CS 175
Action Script

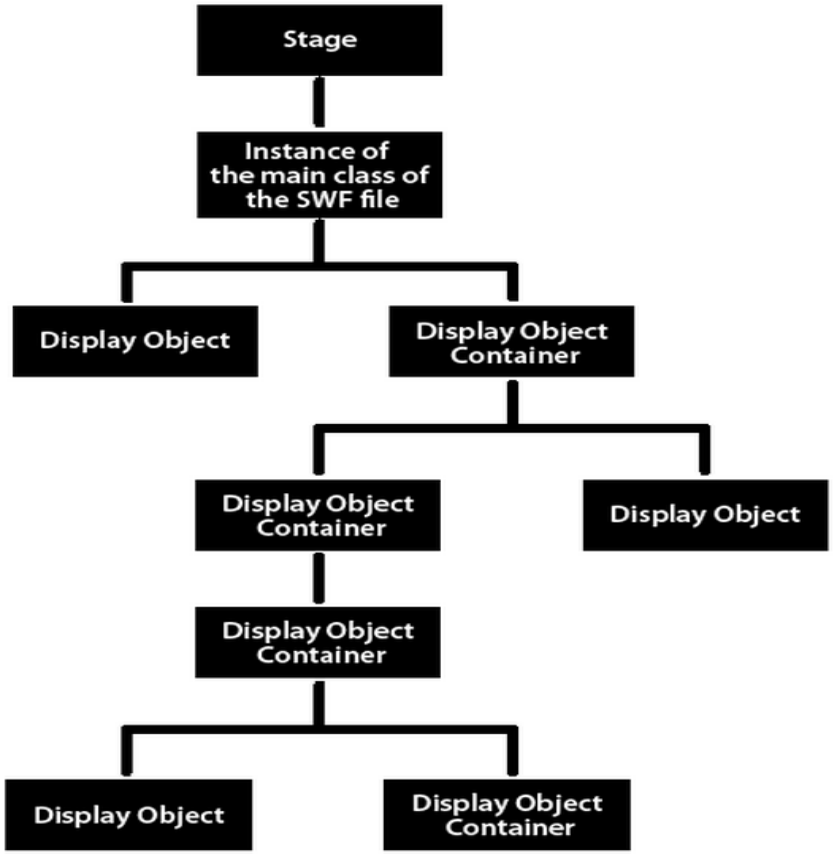
Display Programming

Introduction

- Display programming in Adobe® ActionScript® 3.0 allows you to work with elements that appear on the Stage.
- This chapter describes the basic concepts for working with on-screen elements.
- You'll learn the details about programmatically organizing visual elements.
- You'll also learn about creating your own custom classes for display objects.

Basics

- Each application built with ActionScript 3.0 has a hierarchy of displayed objects known as the display list, illustrated below. The display list contains all the visible elements in the application.



Stage

- The Stage is the base container of display objects.
- Each application has one Stage object, which contains all on-screen display objects.
- The Stage is the top-level container and is at the top of the display list hierarchy:
  - Each SWF file has an associated ActionScript class, known as the main class of the SWF file.
  - When a SWF file opens, Flash Player or AIR calls the constructor function for that class and the instance that is created (which is always a type of display object) is added as a child of the Stage object.
  - The main class of a SWF file always extends the Sprite class.

Display Objects

- In ActionScript 3.0, all elements that appear on screen in an application are types of display objects.
- The flash.display package includes a DisplayObject class, which is a base class extended by a number of other classes.
- These different classes represent different types of display objects, such as vector shapes, movie clips, text fields, etc ….

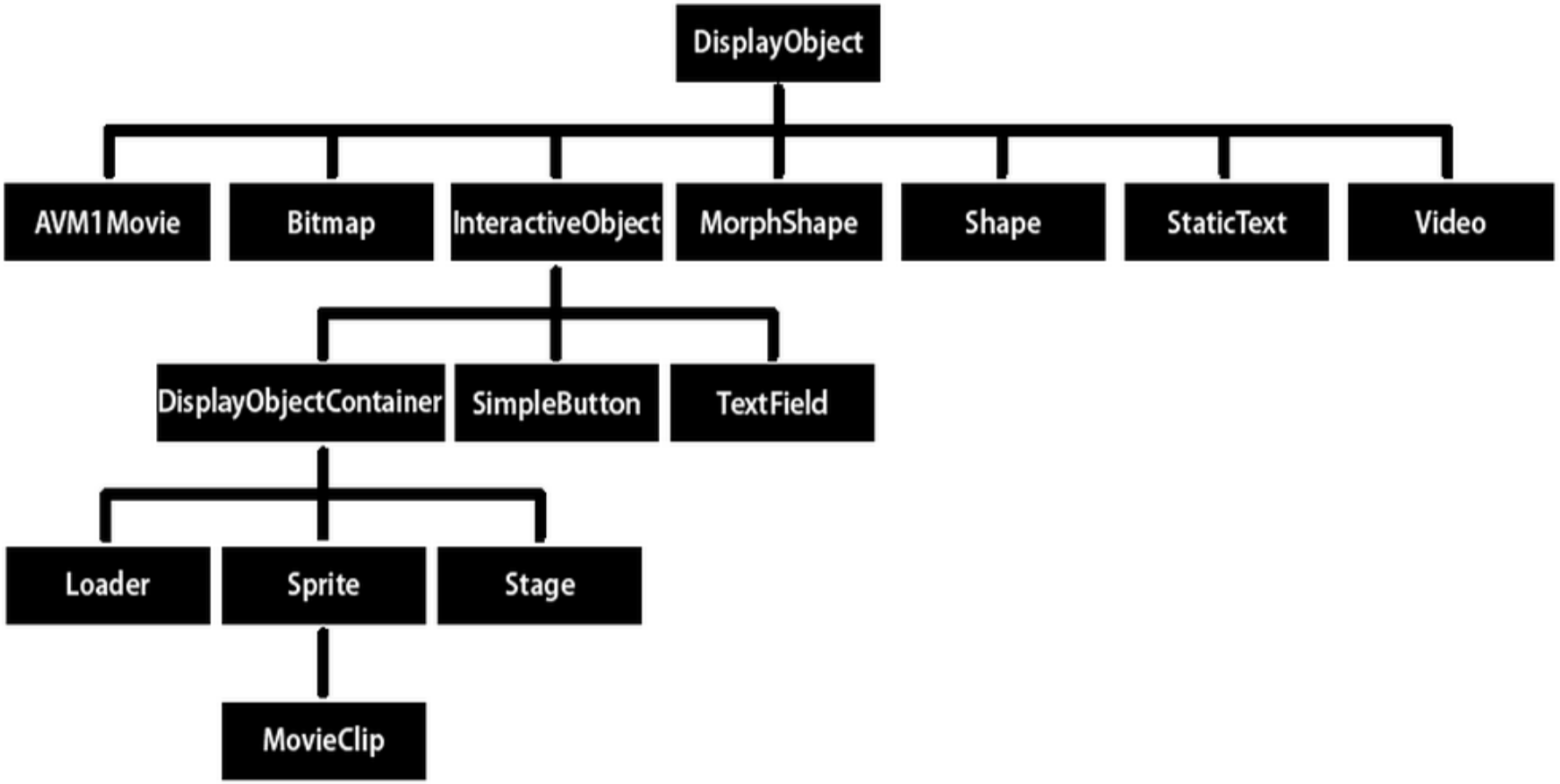
Display Object Containers

- Display object containers (or simply containers) are special types of display objects that, in addition to having their own visual representation, can also contain child objects that are also display objects.

- The DisplayObjectContainer class is a subclass of the DisplayObject class.

- A DisplayObjectContainer object can contain multiple display objects in its childlist.

Core Display Classes



Advantages of the display list approach

- In ActionScript 3.0, there are separate classes for different types of display objects.
- In ActionScript 1.0 and 2.0, many of the same types of objects are all included in one class: the MovieClip class.
- This individualization of classes and the hierarchical structure of display lists have the following benefits:
  - More efficient rendering and reduced memory usage
  - Full traversal of the display list
  - Easier subclassing of display objects

More efficient rendering and smaller file sizes

- In ActionScript 3.0, there are simpler display object classes in which you can draw shapes.
- Not all display object classes include the full set of methods and properties that a MovieClip object includes, they are less taxing on memory and processor resources.
- For example, each MovieClip object includes properties for its timeline, whereas a Shape object does not. So if you are not adding animation to your character, using the Shape object results in better performance and less memory usage.

Full traversal of the display list

- In ActionScript 1.0 and 2.0, you could not access some objects, such as vector shapes, that were drawn in the Flash authoring tool.

- In ActionScript 3.0, you can access all objects on the display list (both those created using ActionScript and all display objects created in the Flash authoring tool).

- We will cover that later in the chapter.

Easier subclassing of display objects

- Since the DisplayObject class in ActionScript 3.0 includes many built-in subclasses (including Shape and Bitmap), it is easier to create basic subclasses of the built-in classes.

- For example, in order to draw a circle in ActionScript 2.0, you could create a CustomCircle class that extends the MovieClip class. However, that class would also include a number of properties and methods from the MovieClip class (such as totalFrames, gotoAndStop …) that do not apply to the class.

- In ActionScript 3.0, however, you can create a CustomCircle class that extends the Shape object, and as such does not include the unrelated properties and methods that are contained in the MovieClip class

Working With Display Objects

Adding display objects to the display list

- When you instantiate a display object, it will not appear on-screen (on the Stage) until you add the display object instance to the display list or to a display object container that is on the display list.

```
var myText:TextField = new TextField();
myText.text = “Hello CS175";
stage.addChild(myText);
```

- When you add any visual element to the Stage, that element becomes a child of the Stage object.

Working with display object containers

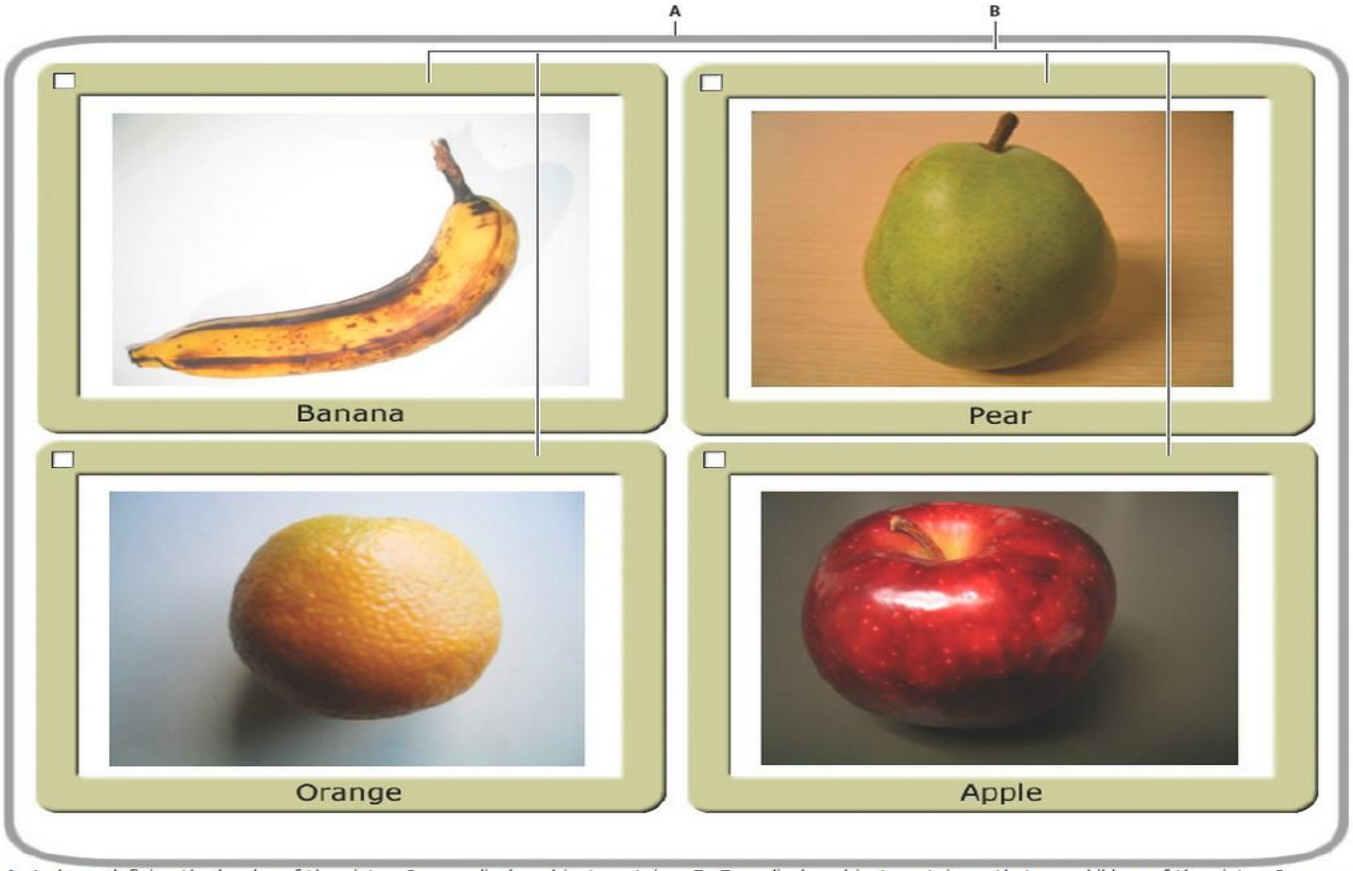
- If a ***DisplayObjectContainer*** object is deleted from the display list, or if it is moved or transformed in some other way, each display object in the ***DisplayObjectContainer*** is also deleted, moved, or transformed.

- A ***display object container*** is itself a type of ***display object***, so you can add it with all its children to another display object.

- You do this by using the ***addChild()*** method or the ***addChildAt()*** method of the container object

Working with display object containers

For example, the following image shows a display object container, ***pictureScreen***, that contains one outline shape and four other display object containers.



A. *A shape defining the border of the pictureScreen display object container* ***B.*** *Four display object containers that are children of the pictureScreen object*

Working with display object containers

- The ***DisplayObjectContainer class*** defines several methods for working with child display objects:

contains(): Determines whether a display object is a child of a DisplayObjectContainer.

getChildAt(): Retrieves the display object at a specified index.

getChildByName(): Retrieves a display object by name.

getChildIndex(): Returns the index position of a display object.

setChildIndex(): Changes the position of a child display object.

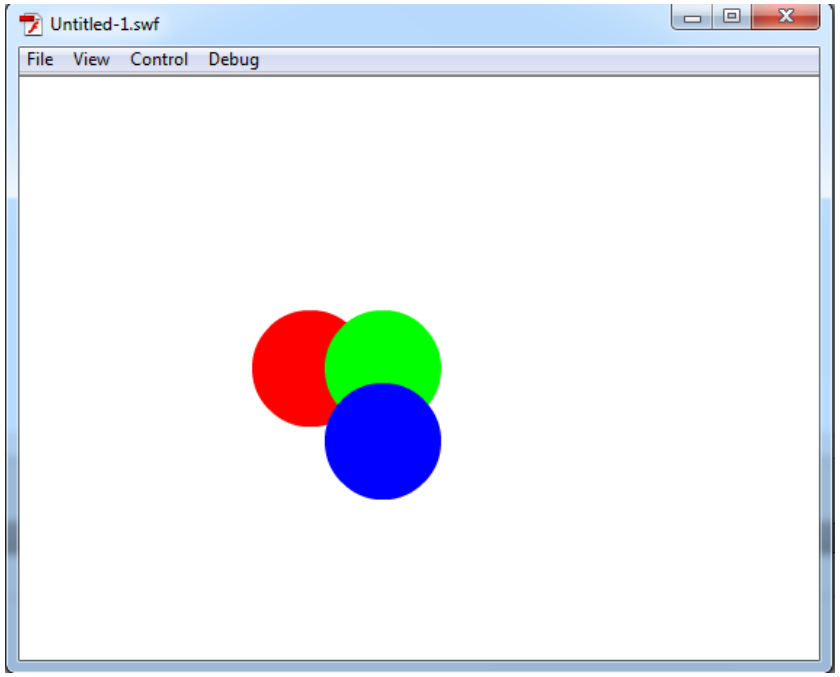
swapChildren(): Swaps the front-to-back order of two display objects.

swapChildrenAt(): Swaps the front-to-back order of two display objects, specified by their index values.

Working with display object containers

Examples:

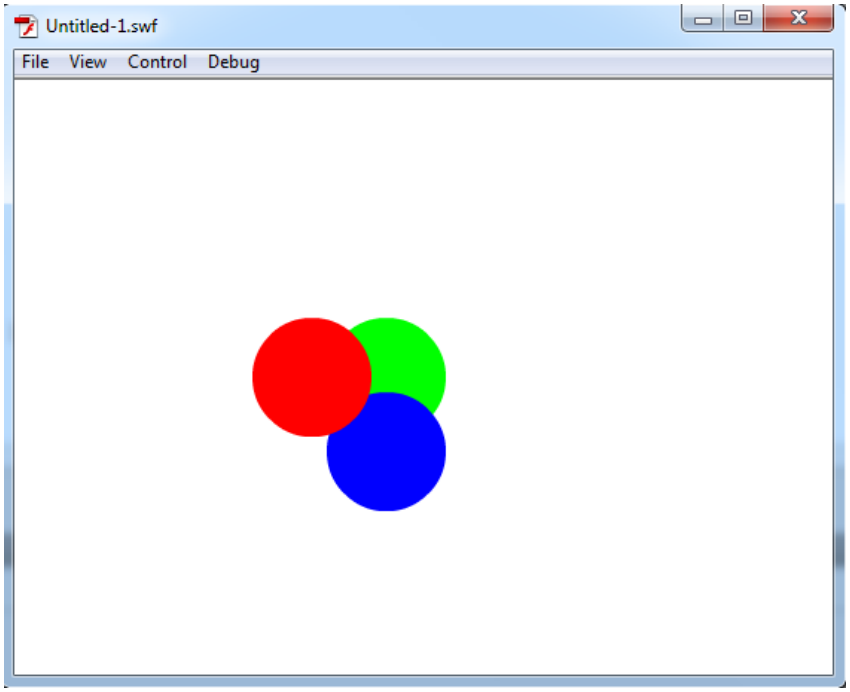
```
/* Circle 1: Red Circle*/
var circle1:Sprite = new Sprite();
circle1.graphics.beginFill(0xFF0000);
circle1.graphics.drawCircle(0, 0, 40);
circle1.x = 200;
circle1.y = 200;
stage.addChild(circle1);

/* Circle 2: Green Circle*/
var circle2:Sprite = new Sprite();
circle2.graphics.beginFill(0x00FF00);
circle2.graphics.drawCircle(0, 0, 40);
circle2.x = 250;
circle2.y = 200;
stage.addChild(circle2);

/* Circle 3: Blue Circle*/
var circle3:Sprite = new Sprite();
circle3.graphics.beginFill(0x0000FF);
circle3.graphics.drawCircle(0, 0, 40);
circle3.x = 250;
circle3.y = 250;
stage.addChild(circle3);
```



Working with display object containers

Examples:

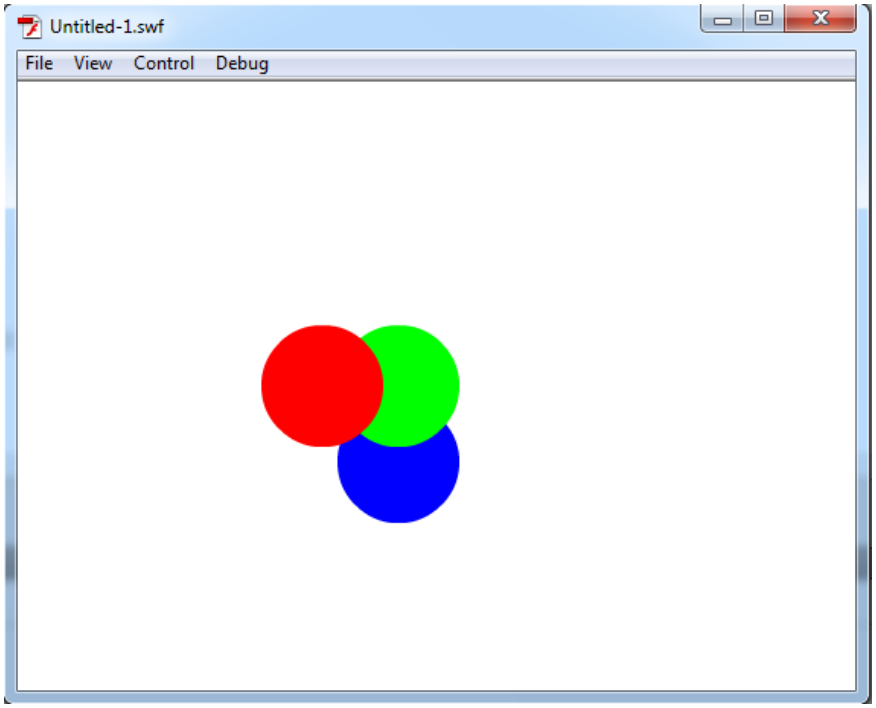
```
/* Circle 1: Red Circle*/
var circle1:Sprite = new Sprite();
circle1.graphics.beginFill(0xFF0000);
circle1.graphics.drawCircle(0, 0, 40);
circle1.x = 200;
circle1.y = 200;
stage.addChild(circle1);

/* Circle 2: Green Circle*/
var circle2:Sprite = new Sprite();
circle2.graphics.beginFill(0x00FF00);
circle2.graphics.drawCircle(0, 0, 40);
circle2.x = 250;
circle2.y = 200;
stage.addChild(circle2);

/* Circle 3: Blue Circle*/
var circle3:Sprite = new Sprite();
circle3.graphics.beginFill(0x0000FF);
circle3.graphics.drawCircle(0, 0, 40);
circle3.x = 250;
circle3.y = 250;
stage.addChild(circle3);

/* The red circles will be re-added and will show on top */
stage.addChild(circle1);
```



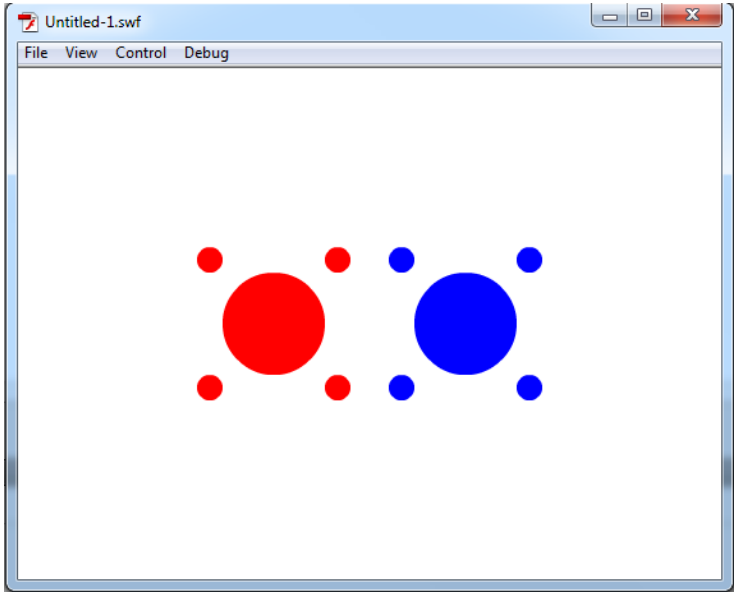
Working with display object containers

Examples:

```
/* Circle 1: Red Circle*/
var circle1:Sprite = new Sprite();
circle1.graphics.beginFill(0xFF0000);
circle1.graphics.drawCircle(0, 0, 40);
circle1.x = 200;
circle1.y = 200;
stage.addChild(circle1);

/* Circle 2: Green Circle*/
var circle2:Sprite = new Sprite();
circle2.graphics.beginFill(0x00FF00);
circle2.graphics.drawCircle(0, 0, 40);
circle2.x = 250;
circle2.y = 200;
stage.addChild(circle2);

/* Circle 3: Blue Circle*/
var circle3:Sprite = new Sprite();
circle3.graphics.beginFill(0x0000FF);
circle3.graphics.drawCircle(0, 0, 40);
circle3.x = 250;
circle3.y = 250;
stage.addChild(circle3);

/* The red and blue circles will be swapped */
stage.swapChildren(circle1 , circle3);
```



Traversing the display list



- Imagine the following scenario:
  - Big red circle is the parent of the four small red circles
  - Big blue circle is the parent of the four small blue circles
  - Big red circle is added to the stage
  - Big blue circle is added to the stage

- Now we want to traverse the whole list and output all the object's names

Traversing the display list

```
function traceDisplayList(container:DisplayObjectContainer, indentString:String = "")
{
    var child:DisplayObject;
    for (var i:uint=0; i < container.numChildren; i++)
    {
        child = container.getChildAt(i);
        trace(indentString, child, child.name);
        if (container.getChildAt(i) is DisplayObjectContainer)
        {
            traceDisplayList(DisplayObjectContainer(child), indentString + "    ")
        }
    }
}

traceDisplayList(stage);
```

Output:

```
[object MainTimeline] root1
[object Sprite] Big Red Circle
    [object Sprite] Small Red Circle 1
    [object Sprite] Small Red Circle 2
    [object Sprite] Small Red Circle 3
    [object Sprite] Small Red Circle 4
[object Sprite] Big Blue Circle
    [object Sprite] Small Blue Circle 1
    [object Sprite] Small Blue Circle 2
    [object Sprite] Small Blue Circle 3
    [object Sprite] Small Blue Circle 4
```

Working With Filters

Introduction

- One of the ways to add polish to an application is to add simple graphic effects, such as a drop shadow behind a photo to create the illusion of 3D, or a glow around a button to show that it is active.

- ActionScript 3.0 includes nine filters that you can apply to any display object or to a BitmapData instance.

- These range from basic filters, such as the drop shadow and glow filters, to complex filters for creating various effects, such as the displacement map filter and the convolution filter.

Creating a new filter

- To create a new filter object, simply call the constructor method of your selected filter class.
- For example, to create a new DropShadowFilter object, use the following code:

```
import flash.filters.DropShadowFilter;
var filterDropShadow:DropShadowFilter = new DropShadowFilter();
```

Note: Although not shown here, the DropShadowFilter() constructor (like all the filter classes' constructors) accepts several optional parameters that can be used to customize the appearance of the filter effect.

Check the help to customize your drop shadow effect.

Applying a filter

- When you apply filter effects to a display object, you apply them through the ***filters*** property.

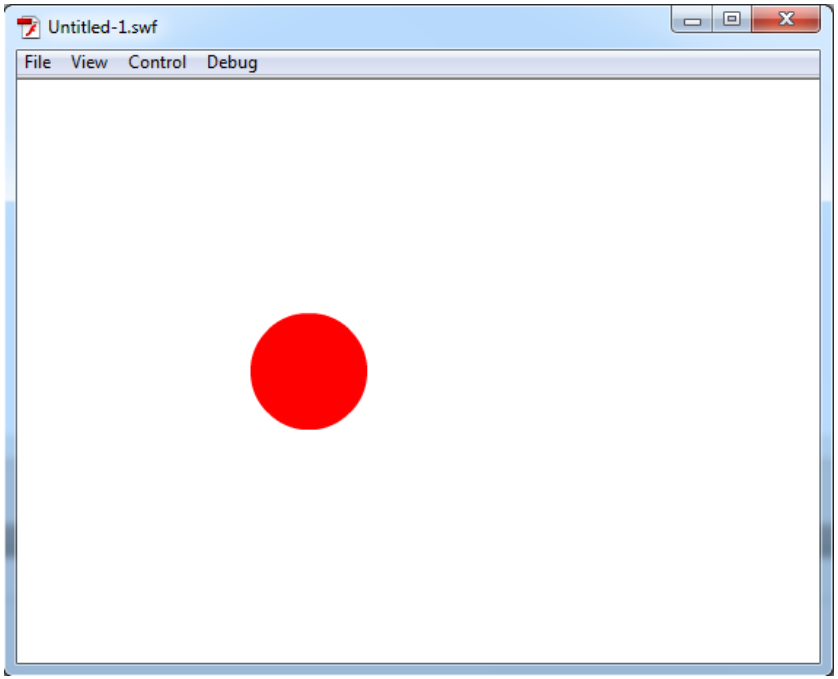
- The ***filters*** property of a display object is an ***Array*** instance, whose elements are the filter objects applied to the display object.

- To apply a single filter to a display object, create the filter instance, add it to an Array instance, and assign that Array object to the display object's filters property:

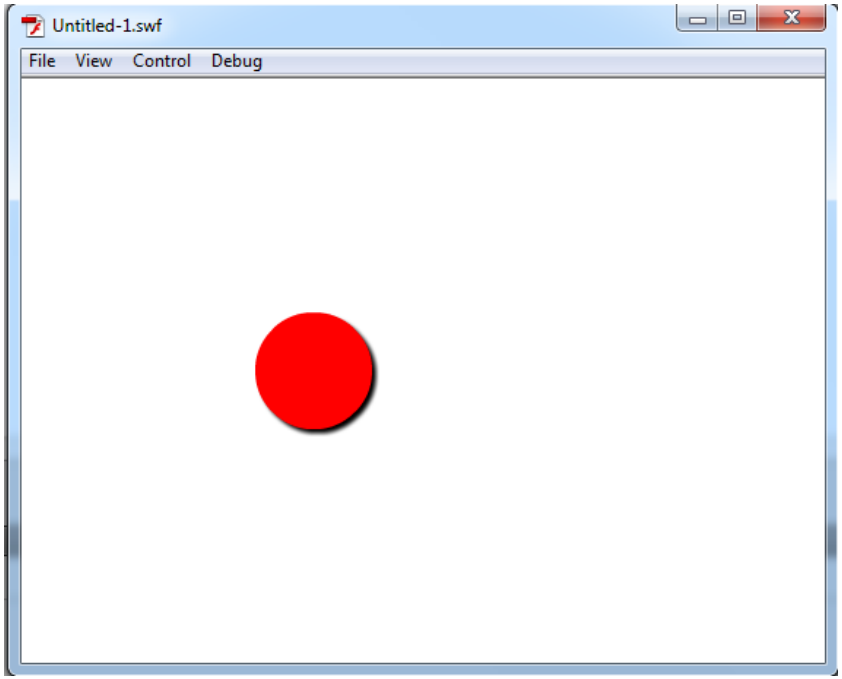
```
import flash.filters.DropShadowFilter;

/* Circle */
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFF0000);
circle.graphics.drawCircle(0, 0, 40);
circle.x = 200;
circle.y = 200;
stage.addChild(circle);

var filterDropShadow:DropShadowFilter = new DropShadowFilter();

circle.filters = new Array(filterDropShadow);
```

Applying a filter

Results:



Before



After

Applying multiple filters

If you want to assign multiple filters to the object, simply add all the filters to the Array instance before assigning it to the filters property.

```
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;

/* Circle */
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFF0000);
circle.graphics.drawCircle(0, 0, 40);
circle.x = 200;
circle.y = 200;
stage.addChild(circle);

var filterDropShadow:DropShadowFilter = new DropShadowFilter();
var filterGlow:GlowFilter = new GlowFilter();

circle.filters = new Array(filterDropShadow , filterGlow);
```

Note: **If you apply multiple filters to display objects, they are applied in a cumulative, sequential manner.**

Removing all filters

Removing all filters from a display object is as simple as assigning a null value to the filters property:

```
circle.filters = null;
```

Note: **If If you've applied multiple filters to an object and want to remove only one of the filters, you must go through several steps to change the filters property array. (I will talk about that in a coming slide).**

Available filters

- Bevel filter
- Blur filter
- Drop shadow filter
- Glow filter
- Gradient bevel filter
- Color matrix filter
- Convolution filter
- Displacement map filter
- Shader filter

Potential Issues

Changing filters at run time

- If a display object already has one or more filters applied to it, you can't change the set of filters by adding additional filters to or removing filters from the ***filters*** property array.

- Instead, to add to or change the set of filters being applied, you must make your changes to a separate array, then assign that array to the filters property of the display object for the filters to be applied to the object.

- The simplest way to do this is:
  - Read the filters property array into an Array variable
  - Make your modifications to this temporary array.
  - Then reassign this array back to the filters property of the display object.

Changing filters at run time

```
import flash.filters.DropShadowFilter;
import flash.filters.GlowFilter;

/* Circle */
var circle:Sprite = new Sprite();
circle.graphics.beginFill(0xFF0000);
circle.graphics.drawCircle(0, 0, 40);
circle.x = 200;
circle.y = 200;
stage.addChild(circle);

var filterDropShadow:DropShadowFilter = new DropShadowFilter();
var filterGlow:GlowFilter = new GlowFilter();

var aFilters:Array = new Array (filterDropShadow);
circle.filters = aFilters;

/* This will not work!!!! */
circle.filters.push(filterGlow);

/* This is fine */
aFilters.push(filterGlow);
circle.filters = aFilters;
```

Note: **The same goes for removing one filter**

The End ☺