ActionScript 3.0

WARREN FERNANDES

APRIL 1, 2011

Outline

- About
- Usage
- AVM
- Goals
- Features
- Language and Syntax
- Object-oriented programming principles

About ActionScript 3.0

- Object-Oriented Programming Language
- Originally developed by Macromedia Inc.
- Currently owned by Adobe Systems
- Dialect of the ECMAScript,
  - Supports 3rd edition (ECMA-262)
  - Supports functionality in 4th edition (ECMA for XML or E4X)

ECMAScript

ActionScript 3.0

- International scripting language standardization
- Used for client-side scripting on the web
- JavaScript, JScript, and ActionScript

Usage

- Enhances the web experience through rapid development of rich internet applications
- Used in software targeting Adobe Flash Player, Adobe Flex, and AIR platform
- Goes beyond scripting capabilities by supporting creation of complex applications with large data sets and object-oriented, reusable code bases
- Click on buttons for examples

Tour de Flex Flex Store

- ActionScript is executed by the AVM

AVM

- ActionScript Virtual Machine or AVM
  - Built into the Flash Player
  - ActionScript is compiled and run on AVM
- Versions:
  - AVM1 – executes legacy ActionScript code
  - AVM2 – executes ActionScript 3.0 code

AVM2

- Built from scratch just for ActionScript 3.0
- Highly optimized and improves performance by 10 times compared to previous versions
- Supported in Flash Player 9.0 and higher
- These flash players also support AVM1 for backward compatibility

Design Goals

- Safety
  - Supports type safety.
- Simplicity
  - Intuitive for developers to be able to read and write programs.
- Performance
  - Allows complex programs to perform efficiently and responsively. 10 times increase in performance.
- Compatibility
  - Backward compatibility. AVM1 and legacy AS code
  - Forward compatibility. ECMAScript for XML (E4X)

Features

- ActionScript 3.0 has two main features
  - The core language
  - Flash Player API
- Core Language
  - Statements, conditions, expressions, loops, types
- Flash Player API
  - Classes that represent and provide access to Flash Player specific functionality

- Packages
  - Allows bundling of class definitions together to facilitate code sharing and avoid naming conflicts
- Namespaces
  - Allow control over visibility of individual properties and methods
- Variables
  - To declare a variable, the `var` statement must be used with the variable name and assign a type using the colon (`:`) operator

```
var i: int;
```

  - Untyped variables are declared as `var i:*;` or `var i;` They hold the value `undefined`

- Data Types
  - The primitive data types include Boolean, int, Null, Number, String, uint, and void
  - Other data types are Object, Array, Date, Error, Function, RegExp, XML, and XMLList
  - Number is used to store integers larger than int and uint and for floating point numbers
  - void type contains the value `undefined`. This can only be assigned to untyped variables. This is usually used as a return type.
    - Undefined type was added into the language based on ECMAScript compliance

- Objects
  - Collections of properties
  - These properties are containers that hold data as well as functions or other objects. If a function is attached to an object like this, it is called a method
- Loops – `for..in`
  - This is a new loop in addition to the other standard loops
  - This loop iterates through the properties of an object, or the elements of an array.

```
var myObj: Object = {id: 2002, fname: "Warren"};
for (var i:String in myObj){
        trace (i +": " + myObj[i]);}
```

Output:
id: 2002
fname: Warren

- Functions
  - Functions are defined in two ways: function statements and function expression
  - Function Statements are the standard, preferred technique in defining functions

```
function myFunc(param:String){
        // function body
}
```

  - Function expressions are more dynamic and define anonymous functions. It is used with an assignment statement

```
var tp:Function = function (param:String){
        // function body
};
```

Classes

- Classes
  - Represented by class objects that store the class properties and methods

```
public class Shape{
        var visible:Boolean = true;
}
var myCircle: Shape = new Shape();
```

  - Other attributes, access modifiers, variables, and methods are similar to other OOP languages such as Java.
  - Class attributes – `final, dynamic, internal (default), public`
  - Class access modifiers – `internal(default), private, protected, public, static`

Interfaces

- Interface is a collection of method declarations that allows unrelated objects to communicate with one another
- To define an interface we use the `interface` keyword
- Only the `public` and `internal` modifiers can be used within an interface
- Interface definitions cannot be placed within a class or another interface
- A class uses an interface by using the `implements` keyword

Interfaces

- Example,

```
public interface IAlpha{
     function foo(str:String):String;
}
public interface IBeta{
     function bar():void;
}
class Alpha implements IAlpha, IBeta
{
     public function foo(param:String):String{}
     public function bar():void{}
}
```

By convention, we use 'I' in the beginning of the interface name

Multiple interfaces may be implemented

Inheritance

- Inheritance is a form of code reuse that allows us to develop new classes based on existing classes
- Key advantage is to reuse code from the base class while defining separate implementations for the subclass
- A subclass inherits a base class by using the keyword `extends`
- Subclass has an “IS-A” relationship with the base class

Inheritance


```
public class Shape {
      public function area():Number
      { return NaN; }
}
public class Circle extends Shape {
      private var radius:Number = 1;
      public override function area():Number
      { return (Math.PI * (radius * radius)); }
}
public class Square extends Shape {
      private var side:Number = 1;
      public override function area():Number
      { return (side * side); }
}

var cir:Circle = new Circle();
trace(cir.area());
var sq:Square = new Square();
trace(sq.area());
```

Subclass Circle inherits properties from Base class Shape by using the keyword extends

Output: 3.141592653589793

Output: 1

Encapsulation

- Encapsulation is the ability to hide and protect data
- It is implemented by applying access modifiers to entities
- The access modifiers are `private`, `protected`, `public`, `internal`, `static`

Encapsulation


```
package foo{
   public class Parent{
         public var str:String = "hello";
   }}
package bar{
   import foo.Parent;
   public class Child{
         public function getString(): String{
                return str;
         }
   }}
package {
   import bar.Child;
   public class Test{
         public function testApp(){
                var myChild:Child = new Child();
                trace(myChild.str);
                trace(myChild.getString());
         }
   }}
```

Notice in this example, every class is in a separate package

Error if str is not public

Error if str is private or internal

Polymorphism

- Polymorphism is the concept of referring to different derivations of a class in the same way, but getting the appropriate behavior of the referred class
- It is implemented by using the keyword `override`
- An inherited method overrides the behavior of the base class method

Polymorphism


```
public class Employee{
        public function work():void{
                trace("I am working");
        }
}
public class Manager extends Employee{
  public override function work():void{
        trace("I am managing");
  }
}
public class SalesPerson extends Employee{
  public function work():void{
        trace("I am selling");
  }
}
```

Subclass method to override base class behavior

No override specified

Polymorphism


```
var e1:Employee = new Manager();
e1.work();
```

Output: I am managing

```
var e2:Employee = new SalesPerson();
e2.work();
```

Output: I am working

Summary

- It is an object-oriented scripting language based on ECMAScript
- Its uniqueness lies in building complex Flash applications, which is used in web pages in the form of embedded SWF files

References

- Web:
  - http://en.wikipedia.org/wiki/ActionScript
  - http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html
- Book:
  - Programming Adobe ActionScript 3.0