

# **Object-Oriented Programming in ActionScript 3.0**

Peter Elst – Multi-Mania 2007

**What's new?**

- DisplayList API
- DOM3 Event Model
- ECMAScript for XML (E4X)
- Runtime error checking
- Regular Expressions
- Method closures

**Everything is a class**

## **ActionScript 2.0**

```
var mc:MovieClip =  
this.createEmptyMovieClip  
("mc",this.getNextHighestDepth());
```

## **ActionScript 3.0**

```
var mc:MovieClip = new MovieClip();
```

## **ActionScript 2.0**

```
var myMC:MovieClip = this.attachMovie  
("mc","myMC",this.getNextHighestDepth());
```

## **ActionScript 3.0**

```
var myMC:MC = new MC();  
this.addChild(myMC);
```

```
<?xml version="1.0">  
<addressbook>  
  <contact name="John">  
    <email>john@abc.com</email>  
  </contact>  
  <contact name="Wayne">  
    <email>wayne@xyz.org</email>  
  </contact>  
</addressbook>
```

## **ActionScript 2.0**

```
myXML.firstChild.childNodes[0].  
firstChild.nodeValue
```

## **ActionScript 3.0**

```
myXML..contact.(@name=="John").email
```

## **ActionScript 2.0**

```
import mx.utils.Delegate;  
myButton.addEventListener("click",  
    Delegate.create(this, onButtonClick));
```

## **ActionScript 3.0**

```
myButton.addEventListener  
(MouseEvent.CLICK, onButtonClick);
```

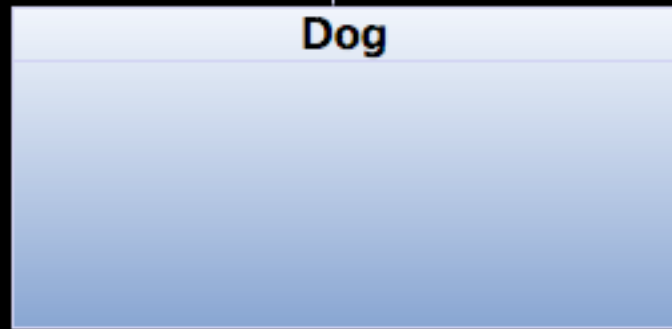
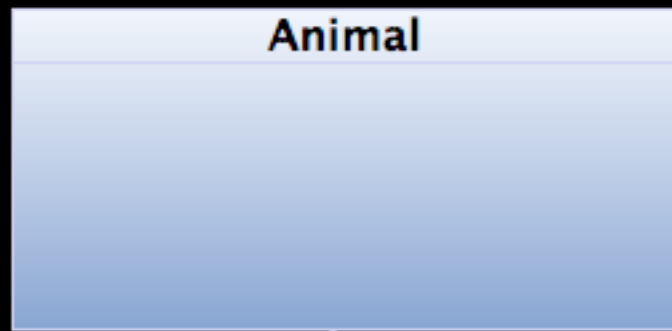
**How do you code it?**

- Flash CS3
- Flex Builder 2
- Flex SDK (free!)
- Open Source Editors (FlashDevelop, ...)

# **OOP concepts**

# Inheritance

- Allows classes to extend existing methods and properties of its superclass
- Requires override keyword to override methods or properties from superclass
- Final class modifier prevents it from being further sub-classed

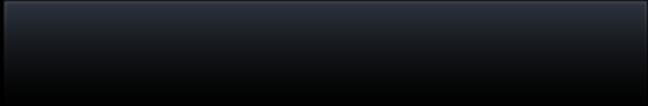
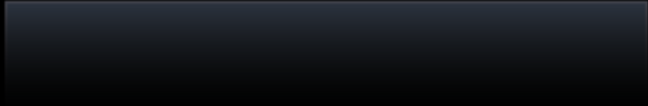


```
package com.peterelst.multimania {  
    public class Animal {  
        public function describe():String {  
            return "I'm an animal";  
        }  
    }  
}
```

```
package com.peterelst.multimania {  
    import com.peterelst.multimania.Animal;  
    public class Dog extends Animal {  
        override public function describe():String {  
            return "I'm a dog";  
        }  
        public function speak():String {  
            return "woof!";  
        }  
    }  
}
```

# Inheritance vs composition

- Composition does not extend a class but instantiates it at runtime
- Inheritance used for *is a* relationship, composition for a *has a* relationship
- Composition gives control over the creation and destruction of the back-end class



```
package com.peterelst.multimania {  
    public class Person {  
        private var face:Face;  
        public function Person() {  
            this.face = new Face();  
        }  
        public function smile():void {  
            this.face.smile();  
        }  
    }  
}
```

# Encapsulation

- Allows you to protect the inner workings of a class through the use of access modifiers
- Provide an API to interact with your class through getter/setter methods
- public - accessible everywhere  
private - only accessible within class  
protected - within class and derivative classes  
internal - accessible within same package

```
package com.peterelst.multimania {  
    public class Person {  
        private var _name:String;  
        public function get name():String {  
            return this._name;  
        }  
        public function set name(val:String):void {  
            this._name = val;  
        }  
    }  
}
```

```
var myPerson:Person = new Person();  
myPerson.name = "Peter";
```

# Polymorphism

- Allows different classes to respond to same method names with its own implementation
- Common method names make classes interchangeable at runtime
- Polymorphism can be enforced through implementing a common interface

```
import com.peterelst.multimania.*;
```

```
function animalTalk(instance:Animal):void {  
    trace(instance.describe());  
}
```

```
animalTalk(new Dog());  
animalTalk(new Cat());
```

# Interfaces

- Allows you to specify a set of methods that classes are required to implement
- Classes can implement multiple interfaces, interfaces can extend each-other
- Interfaces can be seen as contracts to be developed against, great for frameworks

# **Design Patterns**

# Observer pattern

- Allows objects to subscribe for updates to a class implementing the Observer pattern
- Event model similar to what you find in the `flash.events.EventDispatcher` class
- Provides loose coupling between dispatching and listening classes

```
package com.peterelst.multimania {
  public class Observer {
    private var _subscribers:Array = new Array();
    public function subscribe(obj:Object):void {
      _subscribers.push(obj);
    }
    public function unsubscribe(obj:Object):void {
      for(var i:uint=0; i< _subscribers.length; i++) {
        if(_subscribers[i] == obj) _subscribers.splice(i, 1);
      }
    }
    private function notify() {
      for(var i:uint=0; i<_subscribers.length; i++) {
        _subscribers[i].update();
      }
    }
  }
}
```

# Singleton pattern

- Allows a class to have just one instance accessible through a static method
- Singleton classes provide a global point of access to its instance
- ActionScript 3.0 does not allow a constructor to be marked as private

```
package com.peterelst.multimania {
  public class Singleton {
    private static var _instance:Singleton;
    private static var _allowInstance:Boolean;
    public function Singleton() {
      if(!_allowInstance) {
        throw new Error("use Singleton.getInstance()");
      }
    }
    public static function getInstance():Singleton {
      if(_instance == null) {
        _allowInstance = true;
        _instance = new Singleton();
        _allowInstance = false;
      }
      return _instance;
    }
  }
}
```

# Decorator pattern

- Allows you to extend a class at runtime using composition
- Typically a class instance gets passed to the constructor of the Decorator class
- The `flash.utils.Proxy` class enables routing of non-implemented methods to the decorated class instance

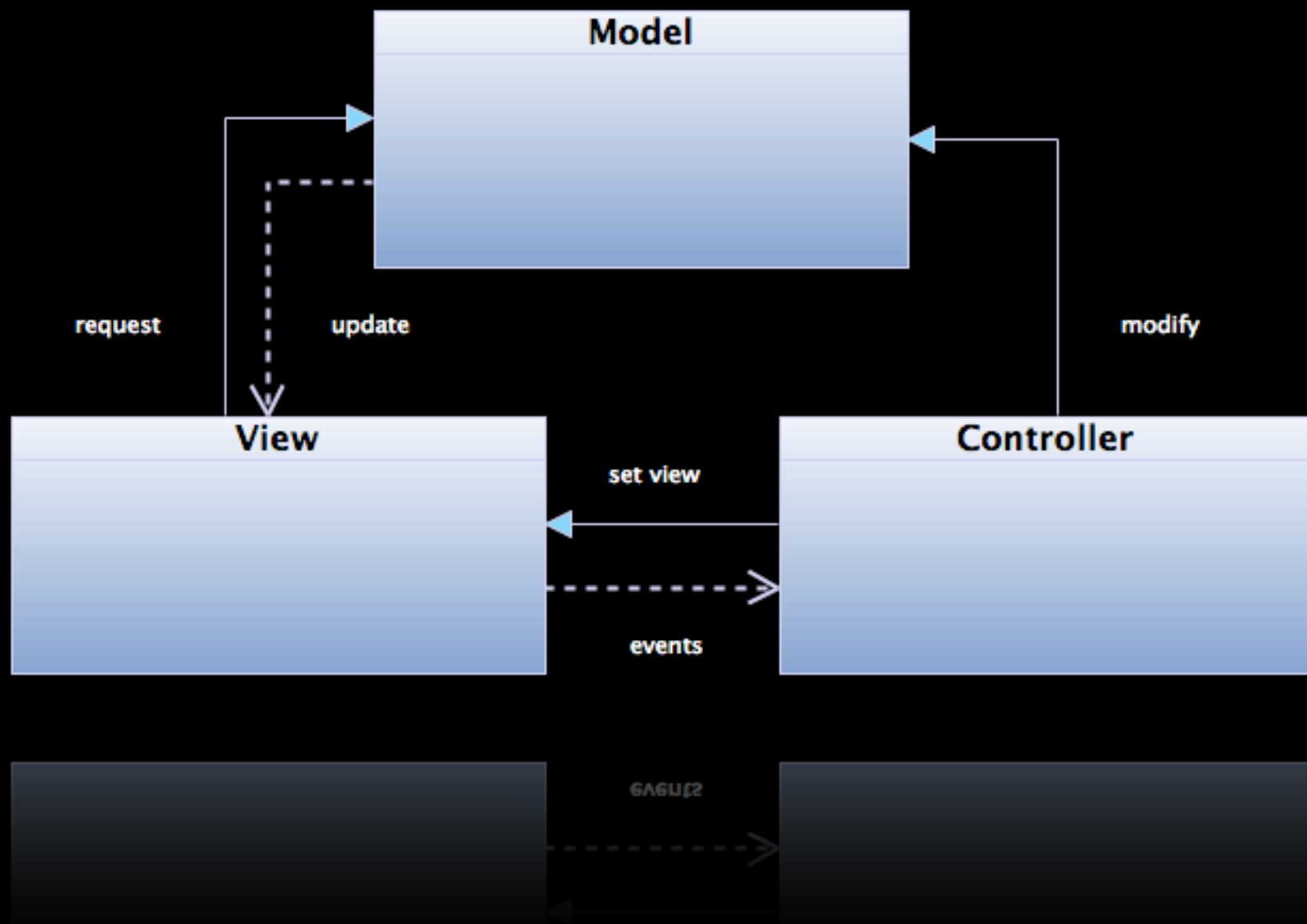
# flash.utils.Proxy

- ActionScript 3.0 approach to the `__resolve` method in ActionScript 2.0
- Class needs to extend `Proxy` and use `flash_proxy` namespace to override `callProperty`
- Class needs dynamic modifier to avoid compiler errors for calling non-implemented methods

```
package com.peterelst.multimania {  
  
    import flash.utils.Proxy;  
    import flash.utils.flash_proxy;  
  
    public dynamic class Decorator extends Proxy {  
        private var _decorated:Object;  
        public function Decorator(obj:Object) {  
            _decorated = obj;  
        }  
        flash_proxy override function callProperty  
        (method:*, ...args):* {  
            return _decorated[method].apply(_decorated, args);  
        }  
    }  
}
```

# Model-View-Controller

- MVC is an architectural pattern, implements Singleton, Observer and other patterns
- Allows you to separate data, view and interaction of your application
- Decoupling data from view allows for easy switching between different views



```
package com.peterelst.multimania {
  public class Model extends Observer {
    private static var _instance:Model;
    private static var _allowInstance:Boolean;
    public function Model() {
      if(!_allowInstance) {
        throw new Error("use getInstance()");
      }
    }
    public static function getInstance():Singleton {
      if(_instance == null) {
        _allowInstance = true;
        _instance = new Singleton();
        _allowInstance = false;
      }
      return _instance;
    }
    public function registerView(view:Object):void {
      super.subscribe(view);
    }
  }
}
```

```
package com.peterelst.multimania {  
    import flash.display.MovieClip;  
    public class View extends MovieClip {  
        private var _model:Object;  
        private var _controller:Object;  
        public function View(model:Object,controller:Object) {  
            _model = model;  
            _controller = controller;  
            _model.registerView(this);  
        }  
        public function update():void {  
        }  
    }  
}
```

```
package com.peterelst.multimania {  
  public class Controller {  
    private var _model:Object;  
    public function Controller(model:Object) {  
      _model = model;  
    }  
  }  
}
```

```
package com.peterelst.multimania {  
    import com.peterelst.multimania.*;  
    import flash.display.MovieClip;  
    public class Application extends MovieClip {  
        public function Application() {  
            var myModel:Model = new Model();  
            var myController:Controller =  
                new Controller(myModel);  
            var myView:View =  
                new View(myModel, myController);  
            addChild(myView);  
        }  
    }  
}
```

Specify `com.peterelst.multimania.Application` class as the Document class for the FLA

**Not convinced yet?**

# **FVNC**

**Open Source VNC client for Flash**

Author: Darron Schall

License: GPL

[www.osflash.org/fvnc](http://www.osflash.org/fvnc)

# **FC64**

## **Commodore 64 Emulator**

Author: Darron Schall - Claus Wahlers

License: GPL

[www.osflash.org/fc64](http://www.osflash.org/fc64)

# FZip

**AS3 class to extract zip files**

Author: Claus Wahlers - Max Herkender

License: OSI - zlib/libpng

[www.codeazur.com.br/lab/fzip](http://www.codeazur.com.br/lab/fzip)

# SMTP

## **AS3 class connecting to SMTP**

Author: Thibault Imbert

[www.bytearray.org/?p=27](http://www.bytearray.org/?p=27)

# **NNTP**

## **AS3 class connecting to NNTP**

Author: Luar

[www.luar.com.hk/blog/?p=647](http://www.luar.com.hk/blog/?p=647)

# Papervision3D

**High performance 3D engine**

Author: Carlos Ulloa, Ralph Hauwert, John Grden

License: MIT open source

[www.osflash.org/papervision3d](http://www.osflash.org/papervision3d)

**Want to learn more?**

## **Books**

Advanced ActionScript 3.0 with Design Patterns

Essential ActionScript 3.0 (coming soon)

Object-Oriented ActionScript 3.0 (coming soon)

## **Websites**

[weblogs.macromedia.com/mxna](http://weblogs.macromedia.com/mxna)

[www.peterelst.com](http://www.peterelst.com)

## **Training**

Lynda.com - ActionScript 3.0 in Flex Builder

TotalTraining.com - ActionScript 3 Essentials

**Thanks!**



**info@peterelst.com**