Actionscript 3.0 Basics

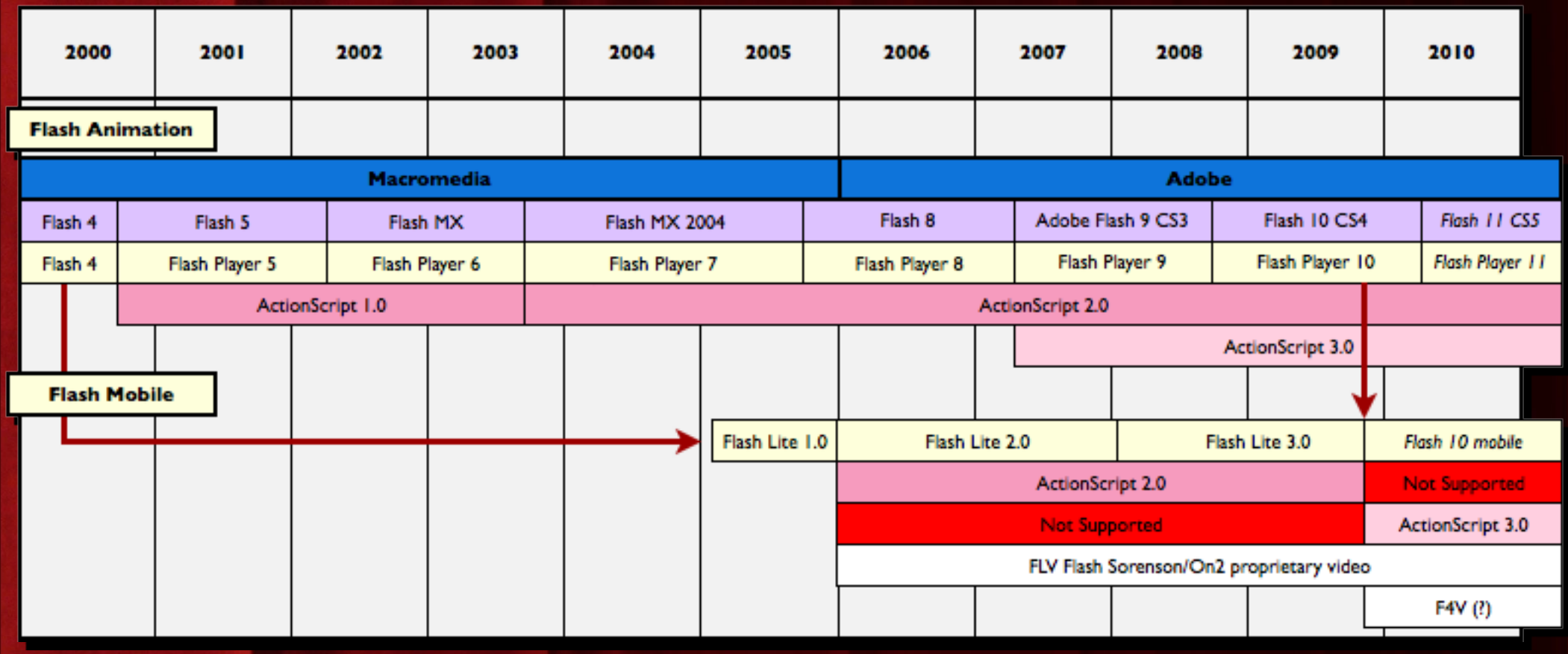
1 – Syntax, OOP, Data Structures

Flash Platform

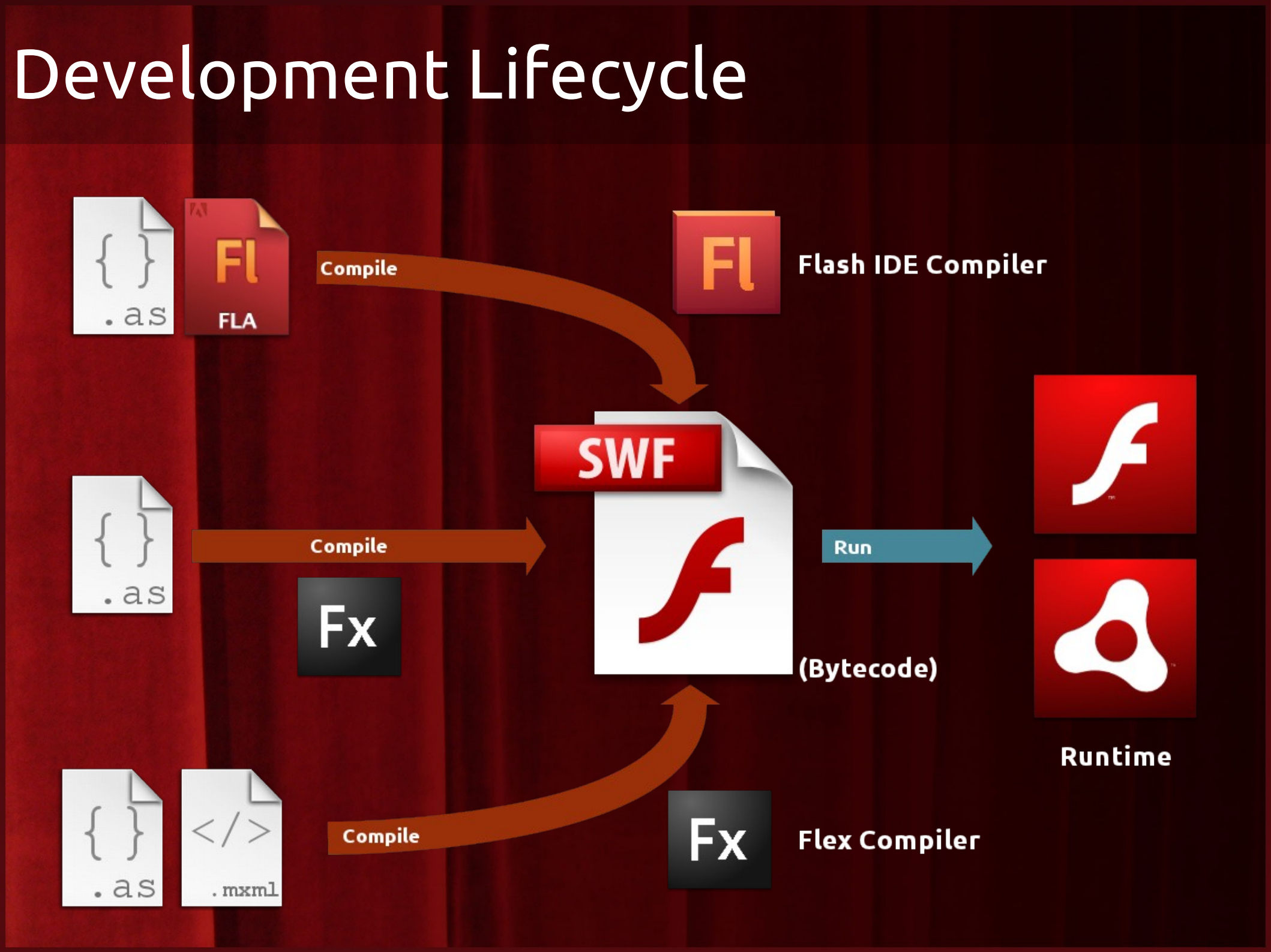
FLASH PLATFORM

Is a stack of web technologies that ease the development of Rich Internet Applications (RIAs)

Actionscript



Development Lifecycle



Actionscript

- ECMAScript Compliant (Javascript).
- Interpreted Language (AVM1/2).
- Compiled language (as bytecode).
- Dynamic typing (Also supports static typing).
- Object oriented.
- Event-Driven.
- Document Object Model (DOM) Level 3 Compliant.
- Multimedia Programming oriented.
- Virtual Machines:
  - Flash Player
  - Air Runtime
  - GNU Gnash (open source)

Strict vs. Standard

Strict Mode

↓

Static Typing

Standard Mode

↓

Dynamic Typing

Implicit declaration
Unsealed Objects

General Language Syntax

Structured Programming:

- Operators *(in order of precedence)*:

Primary
`[] {x:y} () f(x) new x.y x[y] <></> @ :: ..`
Postfix
`x++ x--`
Unary
`++x --x + - ~ ! delete typeof void`
Multiplicative
`* / %`
Additive
`+ -`
Bitwise shift
`<< >> >>>`
Relational
`< > <= >= as in instanceof is`
Equality
`== != === !==`
Bitwise
`&(AND) ^(NOT) | (OR)`
Logical
`&& (AND) || (OR)`
Ternary Conditional
`?:`
Assignment
`= *= /= %= += -= <<= >>= >>>= &= ^= |=`

- Conditional Statements:
  - if... else if … else
  - switch... case... default
- Iterations:
  - while , do... while
  - for ...
  - for each...in
  - for... in

General Language Syntax

Native Data Types:

- Object — Foundational
- int, uint, Number — Numeric
- Boolean, String, Date — Simple values
- Function, Class, Namespace, Error, RegExp — Language structures
- Array, Vector, XML + XMLList, Dictionary — Collections

General Language Syntax

Variables and constants:

```
1  var myVariable;              //Warning on strict
2  var myVariable:String;
3
4  const myConstant;            //Warning on strict
5  const myConstant:int;
6
7  var myVariable:String = "intial value";
8  const myConstant:int = 10;
9
10 var myVar1:String = "hello",
11     myVar2:int = 15,
12     myVar3:Array;
13
14 myVariable = "hola!"  //Error on strict if not
15                       //previously defined
16
17 delete myVariable; // undefined
```

Functions

In actionscript functions are first-class types. Functions are objects.

Basic function concepts
- Calling functions
- Defining your own functions
- Returning values from functions
- Nested functions

Function parameters
- Passing arguments by value or by reference
- Default parameter values
- The `arguments` object
- The `... (rest)` parameter

Function scope
- The scope chain
- Function closures

Functions

Calling functions:

```
1 //Basic syntax
2 aFunction(arg1,arg2);
3
4 //object syntax
5 aFunction.call(thisObject, arg1, arg2);
6 aFunction.apply(thisObject, [arg1,arg2]);
7
```

Functions

Defining your own functions:

```actionscript
//Function statement
function myFunction(arg1:uint, arg2:String):void
{
    trace(arg1, arg2);
}

//Function Expressions
myFunction:Function = function(arg1:uint, arg2:String):void
{
    trace(arg1, arg2);
}

//Function constructor -> DOES NOT WORK ON ACTIONSCRIPT!!
myFunction = new Function("arg1","arg2","trace(arg1, arg2);");
```

Functions

Returning values from functions:

```
1  //No return
2  function noReturn():void
3  {
4      trace("not returning");
5  }
6
7  //No return implicit
8  function noReturn()
9  {
10     trace("not returning");
11 }
12
13 //Return
14 function numDouble(num:Number):Number
15 {
16     return num*2;
17 }
```

Functions

Nested functions:

```
1  function getNameAndVersion():String
2  {
3      function getVersion():String
4      {
5          return "10";
6      }
7      function getProductName():String
8      {
9          return "Flash Player";
10     }
11     return (getProductName() + " " + getVersion());
12 }
13 trace(getNameAndVersion()); // Flash Player 10
```

Functions

Function parameters
- Passing arguments by value or by reference

In actionscript all arguments are passed by reference, but primitive types operations behave as if passed by value.

```
1 function passPrimitives(xParam:uint, yParam:uint):void
2 {
3     xParam++; // xParam += 1  or  xParam = xParam + 1;
4     yParam++;
5     trace(xParam, yParam);
6 }
7
8 var xValue:uint = 10;
9 var yValue:uint = 15;
10 trace(xValue, yValue);// 10 15
11 passPrimitives(xValue, yValue); // 11 16
12 trace(xValue, yValue);// 10 15
```

Functions

Function parameters
- Passing arguments by value or by reference

```
1  function passByRef(objParam:Object):void
2  {
3      objParam.x++;
4      objParam.y++;
5      trace(objParam.x, objParam.y);
6  }
7  var objVar:Object = {x:10, y:15};
8  trace(objVar.x, objVar.y); // 10 15
9  passByRef(objVar); // 11 16
10 trace(objVar.x, objVar.y); // 11 16
```

Functions

Default parameter values:

```
1 function defaultValues(x:uint, y:uint = 3, z:uint = 5):void
2 {
3     trace(x, y, z);
4 }
5 defaultValues(1); // 1 3 5
6 defaultValues(1,4); // 1 4 5
```

Functions

The `arguments` object:

```
function traceArgArray(x:uint):void
{
    for (var i:uint = 0; i < arguments.length; i++)
    {
        trace(arguments[i]);
    }
}

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

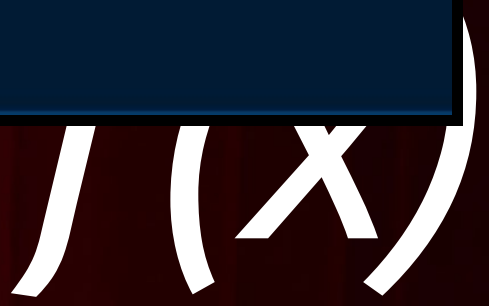
```
var factorial:Function = function (x:uint)
{
    if(x == 0)
    {
        return 1;
    }
    else
    {
        //recursion in anonymous function
        return (x * arguments.callee(x - 1));
    }
}

trace(factorial(5)); // 120
```

Functions

The `...(rest)` parameter:

```
1 function traceArgArray(x: uint, ... args)
2 {
3     for (var i:uint = 0; i < args.length; i++)
4     {
5         trace(args[i]);
6     }
7 }
8
9 traceArgArray(1, 2, 3);
10
11 // output:
12 // 2
13 // 3
```

The `arguments` object is not available if any parameter is named "arguments" or if you use the `...(rest)` parameter.



Functions

Closures:

A closure is a functional language feature that allows functions to retain the references from its lexical environment even after that environment is no longer in execution.

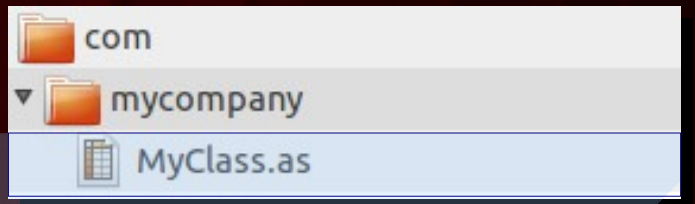
```
1 function adder(x:Number):Function
2 {
3     return function(y:Number):Number
4     {
5         return x+y;
6     }
7 }
8 var addTo2:Function = adder(2);
9 var addTo10:Function = adder(10);
10 trace(addTo2(3)) // 5
11 trace(addTo10(4)) // 14
```

Object-Oriented Programming

Class definitions
Class property attributes
 - Access control namespace attributes
 - static attribute
Class inheritance
Interfaces
Instantiation and access
Methods
 - Constructor methods
 - Static methods & Instance methods
 - Get and set accessor methods
 - Bound methods

Object-Oriented Programming

Class definition:




```
1 package com.mycompany
2 {
3     public class MyClass {
4
5     }
6
7     class MyPrivateClass {
8
9     }
10 }
```

Attribute	Definition
`dynamic`	Allow properties to be added to instances at run time.
`final`	Must not be extended by another class.
`internal` (default)	Visible to references inside the current package.
`public`	Visible to references everywhere.

Object-Oriented Programming

```
1  package com.mycompany
2  {
3      public class MyClass extends BaseClass implements AnInterface, OtherInterface
4      {
5          private var privateInstanceVariable:String = "initial value";
6          public var publicInstanceVariable;
7          public static const privateInstanceConstant:Number = 1;
8
9
10         static function classMethod():void{
11
12         }
13
14         function MyClass(){
15             super();
16         }
17
18         function internalMethod():void{
19
20         }
21
22         public function publicMethod(arg:Number):String
23         {
24             return this.privateInstanceVariable;
25         }
26
27         override public baseClassMethod(arg:int):void
28         {
29             super.baseClassMethod(arg);
30         }
31     }
32 }
```

Inheritance

Interface Inheritance

access modifiers

intialization values

class attributes and methods

constructor

methods

self reference

method override

ancestor reference

Object-Oriented Programming

Interfaces:

```
1  interface IAlpha
2  {
3      function foo(str:String):String;
4  }
5
6  interface IBeta
7  {
8      function bar():void;
9  }
10
11 class Alpha implements IAlpha, IBeta
12 {
13     public function foo(param:String):String {
14         return "foo:" + param;
15     }
16
17     public function bar():void {
18         // implentation
19     }
20 }
```

Object-Oriented Programming

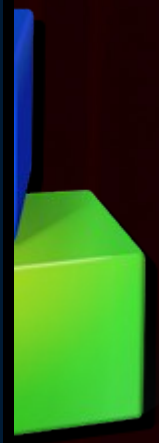
Instantiation and access.

```
1  //import class if not on the same namespace
2  import com.mycompany.MyClass
3
4  //intantiate class using keyword new
5  var myObject:MyClass = new MyClass();
6
7  //using introspection
8  var myClassObject:Class =
9      flash.utils.getDefinitionByName("com.company.MyClass");
10
11 var myObject:MyClass = new myClassObject();
12
13 //dot notation access
14 myObject.property;
15
16 //square bracket notation access
17 myObject["property"];
```

Object-Oriented Programming

Get and set accessor methods:

```
1  class GetSet
2  {
3      private var privateProperty:String;
4
5      public function get publicAccess():String
6      {
7          return privateProperty;
8      }
9
10     public function set publicAccess(setValue:String):void
11     {
12         privateProperty = setValue;
13     }
14 }
15
16 var myGetSet:GetSet = new GetSet();
17 trace(myGetSet.privateProperty); // error occurs
18
19 var myGetSet:GetSet = new GetSet();
20 trace(myGetSet.publicAccess); // output: null
21 myGetSet.publicAccess = "hello";
22 trace(myGetSet.publicAccess); // output: hello
```



Object-Oriented Programming

Bound methods (method closures):

```
1  class ThisTest
2  {
3      private var num:Number = 3;
4      function foo():void // bound method defined
5      {
6          trace("foo's this: " + this);
7          trace("num: " + num);
8      }
9      function bar():Function
10     {
11         return foo; // bound method returned
12     }
13 }
14
15 var myTest:ThisTest = new ThisTest();
16 var myFunc:Function = myTest.bar();
17 trace(this); // output: [object global]
18 myFunc();
19 /* output:
20 foo's this: [object ThisTest]
21 output: num: 3 */
```

Object Collections

Arrays
Vector (Typed Array)
Associative Arrays (POJO)
Dictionary

Object Collections

Arrays

```
1 // Arrays can hold values of any type
2 var myArray:Array = [ "one",2,new Object() ]; //Array literal
3
4 //access
5 myArray[0] == "one"; //true
6
7 //modify
8 myArray[0] = 1 // [1, 2, {}]
9
10 //add items
11 myArray[myArray.length] = 4
12 myArray.push(4)
13 // [1, 2, {}, 4]
14
15 myArray.unshift(0)
16 // [0, 1, 2, {}, 4]
17
18 //Check size
19 myArray.length // 5
20
21 //remove items
22 delete myArray[1]; // [0, undefined, 2, {}, 4]
23 myArray[1] = null; // [0, null, 2, {}, 4]
24 myArray.splice(1,1); // [0, 2, {}, 4]
```

Object Collections

Vector (Typed Arrays)

Vectors provide type information that allows the compiler to optimize the bytecode.

Availability: **Flash Player 10 +**

```
1 // Vectors can hold values of an specific type defined
2 var v:Vector.<String> = new <String>["zero","one", "two"]; // Vector literal
3
4 // type checking
5 v[v.length] = 3; // Error
```

Object Collections

Associative Arrays (POJO)

```
1  // Objects are associative arrays,
2  // they map strings keys to values
3  var obj:Object = {
4      key1:"value1",
5      key2:2
6  }; //object literal
7
8  // add items
9  obj["some key with spaces"] = "some value"
10
11 // remove items
12 delete obj["some key with spaces"];
13 delete obj.key1;
14
15 trace(obj["some key with spaces"]) // some value
16 trace(obj.key1) // value1
17 trace(obj.some key with spaces) // Syntax error
```

Object Collections

Dictionary (flash.utils.Dictionary)

```as3
// Dictionaries Map object keys with object values
import flash.utils.Dictionary;

var d:Dictionary = new Dictionary();

var obj1:Object = {name:"one"};
var obj2:Object = {name:"two"};
var obj3:Object = {name:"three"};

var value1:Object = {name:"value one"};
var value2:Object = {name:"value two"};
var value3:Object = {name:"value three"};

// add
d[obj1] = value1;
d[obj2] = value2;
d[obj3] = value3;

// access
d[obj1] // {name: "value one"}

// remove
delete d[obj1];
```

```as3
//iterate
for (var key:Object in d)
{
    trace(key);
    trace(obj[key]);
    trace();
}
/*
{name:"one"}
{name:"value one"}

{name:"two"};
{name:"value two"};

{name:"three"};
{name:"value three"};
*/
```

Thanks!

github.com/matix/as3basics

matias.figueroa@globant.com
@matixfigueroa