BASICS OF ACTIONSCRIPT 3.0 FOR GAME & ANIMATION
Basic Animation

- Animation is made with frames, with each frame slightly different to present the illusion of motion.

- Frame-by-frame or tweened animation contains an image or a description of an image for each frame.

- Dynamic animation contains an initial description of an image and rules that are applied to alter the description on each frame.
Classes and OOP

- A class basically has two things associated with it: properties (data or information) and behaviors (actions, or things it can do).

- Properties are essentially variables that hold information relating to the class.

- Behaviors are just functions, though when a function is part of a class, we usually refer to it as a method.
classes are templates, and objects (also known as instances) are individual manifestations of a particular class

```
package
{
    public class MyClass
    {
        public var myProperty:Number = 100;
        public function myMethod()
        {
            trace("I am here");
        }
    }
}
```

Packages are a way to group related classes together

You’ll need to include the `package` keyword and brackets.

This is also known as the default package.
The class has a name, “MyClass”, followed by another pair of opening and closing brackets that enclose the class.

Within the class are only two things: a variable named “myProperty” and a function named “myMethod”.

The word “public” means that any code outside the object will be able to access that property or call that method.

If you create properties or methods that are meant only for use within the class itself, you can label them “private”
ActionScript 3.0 also adds the modifiers **internal** and **protected**.

An internal property of a class can only be accessed by other classes in the same package.

Protected means that a property can only be seen by classes that extend this particular class.

A class is written and saved in an external text file named the same as the class, with the suffix .as.

You can create the file by using the ActionScript editor in the Flash IDE, Flex Builder, or any other text-editing program.
Packages are structured according to the directories in which they are located, and they can be nested several layers deep.

For instance, you could have a class named `Utils` in a set of folders: `com/friendsofed/makingthingsmove/`. (It is a common convention to use domain names as packages. This ensures the package will be unique.)

This class would be referred to as `com.friendsofed.makingthingsmove.Utils`.

```java
package com.friendsofed.makingthingsmove {

    public class Utils {
    
    }
}
```
As you can imagine, typing `com.friendsofed.makingthingsmove.Utils` every time you wanted to use some utility function would get pretty old.

You can put the following at the top of the package, before the class itself.

```java
import com.friendsofed.makingthingsmove.Utils;
```
OOP - Constructor

- A constructor which is a method that has the same name as the class and is automatically called when a new instance is created.

- First, create a class’

```java
package
{
  public class MyClass
  {
    public function MyClass(arg:String)
    {
      trace("constructed");
      trace("you passed " + arg);
    }
  }
}
```

- Then create the instance right on the timeline

```javascript
var myInstance:MyClass = new MyClass("hello");
```
OOP - Inheritance

- A class can inherit from, or extend, another class.
- It gets all the properties and methods that the other class has.
- The subclass can then add additional properties and methods, or change some of the ones from the superclass.

```asciidoc
package
{
  public class MyBaseClass
  {
    public function sayHello():void
    {
      trace("Hello from MyBaseClass");
    }
  }
}
```

```asciidoc
package
{
  public class MySubClass extends MyBaseClass
  {
    public function sayGoodbye():void
    {
      trace("Goodbye from MySubClass");
    }
  }
}
```
OOP - Inheritance

- Then create the instance right on the timeline

```javascript
var base:MyBaseClass = new MyBaseClass();
base.sayHello();

var sub:MySubClass = new MySubClass();
sub.sayHello();
sub.sayGoodbye();
```

- You can also change an existing method from the base class in the subclass by using `override`

```javascript
class MySubClass extends MyBaseClass {
    override public function sayHello():void {
        trace("Hola from MySubClass");
    }
    public function sayGoodbye():void {
        trace("Goodbye from MySubClass");
    }
}
```

MySubClass.as
A MovieClip/Sprite Subclass

- The MovieClip class is the template for all the movie clip object.

- Sprite class, you can think of a sprite as a movie clip without a timelined, so it is more lightweight.

- For example, you wanted to make a spaceship for a game. You might want it to contain graphics, have a position, move around, rotate, listen for enterFrame events for animation, and listen for keyboard and mouse events for interaction.

- These are all things that movie clips and sprites can do.

- You could then add custom properties such as speed, fuel, and damage, and custom behaviors such as takeOff, crash, shoot, and selfDestruct.
A MovieClip/Sprite Subclass

- Example of SpaceShip class that extends Sprite class

```java
package
{
import flash.display.Sprite;

public class SpaceShip extends Sprite
{
    private var speed:Number = 0;
    private var damage:Number = 0;
    private var fuel:Number = 1000;

    public function takeOff():void
    {
        // ...
    }

    public function crash():void
    {
        // ...
    }

    public function shoot():void
    {
        // ...
    }

    public function selfDestruct():void
    {
        // ...
    }
}
}```
Creating Your Document Class

- A document class is a class that extends Sprite or MovieClip, and is used as the main class for your SWF.
- When the SWF is loaded, the constructor of that class will be called.

- Create an actionscript 3.0 file
- Here is a template for the document class
- Save it as “Example.as”

```actionscript
package
{
    import flash.display.Sprite;

    public class Example extends Sprite
    {
        public function Example()
        {
            init();
        }

        private function init():void
        {
            // sample code goes here
        }
    }
}
```
Creating Your Document Class

- Create a new Example.fla file and save it in the same directory.
- In the Properties panel, you should notice a new field called Document Class, enter the name of your class there.
- When using a document class, avoid putting any code on the timeline of the FLA as this can cause a conflict with the code.
Creating Your Document Class

- Edit the Example.as
Animating with code

- Almost all coded animation contains some kind of loop.
- If you think about a frame-by-frame animation, you might come up with a series of bitmaps, where each frame is already an image and just needs to be shown.

*Figure 2-3. Frame-by-frame animation*
Things are a bit different in Flash.

Flash doesn’t create and store a new bitmap image for each frame.

Flash stores the position, size, color, and so on of each object on the stage and renders it.

Figure 2-4. Rendering and then displaying frames
But when you consider how I described a dynamic, coded animation, the flowchart looks more like this.
Animating with code

- Here is the sequence of actions Flash takes on each frame:
  - Position all objects that are currently on the stage
  - Execute all ActionScript that needs to be called on that frame.
  - Check if it is time to display the frame yet. Flash will check if at least 50 milliseconds have gone by since the last frame. If so, it will display the frame and move to the next frame. If not, it will wait until the right amount of time has passed.
Frame Loops

- The idea is that you put some code in a particular frame, and then on the next frame you put a statement, usually `gotoAndPlay`, that sends the playhead back to the previous frame.

- For example, you could have a movie clip on stage named `ball`. The code on frame 1 would be something like this.

  ```
  ball.x++; 
  ```

- Frame 2 would just have this.

  ```
  gotoAndPlay(1); 
  ```
Frame Loops

- you can have a three-frame setup.
- Frame 1 being an initialization frame, where you can put any code that you want to run only a single time, rather than loop.
- Frame 2 will now have the main actions.
- Frame 3 will just say

```javascript
gotoAndPlay(2);
```
Clip Events

- Clip events have been removed from AS3, but are worth mentioning as a footnote.
- Clip events refer to code that was placed, not on the timeline, but directly on a movie clip itself.
- This was done by selecting a movie clip on the stage and then typed some code in the Actions panel, that code was assigned directly to that clip.

```javascript
onClipEvent(EventName)
{
    // code goes here
}
```
Clip Events

- The event name refers to one of the many events that may occur in a Flash movie.

- Events fall into two main categories: system events and user events.
  - A system event is when something happens within the computer, Flash, or your movie, such as data loading, content loading, or a new frame starting.
  - User events refer to something the user has done, and generally have to do with keyboard or mouse activity.
Clip Events

- The two most used clip events were load and enterFrame.
- The load event occurred when the movie clip instance first appeared on stage.
- The enterFrame event occurred each time Flash was ready to start rendering the next frame.

```actionscript
onClipEvent (load)
{
    // initialization code
}

onClipEvent (enterFrame)
{
    // action code
}
```
Listeners and Handlers

- A listener is an object that listens for events.

```javascript
addEventListener(Event.ENTER_FRAME, onEnterFrame);
```

- There are other event types set up in the same way on other classes, such as `MouseEvent.MOUSE_DOWN`, `KeyboardEvent.KEY_DOWN`, `TimerEvent.TIMER`, and so on.

- If you have a sprite named `mySpriteButton` that is acting as a button. When the user clicks that sprite, it will generate a mouseDown event. To listen for the mouseDown event, you call the `addEventListener` method of that sprite.

```javascript
mySpriteButton.addEventListener(MouseEvent.MOUSE_DOWN, onSpritePress);
```
if you no longer want an object to listen to events, you can tell it to stop listening by calling the `removeEventListener` method.

```javascript
removeEventListener(Event.ENTER_FRAME, onEnterFrame);
```

For example, create example.fla and example.as, link them together.

The following code creates a new sprite, puts it on stage, draws some graphics in it, and adds an event listener to it.
Listeners and Handlers

```actionscript
package {
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class example extends Sprite {
        private var eventSprite:Sprite;
        public function example() {
            init();
        }
        private function init():void {
            eventSprite = new Sprite();
            addChild(eventSprite);
            eventSprite.graphics.beginFill(0xff0000);
            eventSprite.graphics.drawCircle(0,0,100);
            eventSprite.graphics.endFill();
            eventSprite.x=stage.stageWidth/2;
            eventSprite.y=stage.stageHeight/2;
            eventSprite.addEventListener(MouseEvent.MOUSE_DOWN,onMouseClick);
            eventSprite.addEventListener(MouseEvent.MOUSE_UP,onMouseRelease);
        }
        private function onMouseClick(event:MouseEvent):void {
            trace("mouse down");
        }
        private function onMouseRelease(event:MouseEvent):void {
            trace("mouse up");
        }
    }
}
```
Here comes our first example of ActionScript 3–based animation.

```actionscript
package {
    import flash.display.Sprite;
    import flash.events.Event;
    public class FirstAnimation extends Sprite {
        private var ball:Sprite;
        public function FirstAnimation() {
            init();
        }
        private function init():void {
            ball = new Sprite();
            addChild(ball);
            ball.graphics.beginFill(0xff0000);
            ball.graphics.drawCircle(0, 0, 40);
            ball.graphics.endFill();
            ball.x=20;
            ball.y=stage.stageHeight/2;
            ball.addEventListener(Event.ENTER_FRAME, onEnterFrame);
        }
        private function onEnterFrame(event:Event):void {
            ball.x++;
        }
    }
}
```
Prior to AS 3, you could create a number of different types of visual objects in a Flash movie, including movie clips, graphics, buttons, text fields, bitmaps, components, and basic shapes.

These objects weren’t really organized in a hierarchical structure, and they all had very different ways of being created, destroyed, and manipulated.

The differences among these objects has been totally addressed in AS 3.

Just about any type of object that you can see on stage is created from a class that extends the DisplayObject class.

All of these objects are part of the same big family now.
The Display List

- The code for creating a sprite, a movie clip, or a text field is now very similar. In fact, all display objects are now created the same way you would create any other type of object.

```javascript
var myTextfield:TextField = new TextField();
var myMovieClip:MovieClip = new MovieClip();
var mySprite:Sprite = new Sprite();
```

- Think of the display list as a tree of all the visual objects in your movie. At the base of the tree is the stage. On the stage, you might have several movie clips or other types of visual objects (text fields, shapes, etc.). Within those movie clips might be other movie clips or other visual objects, and within those, others.
The Display List

- The difference between how the display list works in AS 2 and AS 3 is that in AS 2, when you created a movie clip, you had to specify where it would be on the tree at creation time.

- In AS 3, when you create a new movie clip, sprite, or any other display object, it is not automatically added to the display list.

- Your document class represents the base of the tree.

- To make a sprite or any other display object visible, you call the `addChild` method to the scene.

```javascript
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xff0000);
mySprite.graphics.drawCircle(0, 0, 40);
mySprite.graphics.endFill();
addChild(mySprite);
```
To remove an object from the display list, call `removeChild`, passing in a reference to the actual child object.

But removing a child does not destroy it!

You can remove an object from one movie clip and attach it to another one, in the exact same state it was in when you removed it.

In fact, you don’t even need to remove it. Because a child can have only one parent, adding it to another parent automatically removes it from the first.
The following class demonstrates reparenting:

```actionscript
package {
    import flash.display.Sprite;
    import flash.events.MouseEvent;
    public class Reparenting extends Sprite {

        private var parent1:Sprite;
        private var parent2:Sprite;
        private var ball:Sprite;

        public function Reparenting() {
            init();
        }

        private function init():void {
            parent1=new Sprite();
            addChild(parent1);
            parent1.graphics.lineStyle(1,0);
            parent1.graphics.drawRect(-50,-50,100,100);
            parent1.x=60;
            parent1.y=60;

            parent2=new Sprite();
            addChild(parent2);
            parent2.graphics.lineStyle(1,0);
            parent2.graphics.drawRect(-50,-50,100,100);
            parent2.x=170;
            parent2.y=60;

            ball=new Sprite();
            parent1.addChild(ball);
            ball.graphics.beginFill(0xff0000);
            ball.graphics.drawCircle(0,0,40);
            ball.graphics.endFill();
            ball.addEventListener(MouseEvent.CLICK,onBallClick);

            public function onBallClick(event:MouseEvent):void {
                parent2.addChild(ball);
            }
        }
    }
}
```
The class creates three sprites: parent1, parent2, and ball. The parent sprites are added to the display list directly, and squares are drawn in them. The ball sprite is added to parent1, so it is now on the display list and visible.

- When the ball is clicked, it is added to parent2. Note that there is no code to change its x, y position.
- It moves because now it is within a different sprite, which is at a different position.
Subclassing Display Objects

Since `attachMovieClip` is no more in AS 3, how do you get a movie clip symbol from the Flash CS3 IDE library into your movie at runtime?

The answer is with a class that extends either the Sprite or MovieClip class.

This concept is best shown through an example:

1. Create a new FLA, and draw some content on the stage.
2. Select the content and press F8 to convert it to a symbol.
3. In the Convert to Symbol dialog, give the symbol a name, and choose Movie clip as the type.
Now, say the class name you entered was Ball. In your document class, or right on the timeline, you can write the following:

```javascript
var ball:Ball = new Ball();
addChild(ball);
```

This causes the library symbol to be created on stage.

We’ll take another look at the reparenting example just given.

We created a sprite called parent1 with the following code:

```javascript
parent1.graphics.lineStyle(1, 0);
parent1.graphics.drawRect(-50, -50, 100, 100);
```

We then created another sprite, parent2, which had the same box drawn with the same lines of code.
Subclassing Display Objects

- we’ll make a class called ParentBox that extends Sprite.
- In that class, we’ll have the code that draws the box.

```
package
{
    import flash.display.Sprite;
    public class ParentBox extends Sprite
    {
        public function ParentBox()
        {
            init();
        }
        private function init():void
        {
            graphics.lineStyle(1,0);
            graphics.drawRect(-50, -50, 100, 100);
        }
    }
}
```
Then we’ll alter the class to create two ParentBoxes, rather than two Sprites.

```java
package
{
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class Reparenting2 extends Sprite
    {
        private var parent1:ParentBox;
        private var parent2:ParentBox;
        private var ball:Sprite;

        public function Reparenting2() {
            init();
        }
    }

    private function init():void {
        parent1=new ParentBox();
        addChild(parent1);
        parent1.x=60;
        parent1.y=60;

        parent2=new ParentBox();
        addChild(parent2);
        parent2.x=170;
        parent2.y=60;

        ball = new Sprite();
        parent1.addChild(ball);
        ball.graphics.beginFill(0xff0000);
        ball.graphics.drawCircle(0,0,40);
        ball.graphics.endFill();
        ball.addEventListener(MouseEvent.CLICK, onBallClick);
    }

    public function onBallClick(event:MouseEvent):void {
        parent2.addChild(ball);
    }
}
```
User Interaction

- User interaction is based on user events, and these generally come down to mouse events and keyboard events.
Mouse Events

- Any object has to specifically add itself as a listener for them.
- For mouse events to fire in AS 3, it is required that the mouse cursor is over some visible content of the display object.
- This is different than in AS 2, where mouseUp, mouseDown, and mouseMove would fire no matter where the mouse cursor was.
- Here are the available mouse event properties of the MouseEvent class:
  - CLICK
  - MOUSE_DOWN
  - MOUSE_OUT
  - MOUSE_UP
  - ROLL_OUT
  - DOUBLE_CLICK
  - MOUSE_MOVE
  - MOUSE_OVER
  - MOUSE_WHEEL
  - ROLL_OVER
The following code will trace each mouse event as it occurs on the sprite it creates.

```javascript
package {
import flash.display.Sprite;
import flash.events.MouseEvent;
public class MouseEvents extends Sprite {
    public function MouseEvents()
    {
        init();
    }  
    private function init():void
    {
        var sprite:Sprite=new Sprite;
        addChild(sprite);
        sprite.graphics.beginFill(0xff0000);
        sprite.graphics.drawCircle(0,0,50);
        sprite.graphics.endFill();
        sprite.x=stage.stageWidth/2;
        sprite.y=stage.stageHeight/2;
        sprite.addEventListener(MouseEvent.CLICK, onMouseEvent);
        sprite.addEventListener(MouseEvent.DOUBLE_CLICK, onMouseEvent);
        sprite.addEventListener(MouseEvent.MOUSE_DOWN, onMouseEvent);
        sprite.addEventListener(MouseEvent.MOUSE_MOVE, onMouseEvent);
        sprite.addEventListener(MouseEvent.MOUSE_OUT, onMouseEvent);
        sprite.addEventListener(MouseEvent.MOUSE_OVER, onMouseEvent);
        sprite.addEventListener(MouseEvent.MOUSE_UP, onMouseEvent);
        sprite.addEventListener(MouseEvent.MOUSE_WHEEL, onMouseEvent);
        sprite.addEventListener(MouseEvent.ROLL_OUT, onMouseEvent);
        sprite.addEventListener(MouseEvent.ROLL_OVER, onMouseEvent);
    }
    public function onMouseEvent(event:MouseEvent):void
    {
        trace(event.type);
    }
}
```
Mouse Position

- In addition to mouse events, two very important properties are the current location of the mouse pointer: mouseX and mouseY.
- These are properties of a movie clip, and the values returned are the mouse’s position in relation to the registration point of that clip.
- For example, if you have a sprite named sprite sitting at 100, 100 on the stage, and the mouse pointer is at 150, 250
  - mouseX is 150.
  - mouseY is 250.
  - sprite.mouseX is 50.
  - sprite.mouseY is 150.
Keyboard Events

- There are only two keyboard events:
  - KEY_DOWN
  - KEY_UP

```java
package {
    import flash.display.Sprite;
    import flash.events.KeyboardEvent;
    public class KeyboardEvents extends Sprite {
        public function KeyboardEvents() {
            init();
        }

        private function init():void {
            stage.addEventListener(KeyboardEvent.KEY_DOWN,onKeyboardEvent);
            stage.addEventListener(KeyboardEvent.KEY_UP,onKeyboardEvent);
        }

        public function onKeyboardEvent(event:KeyboardEvent):void {
            trace(event.type);
        }
    }
}
```
an event handler gets passed an event object, which contains data about the event that just occurred. In a keyboard event, two properties relate to what key was involved with the event: charCode and keyCode.

- The charCode property gives you the actual character that represents the key that was just pressed.
- If the user pressed the “a” key on the keyboard, charCode would contain the string “a”.
- If the user was also holding down the Shift key at the time, charCode would then contain “A”.
Key Codes

- The keyCode character contains a number that represents the physical key that was pressed.
- If the user pressed the “a” key, keyCode would contain the number 65.
- If the user pressed Shift first and then “a”, you would get two keyboard events: one for Shift (keyCode 16) and then one for “a” (65).
- The flash.ui.Keyboard class also has some properties to stand in for some of the nonalphanumeric keys, so you don’t have to memorize them.
package {
  import flash.display.Sprite;
  import flash.events.KeyboardEvent;
  import flash.ui.Keyboard;
  public class KeyCodes extends Sprite {
    private var ball:Sprite;
    public function KeyCodes() {
      init();
    }
    private function init():void {
      ball=new Sprite ;
      addChild(ball);
      ball.graphics.beginFill(0xfff0000);
      ball.graphics.drawCircle(0,0,40);
      ball.graphics.endFill();
      ball.x=stage.stageWidth/2;
      ball.y=stage.stageHeight/2;
      stage.addEventListener(KeyboardEvent.KEY_DOWN,onKeyboardEvent);
    }
  }
public function onKeyboardEvent(event:KeyboardEvent):void {
    switch (event.keyCode) {
        case Keyboard.UP:
            ball.y-=10;
            break;
        case Keyboard.DOWN:
            ball.y+=10;
            break;
        case Keyboard.LEFT:
            ball.x-=10;
            break;
        case Keyboard.RIGHT:
            ball.x+=10;
            break;
        default:
            break;
    }
}
Summary

- You now know about frame loops, events, listeners, handlers, and the display list.
- We also review classes, objects, and basic user interaction.