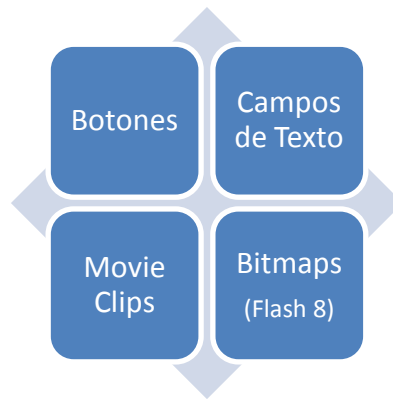
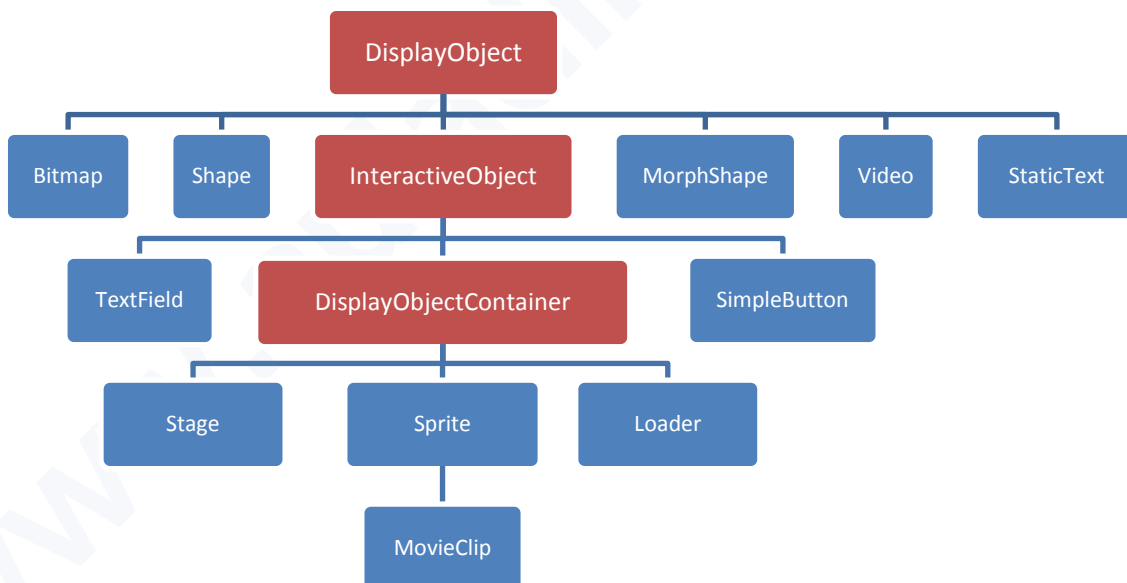


Jerarquía de Clases

En versiones anteriores de Flash y ActionScript, gestionábamos el contenido gráfico a través de cuatro elementos principales:



Aunque estos elementos continúan existiendo en AS3, las clases que los representan han sido notablemente optimizadas y reestructuradas bajo un contexto más amplio, algo que podemos observar en la siguiente jerarquía:



En ActionScript 3.0, todas las clases definidas para manipular el contenido gráfico heredarán su comportamiento de las clases **DisplayObject**, **InteractiveObject** y **DisplayObjectContainer** (resaltadas en el esquema).

La clase DisplayObject

Es la clase principal de la jerarquía, y define las capacidades y características gráficas básicas de todos los objetos visualizables en Flash Player. Se trata de una clase *abstracta*, cuyo único rol es el de definir los métodos y propiedades que heredarán sus subclases.

Las clases **Bitmap**, **Shape**, **MorphShape**, **Video** y **StaticText** son descendientes directas de la clase **DisplayObject**, y representan a los elementos gráficos primitivos, no interactivos:

Bitmap	Representa gráficamente a las imágenes en mapa de bits o <i>bitmaps</i> .
Shape	Actúa como un recipiente para la generación de gráficos vectoriales.
MorphShape	Representa las interpolaciones de forma entre gráficos vectoriales.
Video	Permite reproducir vídeos que no han sido incorporados al archivo SWF.
StaticText	Representa el texto estático, creado en la interfaz de Flash.

Las clases **MorphShape** y **StaticText** no podrán ser instanciadas a través de ActionScript.

La clase InteractiveObject

Esta subclase abstracta de la clase **DisplayObject** establece las normas de conducta para todas aquellas clases cuyas instancias tendrán la capacidad de responder a entradas del teclado (**TextField**) o del mouse (**SimpleButton**).

Los objetos que desciendan de la clase **InteractiveObject** nos permitirán dotar a una aplicación de **interactividad**, de ahí el nombre de la clase.

Las clases **TextField** y **SimpleButton** son descendientes directas de **InteractiveObject**:

TextField	Área rectangular que recibe entradas del teclado.
SimpleButton	Símbolos de botón creados en la interfaz de Flash o a través de código.

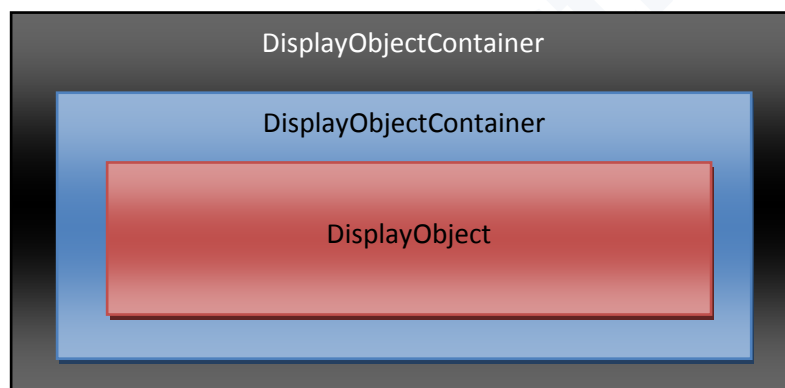
Respondiendo a *eventos* capturados por un campo de texto (**TextField**) o una instancia de botón (**SimpleButton**), lograremos dar interactividad a una aplicación. Existen otros elementos de interacción, tal como veremos a continuación.

La clase DisplayObjectContainer

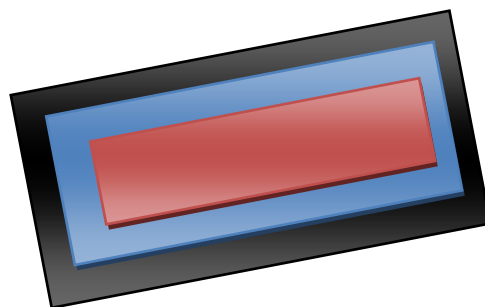
Esta subclase abstracta de la clase **InteractiveObject** es la encargada de fundar las bases para la administración del contenido gráfico. Sus objetos descendientes podrán contener uno o más objetos **DisplayObject**, u objetos de la propia clase **DisplayObjectContainer**, de modo que un objeto **DisplayObjectContainer** contenido en otro, podrá contener a su vez a un tercero.

En cualquier caso, sólo las instancias de las clases descendientes de **DisplayObjectContainer** podrán contener a otras instancias.

En el siguiente esquema observamos una instancia de **DisplayObject** que se encuentra anidada en un objeto **DisplayObjectContainer**, que a su vez se encuentra anidado en otro objeto del mismo tipo:



Al modificar las propiedades gráficas de un objeto contenedor (como su escala, rotación, etc), todos los objetos contenidos en él serán afectados por el cambio. En la siguiente figura, hemos rotado y escalado el objeto **DisplayObjectContainer** que contenía a los otros dos objetos:



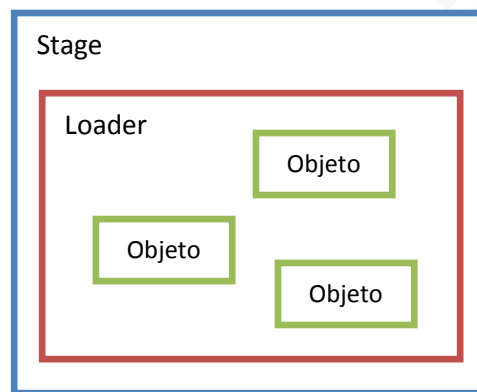
La clase **DisplayObjectContainer** tiene a **Stage**, **Loader** y **Sprite** como descendientes directas:

Stage Representa al escenario (el contenedor principal de cualquier aplicación).

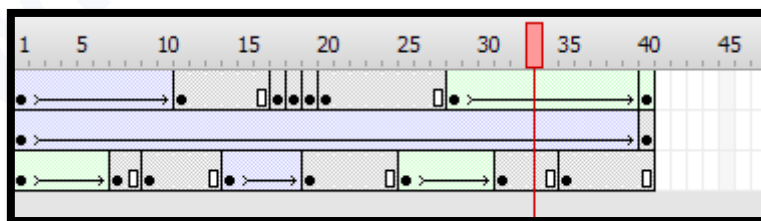
Loader Representa a contenedores que permiten la carga de contenidos externos.

Sprite Representa a objetos contenedores que permiten interacción.

De acuerdo a lo estudiado, todas las instancias de las clases **Stage**, **Loader** y **Sprite** podrán actuar como objetos contenedores. En el siguiente ejemplo, observamos tres objetos externos a la aplicación, que han sido cargados en una instancia de la clase **Loader**, que a su vez se encuentra en el escenario (**Stage**):



Los objetos **MovieClip** son descendientes directos de la clase **Sprite**. Además de actuar como contenedores y permitir la interacción, los objetos **MovieClip** nos ofrecen una línea de tiempo, en la cual podremos anidar todo tipo de animaciones y contenidos organizados en capas:



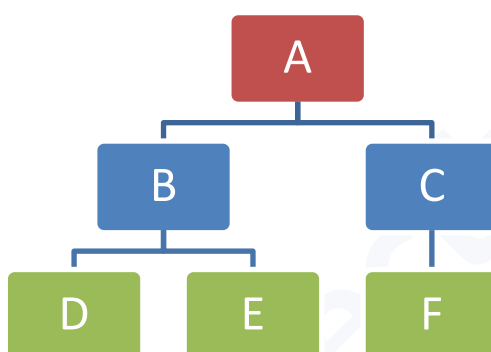
La incorporación de líneas de tiempo es el único factor que diferencia a los objetos **MovieClip** de los objetos **Sprite**. Por defecto, recurriremos a los objetos **Sprite** cuando necesitemos incorporar contenedores interactivos en nuestras aplicaciones, reservando el uso de objetos **MovieClip** para situaciones en las que necesitemos de una línea de tiempo para anidar algún tiempo de animación o secuencia.

Jerarquías de contención

En toda jerarquía, el objeto contenedor recibirá el nombre de *padre* (*parent*), mientras que el objeto contenido se llamará, naturalmente, *hijo* (*child*).

Cuando dispongamos de varios objetos agrupados en una misma estructura jerárquica, llamaremos *ancestros* a los objetos jerárquicamente superiores, y *descendientes* a los inferiores. El objeto que se encuentre al tope de la jerarquía recibirá el nombre de *raíz* (*root*).

En el esquema que mostramos a continuación, **A** es el objeto *raíz*, **B** es *padre* de **D** y de **E**, y **F** es *hijo* de **C**. Por su parte, **A**, **B** y **C** son *ancestros* de **D**, **E** y **F**, mientras que **B**, **C**, **D**, **E** y **F** son *descendientes* de **A**.



Imaginemos ahora que **A** es el escenario (**Stage**) y **B** es un objeto **Sprite** que a su vez contiene una instancia de **TextField** (**D**) y otra de **SimpleButton** (**E**).

Podemos apreciar en este sencillo ejemplo una perfecta jerarquía de contención:



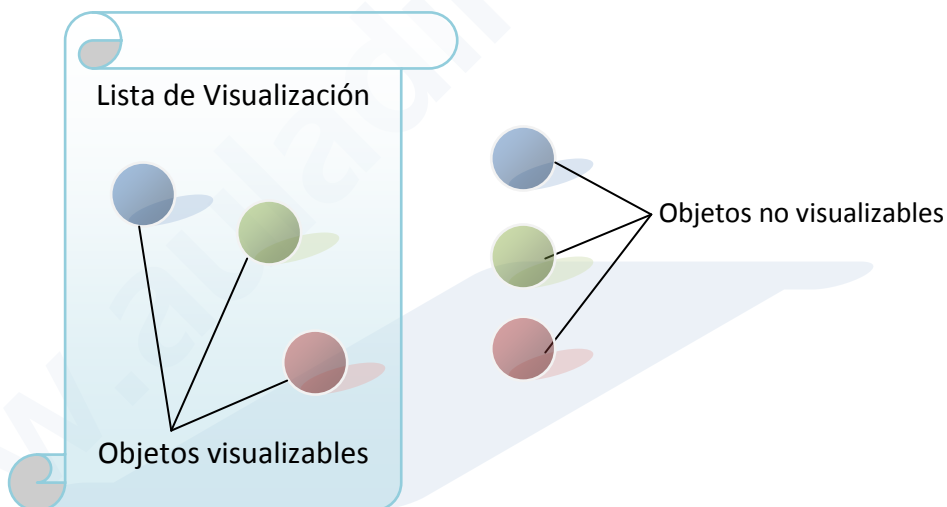
La Lista de Visualización

La lista de visualización representa la jerarquía de todos los objetos que serán visualizados al reproducir un archivo SWF en Flash Player. Todo objeto contenedor, incluyendo el escenario (**Stage**), contendrá una lista de visualización.

Podemos pensar en la lista de visualización como una estructura que establece el orden de “apilamiento” de los objetos gráficos que serán visualizados en Flash Player. El objeto más lejano será el primero de la lista, mientras que el objeto más cercano será el último.

En versiones anteriores de Flash, el sistema de representación de elementos gráficos se basaba en un objeto **MovieClip** principal o *raíz* (el escenario), en cuyo interior se anidaban los distintos elementos, incluyendo otros objetos **MovieClip** (los únicos objetos contenedores existentes en ese entonces). Una vez creados, los objetos eran insertados automáticamente en la jerarquía y representados gráficamente. Para reposicionar cualquiera de estos objetos en la jerarquía era necesario destruirlos y volver a generarlos.

El sistema de representación que utilizaremos en AS3 será menos rígido, en el sentido de que podremos desplazar libremente cualquier objeto gráfico o contenedor dentro de la jerarquía. No obstante, para poder visualizar los objetos creados, tendremos que añadirlos previamente a la lista de visualización de su contenedor.



Cabe aclarar que cualquier objeto podrá ser manipulado con código de ActionScript, aunque no se encuentre en la lista de visualización del contenedor. Una vez insertarlo en ella, será representado gráficamente con sus propiedades actuales (rotación, escala, posicionamiento, transparencia, etc), y al retirarlo de la misma, conservará el valor de estas propiedades aunque ya no podamos verlo. Del mismo modo, las propiedades de un objeto serán mantenidas aunque lo traslademos a un nuevo objeto contenedor.

La clase Stage

La clase **Stage** (o dicho con más propiedad, una “instancia” de la misma) constituye la raíz de la toda lista de visualización. Cada archivo SWF contendrá una única instancia de **Stage**, que será el contenedor principal de todos los elementos gráficos (incluyendo contenedores) que serán visualizados al ejecutar la aplicación.

A su vez, cada elemento contenedor (incluyendo el escenario) dispondrá de una propiedad llamada **stage** (perteneciente a la clase **DisplayObject**), que contendrá la jerarquía de todos los elementos gráficos contenidos en él. A través de esta propiedad podremos acceder a la raíz de toda lista de visualización.

En versiones anteriores de ActionScript, podíamos acceder a los métodos y propiedades del escenario haciendo una referencia directa a la clase **Stage**, accesible a nivel global. Esto es lo que hacíamos, por ejemplo, para recuperar el valor del ancho del escenario:

```
Stage.width;
```

En ActionScript 3.0, la única forma de acceder a las propiedades del escenario será a través de su propiedad **stage** (propiedad heredada de la clase **DisplayObject**). El equivalente al código anterior en AS3, será entonces:

```
stage.stageWidth;
```

Donde **stageWidth** es la propiedad equivalente a **width** en AS3.

Resumiendo, y para despejar toda ambigüedad, la clase **Stage** es la que define los métodos y propiedades relativos al nodo principal de una lista de visualización, al que podremos acceder a través de la instancia **stage**, desde cualquier objeto descendiente de **DisplayObject**.

Niveles de profundidad

Si colocamos dos objetos gráficos sobre el escenario (o dentro de un mismo contenedor) y les asignamos las mismas coordenadas, existirá evidentemente una superposición, en la que el objeto que ocupe el nivel más alto de profundidad cubrirá parcial o totalmente al otro objeto.

El *nivel de profundidad* de un objeto gráfico en AS3 corresponde a un número entero que determina la forma en que dicho objeto se superpondrá con otros objetos gráficos que se encuentren dentro del mismo contenedor.

A mayor número, mayor nivel de profundidad. Teniendo en cuenta que el 0 corresponde al nivel más bajo, el nivel de profundidad del objeto que se encuentre al tope de la lista de visualización, será igual al número de objetos en ese contenedor, menos uno.

Si tenemos, por ejemplo, 7 objetos anidados en un mismo contenedor, 0 será el nivel inferior y 6 el superior.

Cuando insertamos un nuevo elemento a la lista de visualización de un objeto contenedor, le es asignado automáticamente un nivel de profundidad, siempre de forma continua y ascendente, de forma tal que el último objeto añadido a la lista ocupará el nivel superior, situándose automáticamente “sobre” el resto de los objetos.

A diferencia de otras versiones, ActionScript 3.0 no admitirá la existencia de niveles sin ocupar

Insertar elementos en la lista de visualización

Para insertar un elemento en la lista de visualización de cualquier contenedor, nos valdremos del método **addChild()** de la clase **DisplayObjectContainer**.

En el siguiente ejemplo añadimos la instancia del objeto **cuadro** a la lista de visualización del objeto **pared**:

```
pared.addChild(cuadro);
```

En el ejemplo, asumimos que tanto **cuadro** como **pared** son los nombres de instancia de los objetos relacionados.

El método **addChild()** recibirá como único parámetro, el nombre de instancia del objeto que deseemos insertar en la lista de visualización del contenedor, y nos devolverá una referencia a dicho objeto.

Para insertar el objeto en un nivel de profundidad arbitrario, dentro de la lista de visualización del objeto contenedor, nos valdremos de la variante **addChildAt()**, al que pasaremos como parámetros:

- 1) El nombre de instancia del objeto a insertar.
- 2) Un valor numérico entero correspondiente al nivel de profundidad deseado.

En el siguiente ejemplo insertamos el objeto **libro** en el segundo nivel de profundidad (1, recordando que 0 es el primero) del objeto **biblioteca**:

```
biblioteca.addChildAt(libro,1);
```

Debemos tener cuidado al elegir el nivel, ya que como afirmamos varias líneas atrás, AS3 no admitirá la existencia de niveles discontinuos. Si tenemos ocupados, por ejemplo, los niveles 0 y 1 de la lista de visualización del contenedor, e invocamos al método **addChildAt()** pasándole un 3 como segundo parámetro, recibiremos un error. Atendiendo a las restricciones propias de AS3, sólo podremos añadir objetos en niveles de profundidad ya existentes, o en niveles inmediatamente superiores al último nivel ocupado. En nuestro ejemplo, 0,1 o 2.

Esto nos lleva a preguntarnos qué ocurriría si insertásemos un objeto en un nivel ocupado por otro objeto. En estos casos, el objeto que ocupaba originalmente el nivel elegido, pasaría a ocupar el nivel inmediatamente superior, y así lo harían el resto de los objetos posicionados en niveles superiores.

En nuestro ejemplo, si insertásemos un nuevo objeto en el nivel 1, el objeto que ocupaba dicho nivel pasaría al nivel 2, y tendríamos así los tres primeros niveles ocupados: 0, 1 y 2.

Eliminar elementos de la lista de visualización

Hemos visto dos métodos para insertar elementos en la lista de visualización de un objeto contenedor, delegando en AS3 la asignación del nivel de profundidad al utilizar `addChild()` o especificando el nivel con `addChildAt()`.

Para eliminar elementos de la lista de visualización de un objeto contenedor, contamos con los métodos análogos `removeChild()` y `removeChildAt()`. En el primer caso, pasaremos como parámetro el nombre de instancia del objeto a eliminar de la lista de visualización. En el segundo caso, especificaremos en cambio el nivel de profundidad del objeto a eliminar.

En el siguiente ejemplo, eliminamos el objeto `cuadro` de la lista de visualización del objeto `pared`:

```
pared.removeChild(cuadro);
```

Obtener el nivel de un objeto

Para obtener el nivel de profundidad de un objeto que se encuentra en la lista de visualización de un objeto contenedor, nos valdremos del método `getChildIndex()`, perteneciente a la clase `DisplayObjectContainer`. Este método tomará como único parámetro el nombre de instancia del objeto cuyo nivel deseamos obtener, y nos devolverá el valor numérico correspondiente a dicho nivel.

En el siguiente ejemplo invocamos al método `getChildIndex()` para obtener el nivel que ocupa el objeto `cuadro` dentro de la lista de visualización del objeto `pared`:

```
var nivel:uint = pared.getChildIndex(cuadro);
```

Obtener el total de objetos en una lista de visualización

Para conocer la cantidad de elementos que se encuentran en la lista de visualización de un objeto contenedor, usaremos la propiedad `numChildren` de la clase `DisplayObjectContainer`.

En el siguiente ejemplo recuperamos la cantidad de elementos en la lista de visualización del objeto `contenedor`, y almacenamos el valor en la variable `cantidad`:

```
var cantidad:uint = contenedor.numChildren;
```

Cambiar el nivel de profundidad de un objeto

La clase **DisplayObjectContainer** nos provee de los siguientes tres métodos para asignar o cambiar el nivel de profundidad de un objeto que se encuentra en la lista de visualización de un contenedor:

swapChildren()

Este método nos permitirá intercambiar los niveles de profundidad de dos objetos situados en la lista de visualización de un mismo contenedor. Para ello, le pasaremos como parámetros los nombres de instancia de los objetos cuyos niveles deseemos intercambiar.

En el siguiente ejemplo, intercambiamos los niveles de profundidad de los objetos **obj1** y **obj2** en la lista de visualización del objeto **contenedor**:

```
contenedor.swapChildren(obj1,obj2);
```

swapChildrenAt()

Esta variante del método **swapChildren()** nos permitirá intercambiar los niveles de los objetos que ocupen dos posiciones dadas en la lista de visualización de un objeto contenedor, cuyos valores numéricos le pasaremos como parámetros.

En el siguiente ejemplo, intercambiamos los niveles de los objetos que se encuentran en las posiciones 1 y 3 de la lista de visualización del escenario:

```
swapChildrenAt(1,3);
```

setChildIndex()

Con este método podremos asignarle a un objeto un nivel de profundidad arbitrario (siempre que sea válido), pasándole como parámetros el nombre de instancia del objeto y el valor numérico correspondiente al nivel donde deseamos posicionarlo.

En el siguiente ejemplo, asignamos al objeto **obj1** el nivel de profundidad del objeto **obj3**:

```
setChildIndex(obj1,getChildIndex(obj3));
```

Imaginemos ahora que tenemos tres objetos cuyos nombres de instancia son **obj1**, **obj2** y **obj3**, ocupando respectivamente los niveles 0, 1 y 2. Al ejecutar el código del ejemplo anterior, la instancia **obj1** pasará a ocupar el nivel de **obj3** (2), pero ¿qué pasará entonces con **obj3**, y qué instancia ocupará el nivel dejado por **obj1**?

Cuando llevamos un objeto a un nivel superior con el método **setChildIndex()**, el nivel de los objetos situados entre la posición original y la final decrecerá en 1. En nuestro ejemplo, **obj2** pasará a ocupar el nivel 0, y **obj3** el nivel 1.

Si por el contrario llevamos el objeto a un nivel inferior, el nivel de los objetos situados entre las posiciones original y final será incrementado en 1.

Por ejemplo, al hacer lo siguiente:

```
setChildIndex(obj3, getChildIndex(obj1));
```

La instancia **obj3** pasará a ocupar el nivel de **obj1** (0), de modo que **obj1** se ubicará en el nivel 1, y **obj2** el nivel 2.

Acceder a objetos anidados en contenedores

Para hacer referencia a objetos que se encuentren anidados en contenedores, nos valdremos del operador punto (.), separando con él los nombres de las instancias implicadas, siguiendo la secuencia de contención:

Objeto contenedor » Objeto anidado

Por ejemplo, para hacer referencia al objeto **libro**, que se encuentra anidado en el contenedor **biblioteca**, haríamos lo siguiente:

```
biblioteca.libro
```

Ejercicio M5-01

Utilizar la sentencia **while** en combinación con el método **removeChildAt()** para eliminar todos los objetos de una lista de visualización.

Ejercicio M5-02

Combinar los métodos **addChildAt()** y **getChildIndex()** para insertar un objeto en el mismo nivel de profundidad de un objeto dado, haciendo que este último se desplace hacia el nivel inmediatamente superior.

Ejercicio M5-03

Combinar el método **setChildIndex()** con la propiedad **numChildren** para desplazar un objeto al tope de la lista (es decir, sobre el resto de los objetos situados en el mismo contenedor).