

AS34J

ActionScript for Java Developers

James Ward

www.jamesward.com

@_JamesWard

Flex Intro

- Flex is a set of libraries for developing Flash applications
- Flex programs are written in ActionScript3 and MXML
- MXML is a declarative markup language
- preprocesses into AS3

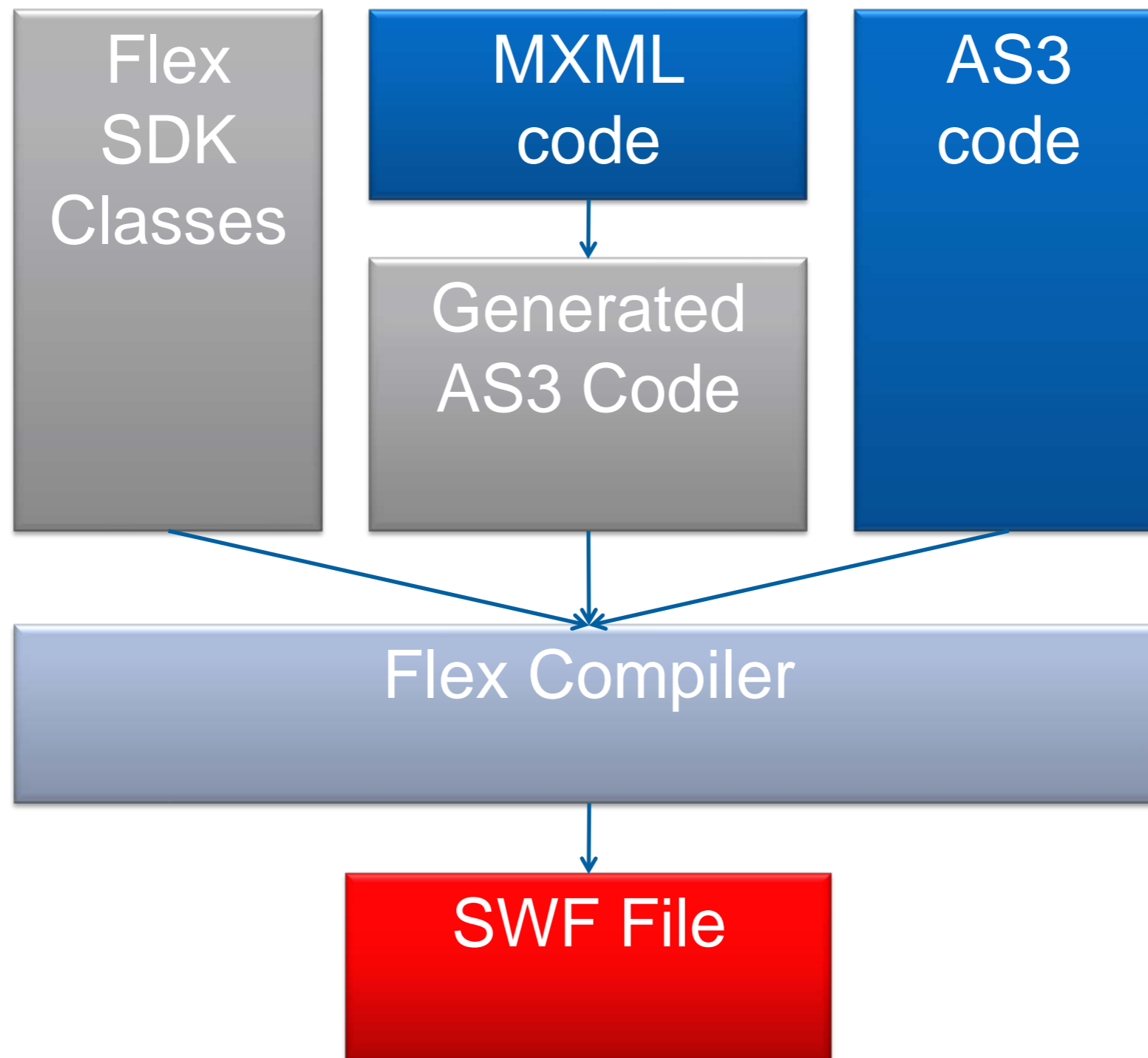
```
<mx:Object id="o" foo="bar"/>
```

```
var o:Object = new Object();  
o.foo = "bar";
```

MXMLC



From Flex code to Flash App



ActionScript3 Intro

- Java and AS3 are remarkably similar in many ways
- Both have
 - classes
 - interfaces
 - object-oriented style of programming
 - types
 - ...
- But there are some differences...

same syntax?

```
package foo;

import somepackage.*;

public class Foo
    extends FooParent
    implements IFoo
{
    private String blah = "blah";

    /**
     * bar performs important string-
     * setting functionality
     *
     * @param b the String to be set
     */
    public boolean bar(String b) {
        blah = b;
        return true;
    }
}
```

```
package foo {

import somepackage.*;

public class Foo
    extends FooParent
    implements IFoo
{
    private var blah:String = "blah";

    /**
     * bar performs important string-
     * setting functionality
     *
     * @param b the String to be set
     */
    public function bar(String b):Boolean {
        blah = b;
        return true;
    }
}
```

same syntax?

```
package foo;

import somepackage.*;

public class Foo
    extends FooParent
    implements IFoo
{
    private String blah = "blah";

    /**
     * bar performs important string-
     * setting functionality
     *
     * @param b the String to be set
     */
    public boolean bar(String b) {
        blah = b;
        return true;
    }
}
```

```
package foo {

import somepackage.*;

public class Foo
    extends FooParent
    implements IFoo
{
    private var blah:String = "blah";

    /**
     * bar performs important string-
     * setting functionality
     *
     * @param b the String to be set
     */
    public function bar(String b):Boolean {
        blah = b;
        return true;
    }
}
```

Agenda

- Syntax
- Properties
- Dynamic Behavior
- Classes
- Functions
- Events
- Primitives & Data Structures
- Puzzlers

The Wages of Syntax

Type Syntax

- Java does this

```
public int blah;
```

```
public Object foo = new Object();
```

var Syntax:Type

- Java does this

```
public int blah;  
public Object foo = new Object();
```

- AS3 does it this way

```
public var blah:int;  
public var foo:Object = new Object();
```

Undefined != null

- Java has no 'undefined' concept

```
Object foo;  
Number num;  
System.out.println(foo);  
// outputs 'null'  
System.out.println(num);  
// outputs 'null'
```

- AS3 has undefined, null, NaN

```
var foo;  
var bar:Object;  
var num:Number;  
trace(foo);  
// outputs 'undefined'  
trace(bar);  
// outputs 'null'  
trace(num);  
// outputs 'NaN'
```

Package Declarations

- Java declares packages up top

```
package foo;  
class FooThing {  
}
```

- AS3 puts classes inside package block

```
package foo {  
class FooThing {  
}  
}
```

Scope

- In Java, scope is defined by the enclosing block:

```
// using i for both is fine - completely separate scopes
public void foo() {
    for (int i = 0; i < 10; ++i) {}
    for (int i = 0; i < 5; ++i) {}
}
```

- In AS3, scope is function-wide:

```
// causes a compiler warning
public function foo():void {
    for (var i:int = 0; i < 10; ++i) {}
    for (var i:int = 0; i < 5; ++i) {}
}
```

```
// better:
public function foo():void {
    var i:int;
    for (i = 0; i < 10; ++i) {}
    for (i = 0; i < 5; ++i) {}
}
```

Metadata

- Java annotations

```
@Foo
public class Bar {
    @Monkey
    String banana;
}
```

- AS3 metadata

```
[Foo]
public class Bar {
    [Monkey]
    var banana:String;
}
```

Semicolons

- Java;
- Requires;
- Semicolons;
- AS3
- Does
- Not
- Unless; You; Do; Multiple; Operations; On; A; Single; Line;
- (but readable code should use them)

final vs. const

- Java uses 'final' for constants

```
public final int FOO = 5;
```

- AS3 uses 'const'

```
public const FOO:int = 5;
```

Casting Call

- Java

```
float foo = 5;  
int fooInt = (int)foo;
```

- AS3

```
var foo:Number = 5;  
var fooInt:int = int(foo);
```

- and

```
var foo:Number = 5;  
var fooInt:int = foo as int;
```

Exceptional Handling

- Java throws up

```
public void foo() throws MyException
{
    try {
        // really awesome code
    } catch (Exception e) {
        throw new MyException("AAAUUUGGGHHHH!");
    }
}
```

- AS3 throws without declaration

```
public function foo():void
{
    try {
        // really awesome code
    } catch (var e:Error) {
        throw new Error("AAAUUUGGGHHHH!");
    }
}
```

Generics

- Java has generics and typed collections

```
List<FooType> list = new ArrayList<FooType>();  
list.add(new FooType());  
FooType foo = list.get(0);
```

- AS3 does not
- ...but it does have a typed Vector class

```
var vec:Vector.<FooType> = new Vector.<FooType>();  
vec[0] = new FooType();  
var foo:FooType = vec[0];
```

XML Handling

- Java processes XML through libraries (many of them)
 - JAXP, JAXB, SAX, Xerces, JDOM, ...
- AS3 has 'e4x'

```
var myXML:XML = <Grob>gable</Grob>;  
trace(myXML.Grob);  
// outputs 'gable'
```

- e4x has queries, XML manipulation, ...

Proper Tease

Properties: Java

- Java has JavaBeans conventions for getters/setters

```
public class JavaProps {  
    public int blah;  
    private int foo;  
    public void setFoo(int value) {  
        foo = value;  
    }  
    public int getFoo() {  
        return foo;  
    }  
}
```

```
JavaProps props = new JavaProps();  
props.blah = 5;  
props.setFoo(5);
```

Properties: AS3

- AS3 has properties as first-class citizens via get/set

```
public class AS3Props {  
    public var blah:int;  
    private var _foo:int;  
    public function set foo(value:int):void {  
        _foo = value;  
    }  
    public function get foo():int {  
        return _foo;  
    }  
}
```

```
AS3Props props = new AS3Props();  
props.blah = 5;  
props.foo = 5;
```

Properties: AS3

```
public class AS3Props {  
    public var foo:int;  
}
```

is equivalent to

```
public class AS3Props {  
    private var _foo:int;  
    public function set foo(value:int):void {  
        _foo = value;  
    }  
    public function get foo():int {  
        return _foo;  
    }  
}
```

Properties: AS3

- Variable -> set/get change can be made without affecting public API
- In Java, need to define set/get functions up front
 - Can't have users rewrite code later on change of foo to setFoo/getFoo
- In AS3, can replace variable with set/get functions any time
- *Note:* cannot override instance variables in subclasses
 - set/get is a good convention to follow in general, when subclassing is expected

Property Access

- Java accesses named fields checked at compile time

```
FooType foo = new FooType();  
foo.x = 5;  
// compiler error if x not in FooType
```

- AS3 provides the same syntax

```
var foo:FooType = new FooType();  
foo.x = 5;  
// compiler error if x not in FooType  
foo["x"] = 5;  
// runtime error
```

- But Object or untyped is more flexible

```
var foo:Object = new Object();  
var bar:* = new Object();  
foo.x = 5;  
bar.x = 5;  
// no compiler errors
```

Property Access

- AS3 allows access to properties via Strings

```
var foo:Object = new Object();  
foo["x"] = 5;  
var arg:String = "x";  
foo[arg] = 5;
```

- Useful for access of arbitrary fields known only at runtime

Dynamic Behavior: maps

- In Java, hashmapping is handled by libraries

```
HashMap map = new HashMap();  
map.put("foo", 5);  
Object blah = new Object();  
map.put(blah, 14.0);
```

- In AS3, every Object is a String map

```
var map:Object = new Object();  
map["foo"] = 5;
```

- ... but Dictionary is used for Object mapping

```
var map:Dictionary = new Dictionary();  
var blah:Object = new Object();  
map[blah] = 14.0;
```

Dynamism: maps

- In AS3, arbitrary classes can also perform mapping, if declared dynamic:

```
public dynamic class MyClass {  
    // [interesting behavior deleted]  
}  
var map:MyClass = new MyClass();  
map.foo = 5;
```

Dynamic Arrays

- In Java, Arrays are declared with a set size

```
Array myArray = new Array(1);  
myArray[0] = new Object();  
myArray[1] = new Object();  
// throws really long ArrayIndexOutOfBoundsException
```

- In AS3, Arrays are dynamic

```
var myArray:Array = [];  
myArray[0] = new Object();  
myArray[1] = new Object();  
// no problem
```

Untyped properties

- Java doesn't do untyped properties

```
Object foo;  
bar;  
// Compiler pukes on bar declaration
```

- AS3 allows untyped, will be void-typed

```
var foo:Object;  
var bar;  
var blah:*;  
// bar and blah will both be void-typed
```

Object Creation

- Java populates variables through construction, or through manual sets after construction

```
public class FooObject {  
    int blah;  
}  
FooObject foo = new FooObject();  
foo.blah = 5;
```

- AS3 allows JSON-like setting of properties on Objects

```
var foo:Object = {blah:5};
```

A
Class
Act

Constructor Permissions

- Java allows permissions on constructors

```
public class FooObject {  
    private FooObject() {}  
}
```

```
FooObject foo = new FooObject();  
// Compiler error if in another class
```

- AS3 constructors must always be public

```
public class FooObject {  
    private function FooObject() {}  
}
```

```
// Compiler error
```

Properties and Interfaces

- Java allows properties and constants on interfaces

```
public interface IFoo {  
    public int blah = 5;  
    public final int VAL = 7;  
}
```

- (by the way, these are implicitly final and static)
- AS3 ... does not

Abstract Classes

- Java allows abstract classes

```
public abstract class FooObject {  
    public abstract void foo();  
}
```

- AS3 ... does not

Namespaces

- Java has permissions
 - public, private, protected, package-private
- ... But not 'namespaces'
- AS3 has namespaces
 - public, private, protected, internal

Namespaces

```
package example {  
    public namespace myNamespace = "http://foo.com/blah";  
}
```

```
package example.a {  
    import example.myNamespace;  
    public class Foo {  
        myNamespace static var bar:Object;  
    }  
}
```

```
package example.b {  
    import example.myNamespace;  
    use namespace myNamespace;  
    public class Blah {  
        public function Blah() {  
            trace(Foo.bar);  
        }  
    }  
}
```

Form
vs.
Function

Functions as Objects

- Java
 - methods are members of classes
- AS3
 - functions are Objects
 - in fact, function extends Object

Functions and Packages

- Java has functions only at the class level

```
package foo;
```

```
public class MyClass {  
    public void blah() {}  
}
```

- AS3 has functions at package and class level

```
package foo {  
    public function bar():void {}  
}
```

```
package foo {  
    public class MyClass {  
        public function blah():void {}  
    }  
}
```

- (single file may have only one public class or function)

Function Overloading

- Java can have multiple overloads of same function

```
public void blah() {}  
public void blah(String bar) {}
```

- AS3 can have only one

```
public function blah():void {}  
public function blah(bar:String):void {}  
// compiler error
```

- but: default parameter values provide workaround

```
public function blah(bar:String = ""):void {}  
blah();  
blah("Grob");
```

Default values

- Java does not allow parameter values in functions
- AS3 does, with some options

```
public function blah(foo:int, bar:String = ""):void { }
```

```
blah(5);
```

```
blah(5, "Groob");
```

... args

- AS3 supports variable number of args

```
public function blah(... args):void {}
```

```
blah(5);
```

```
blah(5, "Groob");
```

```
public function blah(first:int, ... rest):void {}
```

```
blah(5);
```

```
blah(5, "Groob", "blah");
```

Anonymous Functions

- Java has methods only in context of classes
- AS3 can have anonymous functions, declared inline

```
var f:Function = function():void {};  
f();
```

Function Passing

- Java method accessed is only through objects
- not method-passing... (possible with Reflection)
- AS3 functions are objects and can be passed around willy-nilly

```
public class Foo {  
    public static function blah():void {}  
}
```

```
bar.blahRef = Foo.blah;  
bar.blahRef();
```

```
public function groob(func:Function):void {  
    func();  
}
```

```
groob(Foo.blah);
```

Overrides

- Java overriding is automatic
- same method signature in subclass overrides superclass method

- AS3 requires 'override' keyword

```
public class Foo { // extends Object
    override public function toString():String {}
}
```

In
any
Event

Events: Java

- Java: Manually set up event listening and dispatching
- create addListener()/removeListener() methods
- manage listener list
- when appropriate, iterate through listeners and send out events

```
public void addEventListener(IFoo listener) {  
    // manage listeners  
}  
public void removeEventListener(IFoo listener) {  
    // manage listeners  
}  
public void someMethod() {  
    // dispatch event to listener list  
}
```

Events: AS3

- AS3: Most visual items are event dispatchers already
- Your class needs to expose the event type it dispatches
 - Done via metadata
- Listeners just add themselves via `addEventListener()`
- Flash automatically manages listeners, dispatches events

```
[Event(name="someEvent", type="SomeEventType")]
```

```
public void someMethod() {  
    dispatchEvent("someEvent");  
}
```

Primitives & Data Structures

Numbers

- AS3 has 3 number types:
 - uint - 32 bit unsigned integer
 - int - 32 bit signed integer
 - Number - IEEE-754 double-precision floating point
 - NO long integer / 64-bit integer!!

Puzzlers

This Function

What is the output for the following code:

```
var f:Function = function(that:Object):void {  
    trace(this === that);  
}  
f(this);
```

- A)
true
- B)
false
- C) Runtime error
- D) Compile error

This Function

What is the output for the following code:

```
var f:Function = function(that:Object):void {  
    trace(this === that);  
}  
f(this);
```

A)
true

B)
false

C) Runtime error

D) Compile error

This Function

- Anonymous functions aren't in the same scope as a function on a class
- You can use `Function.call()` or `Function.apply()` to overwrite the "this" within a function call.

Array Casting

What is the output for the following code:

```
var a:Array = ['a', 'b', 'c'];  
trace(Array(a)[0]);
```

A)
a

B)
a,b,c

C) Runtime error

D) Compile error

Array Casting

What is the output for the following code:

```
var a:Array = ['a', 'b', 'c'];  
trace(Array(a)[0]);
```

A)
a

B)
a,b,c

C) Runtime error

D) Compile error

Array Casting

Casting an object to an Array (when not using the "as" operator) a **new Array** is created with the first element being the Array that was "casted".