CGT 353: Principles of Interactive and Dynamic Media
The Display API

Display Tools:

- Two kinds of display tools:
    - **The *display API***
    - **Components**
- **Components** will be discussed later….
- In Flash 8 (AS 2.0) and older, we used the following a LOT:
    - Movie Clip
    - Text Field
    - Button
    - Bitmap
- Still used in AS 3.0, but have been reorganized and redesigned to fit within the larger class-based structure of the new language.

Overview:

- In AS 3.0, all graphical content is created and manipulated with classes

  Question: *When you create a symbol with the FAT by manually dragging, that's not using classes, is it?*

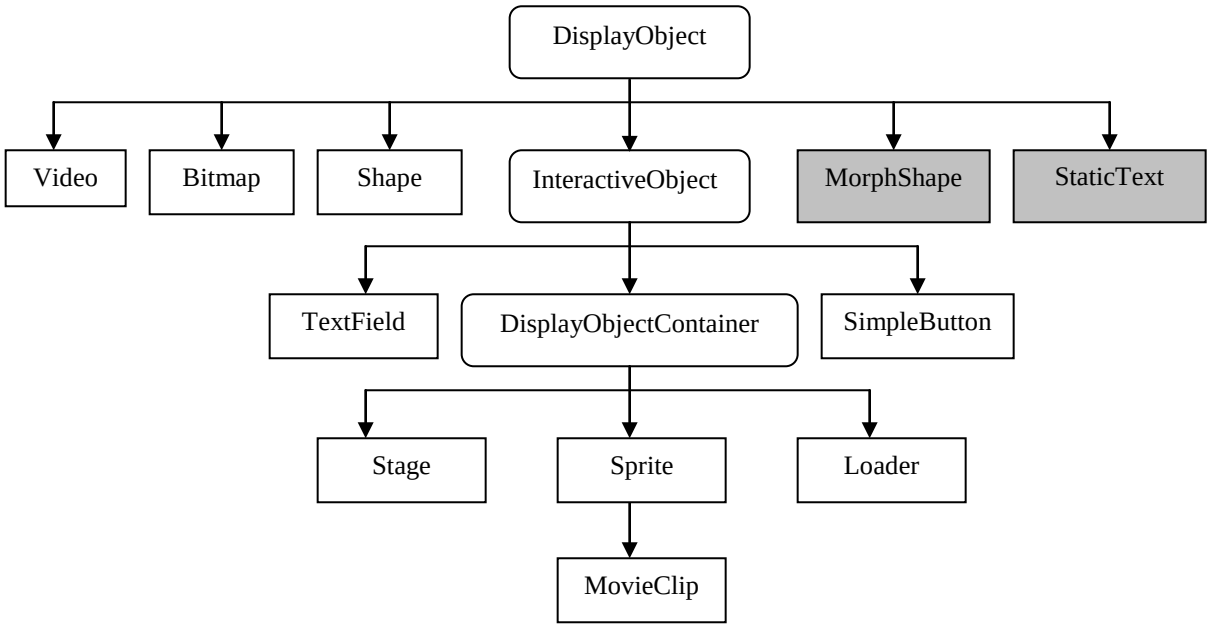
  Answer: *Yes it is…you are just drawing upon the automated features of what the FAT provides.*

- **Core display classes –** classes that directly represent on-screen content
- **Supporting display classes** – classes in the display API that indirectly represent on-screen content

Three Tiers of Display API Functionality:

1. **DisplayObject**
2. **InteractiveObject**
3. **DisplayObjectContainer**

- While these cannot be classified as ***abstract classes***, they function in that respect.
- AS does not support true abstract classes.
- **Abstract classes** have a name, parameters, and return type but no implementation (method body.)
- Solely used for extension and to create subclasses.
- So in AS, these type of classes are known as **abstract-style classes.**



Gray boxes denote classes available to the FAT only

- ***DisplayObject*** is the root of the core display classes and function to provide on-screen display
- Good for:

  1. Converting coordinates (ex. globalToLocal, localToGlobal, etc…)
  2. Checking intersections between points and objects (hitTestObject, hitTestPoint(), etc…)
  3. Applying filters, transforms, and masks
  4. Scaling disproportionately for "stretchy" graphical layouts

- **Note:** The phrase "Display object" refers to any object descending from the DisplayObject class...
- DisplayObject's subclasses (Video, Bitmap, Shape, MorphShape, and StaticText) are the simplest type of on-screen graphics...

- These cannot receive input or contain nested content....
- As FAT classes….*MorphShape* and *StaticText* cannot be instantiated in AS.....

InteractiveObject:

- Abstract class of DisplayObject.....
- Establishes second tier of display API functionality: **interactivity**
- All classes inheriting from *IO* can respond to input events from the mouse and keyboard.

DisplayObjectContainer:

- The only abstract subclass of *Interactive Object...*
- Represents the last tier of display API functionality: **containment.**
- **Containment** means that these classes can "hold" other classes of the display API...
- Used to group objects so they can be manipulated at once.
- Whenever a **DOC is added, manipulated, or deleted…**everything inside it goes along with it
- ***Sprite, MovieClip, Stage, and Loader*** subclasses each represent a unique containment structure waiting to be filled with content.
- ***Sprite*** is the foundation subclass, while ***MovieClip*** is basically a ***Sprite + animated content***
- ***Stage*** represents the main display area.
- ***Loader*** used to load external graphical content.
- Note that all this functionality used to be contained in the MovieClip class in AS 2.0
- Because AS 3.0 doesn't provide a way to create timeline elements such as frames and tweens….no reason to create new empty MovieClips like in AS 2.0
- To make programmatically created graphics in AS3, use one of the core display classes (*Bitmap, Shape, Sprite, Textfield, etc…)*
- *Remember in a 16-week class we don't have number to specifically cover the hundreds of new classes, methods, and properties*

Remember…

- Terminology comes to play in display API code and text:
    - Parent
    - Child (ex. addChild())
    - Ancestors
    - Descendents

Example: 2.0 vs. 3.0

2.0

```
// draw a red rectangle by creating an empty movieclip and using the drawing API

var shape_mc:MovieClip=_root.createEmptyMovieClip("shape",_root.getNextHighestDepth());
shape_mc.lineStyle(1, 0x000000);
shape_mc.beginFill(0xff0000);
shape_mc.moveTo(0, 0);
shape_mc.lineTo(50, 0);
shape_mc.lineTo(50, 50);
shape_mc.lineTo(0, 50);
shape_mc.lineTo(0, 0);
shape_mc.endFill();


// write an event handler

shape_mc.onPress=function(){
    text_txt.text="You pressed the rectangle";
}


// create a textfield object with the createTextField of the root movieclip

var text_txt = _root.createTextField("text", _root.getNextHighestDepth(), 100, 0, 150, 20);
text_txt.text = "Click the rectangle.";
```

In 3.0:

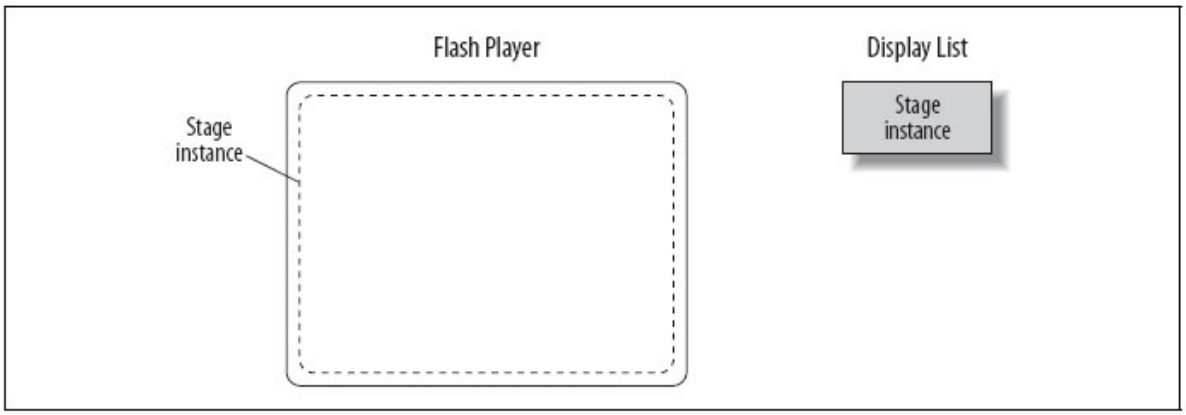
```
// create the new Sprite instance

var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xff0000);
mySprite.graphics.lineStyle(1, 0x000000);
mySprite.graphics.drawRect(0, 0, 50, 50);
mySprite.graphics.endFill();


//create the event handler

function clickHandler(event:Event):void {
    txt.text="You clicked the rectangle.";
}
mySprite.addEventListener(MouseEvent.CLICK,clickHandler);
```

```
// add the Sprite to the top-level container

addChild(mySprite);


//instantiate a text field

var txt:TextField = new TextField();
txt.x = 100;
txt.y = 0;
txt.width = 150;
txt.height = 20;

txt.text = "Click the rectangle.";


// add the text field to the top-level container

addChild(txt);
```

Source: See tutorial at
http://www.adobe.com/devnet/actionscript/articles/display_api.html

The Display List:

- To create actual graphics from theoretical classes you have to create instances of the core display classes and add them to the **display list.**
- The **display list** is the hierarchy of all graphical objects currently displayed by the Flash runtime.
- When you drag instances of symbols onto the Stage in the FAT this is done automatically.
- Any object added to the list and positioned in a visible area will show up on screen.
- Root of the display list is an instance of the Stage class, automatically created at runtime.



The Stage – Purposes:

1. Serves as the outermost container for all graphics displayed.
2. Provides information about the characteristics of the display area.

- *Stage* instances always accessed relative to some object on the display list

  Ex. output.txt.stage

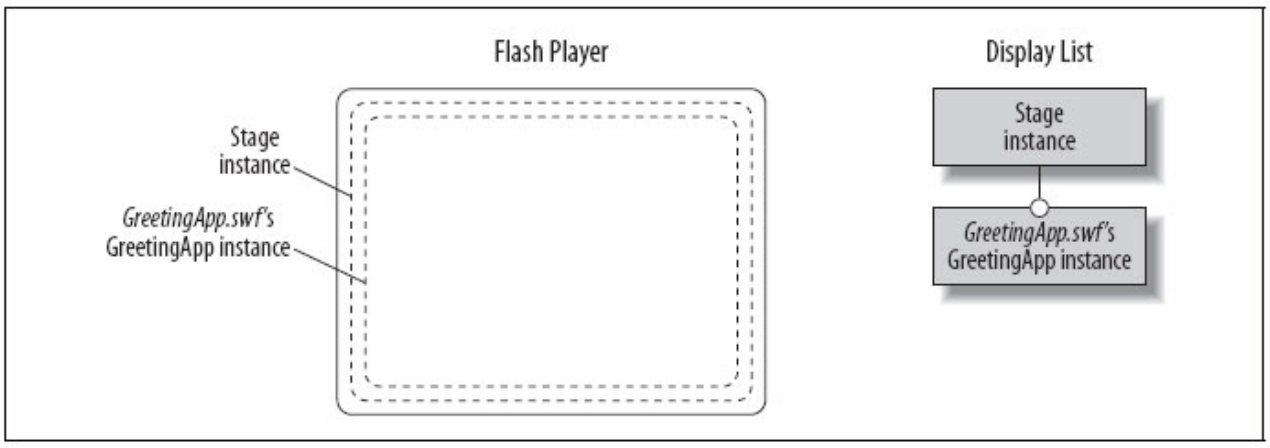
- In AS2, the *Stage* class didn't contain objects on the display list
- Used to use the *Stage* directly

  Ex. trace(Stage.align)

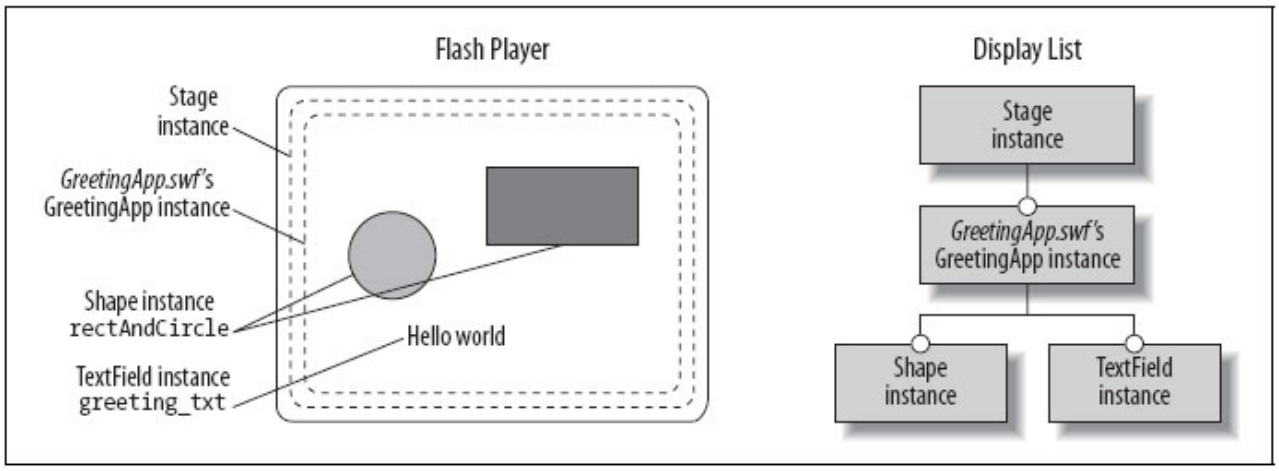
- In AS3 *Stage* methods and properties are not accessed directly, and there is no global point of reference.

Using the Stage:

- Remember that when you run a .swf file, the runtime locates the main class and makes an instance of it.
- The runtime then adds that instance to the display list as the *Stage* instances' first child.
- Even if the first class has no graphics (most will), it's still added to the display list



```
package {

  import flash.display.*;
  import flash.text.TextField;

  //GreetingApp extends a Sprite because this example is not intended for the FAT

  public class GreetingApp extends Sprite {
    public function GreetingApp() {

      // Create a rectangle
      var rect:Shape = new Shape();
      rect.graphics.lineStyle(1);
      rect.graphics.beginFill(0x0000FF, 1);
      rect.graphics.drawRect(0, 0, 75, 50);

      // Create a circle
      var circle:Shape = new Shape();
      circle.graphics.lineStyle(1);
      circle.graphics.beginFill(0xFF0000, 1);
      circle.graphics.drawCircle(0, 0, 25);
      circle.x = 75;
      circle.y = 35;

      // Create a text message
      var greeting_txt:TextField = new TextField();
      greeting_txt.text = "Hello world";
      greeting_txt.x = 60;
      greeting_txt.y = 25;

      // Add assets to the display list
      addChild(greeting_txt);  // Depth 0
      addChild(circle);        // Depth 1
      addChild(rect);          // Depth 2

      // Create a triangle
      var triangle:Shape = new Shape();
      triangle.graphics.lineStyle(1);
      triangle.graphics.beginFill(0x00FF00, 1);
      triangle.graphics.moveTo(25, 0);
      triangle.graphics.lineTo(50, 25);
      triangle.graphics.lineTo(0, 25);
      triangle.graphics.lineTo(25, 0);
      triangle.graphics.endFill();
      triangle.x = 70;
      triangle.y = 8;

      // Put the triangle beneath the circle.
      addChildAt(triangle, getChildIndex(circle));
    }
  }
}
```



Containers and Depths:

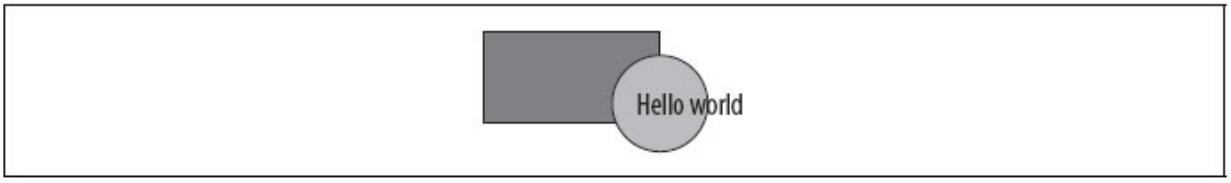
- From CGT 353 we remember that **depth** controls how objects overlap on the stage....
- The greater the number, the higher the position...
- Lowest object in the stacking order has a depth position of 0.

In AS 2…

- You could have "unoccupied" depths.
- Not allowed in 3.0.....
- Display objects added to a container using *addChild()* are assigned depth positions automatically
- Most recently added will always appear on top....

```
// The rectangle
var rect:Shape = new Shape( );
rect.graphics.lineStyle(1);
rect.graphics.beginFill(0x0000FF, 1);
rect.graphics.drawRect(0, 0, 75, 50);

// The circle
var circle:Shape = new Shape( );
circle.graphics.lineStyle(1);
circle.graphics.beginFill(0xFF0000, 1);
circle.graphics.drawCircle(0, 0, 25);
circle.x = 75;
circle.y = 35;

// The text message
var greeting_txt:TextField = new TextField( );
greeting_txt.text = "Hello world";
greeting_txt.x = 60;
greeting_txt.y = 25;
```

```
addChild(rect); // Depth 0
addChild(circle); // Depth 1
addChild(greeting_txt); // Depth 2
```



- To retrieve the depth position of any object in a display object container, we use *getChildIndex( )*:

```
trace(getChildIndex(rect)); // Displays: 0
```

- To add a new object at a specific depth position, we use *addChildAt( )*

```
theContainer.addChildAt(theDisplayObject, depthPosition)
```

- If the specified *depthPosition* is already occupied by an existing child, then *theDisplayObject* is placed behind that existing child (all others increase by one)

From:

```
greeting_txt 0
circle 1
rect 2
```

To:

```
greeting_txt 0
triangle 1
circle 2
rect 3
```

- To add a new object above all existing children:

```
theContainer.addChildAt(theDisplayObject, theContainer.numChildren)
```

- *Use addChildAt( )* in combination with *getChildIndex( )* to add an object below an existing child:

```
theContainer.addChildAt(newChild, theContainer.getChildIndex(existingChild))
```

- To swap depths of two children:

```
theContainer.swapChildren(existingChild1, existingChild2);

or

theContainer.swapChildrenAt(existingDepth1, existingDepth2);
```

- The *setChildIndex( )* method takes the following form:

```
theContainer.setChildIndex(existingChild, newDepthPosition);
```

- Be careful when using this method because of it bumping all the other children up – like putting a single card into a deck.

Removing Assets:

- Use *removeChild()* and *removeChildAt()*

- Be careful, because these only remove from the display hierarchy, <u>not memory.</u>

```
theContainer.removeChild(existingChild)

theContainer.removeChildAt(depth)
```

- To remove items from memory, you also have to remove all references to it.

- Even after all references have been removed object is active until the garbage collector deletes is from memory (more on this later)

Removing All Children:

- No direct method. Must be done with loops:

```
while (theParent.numChildren > 0) {
theParent.removeChildAt(0);
}


for (;numChildren > 0;) {
theParent.removeChildAt(0);
}
```

Manipulating Objects in Containers Collectively:

```
// Create two rectangles
var rect1:Shape = new Shape( );
rect1.graphics.lineStyle(1);
rect1.graphics.beginFill(0x0000FF, 1);
rect1.graphics.drawRect(0, 0, 75, 50);
var rect2:Shape = new Shape( );
rect2.graphics.lineStyle(1);
rect2.graphics.beginFill(0xFF0000, 1);
rect2.graphics.drawRect(0, 0, 75, 50);
rect2.x = 50;
rect2.y = 75;

// Create the container
var group:Sprite = new Sprite( );

// Add the rectangles to the container
group.addChild(rect1);
group.addChild(rect2);

// Add the container to the main application
someMainApp.addChild(group);

// Move, scale, and rotate container
group.x = 40;
group.scaleY = .15;
group.rotation = 15;
```

Descendant Access to .swf Main Class Instance:

```
package {
import flash.display.*;
import flash.geom.*;

public class App extends Sprite {
public function App ( ) {


// Make the descendants...
var rect:Shape = new Shape( );
rect.graphics.lineStyle(1);
rect.graphics.beginFill(0x0000FF, 1);
rect.graphics.drawRect(0, 0, 75, 50);

var sprite:Sprite = new Sprite( );
sprite.addChild(rect);
addChild(sprite);

// Use DisplayObject.root to access this App instance
trace(rect.root); // Displays: [object App]
trace(sprite.root); // Displays: [object App]
            }
      }
}
```

The Rebirth of _root:

- In AS2, _root referred to the top-level movie clip….
- Always followed that _root should be avoided because it was inflexible…
- New root variable doesn't suffer from this…

As for _level0…

- *loadMovieNum()* was used to stack external .swf files on top of one another.
- In AS3, external .swf files cannot be loaded directly onto stage instance child list...
- Instead, you have to load the .swf with a Loader object then move it to the Stage using stage.addChild()

```
var loader:Loader = new Loader( );
loader.load(new URLRequest("newContent.swf"));
stage.addChild(loader);
```

- Can also no longer remove everything by unloading _level0

```
unloadMovieNum(0);
```

- Can use:

```
stage.removeChildAt(0);
```

- But remember that you still have to remove the instances

```
while (stage.numChildren > 0) {
      stage.removeChildAt(stage.numChildren-1);
      // When the last child is removed, stage is set to null,
      so quit
      if (stage == null) {
            break;
      }
}
```