al que estaba en el campo de texto de Flash)

Ya con esto saben como comunicar de manera efectiva los lenguajes ActionScript y JavaScript; así logrando muchísimas cosas mas, por ejemplo, usar JavaScript para pasarle datos a Flash que no puede hacerlos por si solo, como valores del navegador o versiones del explorador ETC, así como usar propiedades y funciones de JavaScript en Flash.
Su código debió quedar así:

```
<HTML>
<HEAD>
<TITLE>Pelicula</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
var InternetExplorer = navigator.appName.indexOf("Microsoft") != -1;]
//Maneja todos los mensajes de FSCommands en una película
function Pelicula_DoFSCommand(command, args) {
var PeliculaObj = InternetExplorer? Pelicula : document.Pelicula;
//Aquí va a ir nuestro código de manejo
boton.value=args;
}
if (navigator.appName && navigator.appName.indexOf("Microsoft") != -1 &&
navigator.userAgent.indexOf("Windows") != -1 && navigator.userAgent.indexOf("Windows 3.1") == -1)
{
document.write('<SCRIPT LANGUAGE=VBScript\> \n');]
document.write('on error resume next \n');
document.write('Sub Pelicula_FSCommand(ByVal command, ByVal args)\n');
document.write(' call Pelicula_DoFSCommand(command, args)\n');
document.write('end sub\n');
document.write('</SCRIPT\> \n');
}
-->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
<!--Esconde el código de los navegadores que no lo entienden
function ValorEnTexto (valor){
var PeliculaObj = InternetExplorer? Pelicula: document.Pelicula;
PeliculaObj.SetVariable ("campotexto",valor);
}
-->
</SCRIPT>
</HEAD>
<BODY>
<OBJECT id=Pelicula
codeBase=http://download.macromedia.com/pub/
shockwave/cabs/flash/swflash.cab#version=6,0,0,0
classid=clsid:D27CDB6E-AE6D-11cf-96B8-444553540000 width=550 height=400><PARAM NAME="_cx"
VALUE="14552">
```

```
<PARAM NAME="_cy" VALUE="10583">
<PARAM NAME="FlashVars" VALUE="14552"><PARAM NAME="Movie" VALUE="Pelicula.swf">
<PARAM NAME="Src" VALUE="Pelicula.swf"><PARAM NAME="WMode" VALUE="Transparent">
<PARAM NAME="Play" VALUE="-1"><PARAM NAME=" Loop " VALUE="-1">
<PARAM NAME="Quality" VALUE="High"><PARAM NAME="SAlign" VALUE="">
<PARAM NAME="Menu" VALUE="0"><PARAM NAME="Base" VALUE="">
<PARAM NAME="AllowScriptAccess" VALUE="always"><PARAM NAME="Scale" VALUE="ShowAll">
<PARAM NAME="DeviceFont" VALUE="0"><PARAM NAME="EmbedMovie" VALUE="0">
<PARAM NAME="BGColor" VALUE="FFFFFF"><PARAM NAME="SWRemote" VALUE="">
<EMBED src="Pelicula.swf" menu=false quality=best wmode=transparent bgcolor=#FFFFFF WIDTH="550"
HEIGHT="400" NAME="Pelicula" ALIGN="" TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">
</EMBED></OBJECT>
<P><INPUT id="texto"><INPUT id="boton" type=button value="Click Aqui"
OnClick="javascript:ValorEnTexto(texto.value);"></P>
</BODY>
</HTML>
```

Por último dejo esta tabla de los métodos que posee el objeto ActiveX Shockwave Flash y para que sirve cada uno, además de un ejemplo en JavaScript de cómo usarlo:

Método	**Explicación**	**Ejemplo**
Pelicula.Back()	Regresa un frame atrás la película	Pelicula.Back();
Pelicula.CurrentFrame()	(Solo lectura) Devuelve la posición actual de la cabeza de lectura en la línea de tiempo	Variable=Pelicula.CurrentFrame();
Pelicula.FlashVersion()	Regresa como valor la versión del plugin de Flash instalada	Variable= Pelicula.FlashVersion();
Pelicula.Forward()	Avanza un frame en la película	Pelicula.Forward();
Pelicula.FrameLoaded (FrameNum)	Devuelve un valor booleano(True o False) que dice si esta o no cargado el frame especificado por parámetro	Var=Película.FrameLoaded(1);
Pelicula.GetVariable(name)	Obtiene el valor de una variable	Var=Pelicula.GetVariable(campotexto);
Pelicula.GotoFrame(FrameNum)	Lleva la cabeza de lectura al frame especificado	Pelicula.GotoFrame(5);
Pelicula.IsPlaying	Booleano que indica si esta o no en ejecución la película	Var= Pelicula.IsPlaying();
Pelicula.LoadMovie(layer,url)	Carga una película en la película igual que la instrucción loadMovie de Flash	Pelicula.LoadMovie(1,movie.swf)
Pelicula.PercentLoaded	Devuelve el porcentaje cargado de la película	Var= Pelicula.PercentLoaded();
Pelicula.Play	Inicia el movimiento de	Pelicula.Play()

	la cabeza de lectura	
Pelicula.Rewind	Rebobina la película hasta el primer frame	Pelicula.Rewind();
Pelicula.SetVariable("nombre",valor)	Asigna un valor a una variable	Pelicula.SetVariable("patos",5)
Pelicula.SetZoomRect(left,top,right,bottom)	Ajusta el zoom de la película de acuerdo a unas coordenadas	Pelicula.SetZoomRect(50,75,100,50)
Pelicula.Stop	Para la película	Pelicula.Stop();
Pelicula.TCallFrame(target,FrameNum)	Llama el código que hay en un frame determinado de una instancia	Pelicula.TCallFrame("_root",5);
Pelicula.TCallLabel(target,label)	Lo mismo que el anterior pero con el label del frame en vez del numero	Pelicula.TCallLabel("_root","funciones");
Pelicula.TCurrentFrame(target)	Obtiene el frame en el que esta la cabeza de lectura actualmente de una instancia	Var=Pelicula.TCurrentFrame("_root.pato_mc");
Pelicula.TCurrentLabel(target)	Lo mismo que el anterior pero devuelve el nombre del label del frame actual	Var=Pelicula.TCurrentLabel("_root");
Pelicula.TGetProperty(target,property)	Devuelve el valor de la propiedad de un movie clip	Var=Pelicula.TGetProperty("movie_mc",_alpha);
Pelicula.TSetProperty(target,property,value)	Asigna un valor a la una propiedad de un movie clip	Pelicula.TSetProperty("movie_mc,_alpha,50)

5. Conclusiones

La realización de un conjunto de animaciones de tipos abstractos de datos y de un entorno web que facilite su visualización y distribución, supone nuevas alternativas no sólo a la hora de impartir las asignaturas que traten con estructuras de datos, puesto que muestran animaciones que reflejan claramente cómo se comportan las distintas estructuras y permiten realizar comparaciones entre unas y otras de manera clara y pueden ser una buena base para el profesor a la hora de impartir sus clases. Si no que también suponen un estimulante a los alumnos, ya que por medio de animaciones que ellos mismos pueden realizar pueden comprender y asimilar más fácilmente los conceptos y el comportamiento de todas ellas.

Con este nuevo proyecto se han tratado de cubrir más tipos de datos y a su vez, como ya se ha comentado, dotarlos de interactividad, resultando así más atractivas éstas tanto para profesores como para alumnos.

La elaboración de un manual en el que se refleja claramente el manejo de ActionScript en términos generales, pero sobretodo enfocado a lo que nosotros hemos utilizado, pretende resolver dudas y servir de ayuda a cualquiera que quiera realizar aplicaciones en ActionScript. Pero sobretodo se ha realizado pensando en una posible continuación de este proyecto, facilitando así a los sucesores la extensión de éste ya que se explican en él todos los problemas surgidos y las soluciones tomadas.

6. Bibliografía

[1] Sistema multiusuario para la gestión universitaria vía Web, incluyendo el desarrollo de una interfaz para el diseño de aplicación de datos visuales; R. Carrasco, A. J. López, J. J. Crespo; Madrid (2006).

[2] Estructuras de datos y métodos algorítmicos, Ejercicios resueltos; N. Martí Oliet, Y. Ortega Mallén, J. A. Verdejo López; Madrid (2004).

[3] Algoritmo de inserción de árboles 234: www.owlnet.rice.edu/~comp212/01-fall/lectures/33/

[4] http://www.actionscript.org/

[5] http://www.cristalab.com/

[6] http://www.flashadvisor.com/forum/archive/flash-scripting-f12.html

[7] http://miprogramacionenjuegos.wordpress.com/2007/01/16/asociar-vincular-o-heredar-una-clase-a-un-movieclip/

[8] http://www.x-flash.org/

[9] http://www.adobe.com/support/flash/action_scripts/actionscript_dictionary/

7. **Autorización.**

Los autores de este proyecto, Ana Colinas Lorenzo, Alejandro García Asenjo y Aída Mengual Castro, de la asignatura Sistemas Informáticos, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, tanto la propia memoria, como el código y/o la documentación, siempre que seamos mencionados expresamente como los autores.

Ana Colinas Lorenzo

Alejandro García Asenjo

Aída Mengual Castro