Flex in a Week, Flex 4.5

# Video 2.09: Programming ActionScript classes

In the first two Days of this training, you have written both MXML and ActionScript code.

You have placed the ActionScript code in your MXML file in a Script tag block.

In this video, you will learn how to create ActionScript classes without any MXML code.

You will also learn how to import and instantiate these classes in your MXML application code.

Lastly, you will learn how to create an ActionScript method that manipulates class properties and that can be called from MXML.

At the end of Day 1 you learned some basic OOP terminology and concepts and how they apply to Flex application development.

You should now understand that MXML “tags” are actually instances of MXML classes.

You should also understand how to extend an MXML class to create your own custom Flex components.

In this case, you created a custom MXML class named EmployeeDisplay.mxml and instantiated it twice, in the main application file, to display two employees.

When I run the application, you can see both Athena and Saul are displayed.

Their details are passed into the custom component via these class properties that, if you look in the custom component file, were created with ActionScript.

Remember that the Flex framework contains two programming languages.

MXML is primarily used for visual display while ActionScript is used for business logic and data manipulation.

In this video, I will show you how to create ActionScript classes.

Specifically, I will show you how to create a non-visual data class that will contain the employee data to pass into the EmployeeDisplay custom component.

To be clear, I will end up with two classes:

The visual MXML custom component class that you created in the last exercise to display an image and employee name, and the non-visual ActionScript class that will be used to populate the data for the visual component instances.

To create an ActionScript class, you first create a text file with an .as file extension.

Note that a single ActionScript file may define a single class or multiple classes.

In any case, only one of the classes in the file can be a public class.
I have used the term package a few times already in this series without fully explaining what it means.

Essentially a package corresponds to the directory where the ActionScript class text file is stored.

Here in the solution files for this video and associated exercise, I have stored the EmployeeDisplay.mxml custom component file and the Employee.as class file in a directory called components.

The package, for the Employee class file starts at the src folder and then includes any subfolders in the chain before it reaches the file.

In this case, the package is just components.

However, if the EmployeeDisplay class was inside of a subdirectory of components named display, then the package would be components.display.

Each directory in a package is separated by a period..

Note that it is not just coincidence that the Flex projects are organized in this view named Package Explorer.

Since the Flex framework is all about working with class files, the Package Explorer view reminds you that these folders are actually packages of class files.

By common convention, class names are comprised of nouns and adjectives; no verbs.

They also begin with a capital letter and use camel case for the remaining words within the name.

When you create an ActionScript class, you must reference the class name in three places: the file name of the .as file, the class name in the file and the name of the constructor, which is a special function in the class.

This is the basic syntax for the class code that you place in the text file.

Note that the class name is declared using the class keyword and the constructor is declared using the function keyword.

Both the class and constructor are also preceded by the public access modifiers.

As I mentioned before, you can have more than one class defined within one file, but only one of the classes can have the public access modifier defined.

Based on the naming rules I just outlined, because the name of the class and constructor are Employee, you can infer that the text file is also named Employee.

The package statement also tells you that the Employee.as file is stored in the components subdirectory of your Flex project.

You did not define a constructor in your MXML custom component class.

You can only create them in an ActionScript class.

A constructor is a function which runs automatically when each instance of the class is created.

This means that it is a great place to write code that initializes the instance.

A constructor is not required in an ActionScript class but it is considered a good practice to create one, even if it's empty, since the Flex framework will generate one automatically for you.

Lastly, unlike other functions that you have worked with in this training series, the constructor should not define a return type, even if it's void.

Here is the starter project for this video and associated exercise.

Remember that you already have a custom component named EmployeeDisplay.mxml in the components directory.

It displays the employee's name and image.

Now you will create an ActionScript-based class.

You can write the ActionScript class code by hand, but Flash Builder does have a wizard to help you.

I am right-clicking on the components directory because this is where I want to store my new ActionScript file.

I am selecting New > ActionScript Class from the menu, which is outside the visible area in this recording.

In the New ActionScript Class dialog,note that the package is automatically filled out for me.

I am naming the class Employee and leaving the access modifier set to public.

You can leave the other settings alone and click Finish.

The new Employee.as file opens in the Editor view with this code.

I mentioned earlier that I want this ActionScript class to store all of the data about an employee.

I am going to create each of these pieces of data, which I just discussed, as properties in the class.

You learned how to create class properties in MXML components in the OOP video at the end of Day 1.

The rules are exactly the same.

You first declare the access modifier, and then use the var keyword in front of the variable name.

Lastly, you define the data type for the variable.

For simplification purposes, I am going to limit the employee data to first name, last name and the name of the image file, but the code could easily be expanded to include email address, phone number, snail mail address and more.

In the MXML class file, you created the properties in Script tag block.

In an ActionScript file, properties are commonly declared above the constructor but within the body of the class.

I am typing imageFile and pressing CTRL+1 to invoke the quick assist tool and selecting the Create instance variable 'imageFile' option. This creates a private variable named imageFile data typed to the Object class.

I want to make it a public variable so I am changing the access modifier to public and using the content assist tool to change the Object class to the String class.

I am creating two more public variables, also setting them to String data types but naming them firstName and lastName.

Remember from the oop video, I assigned property values to instances of the MXML custom component, by assigning them inline in the MXML tag.

Here are the imageFile and fullName properties being passed to the two instances of the EmployeeDisplay custom component.

ActionScript class properties can be assigned differently based on arguments defined in the constructor.

To allow a class to accept property values in the constructor, you assign arguments between the parentheses of the constructor.

You give the argument a reference name for use within the constructor and define the data type for that argument.

Lastly, within the body of the constructor, you assign the argument to the class property.

In this code, you can see that I've added three arguments for the constructor that correspond to, but are not the same as, the three public variables that I created earlier.

Inside the body of the constructor, I have associated the argument values, fileName, fName and lName, to the class properties, imageFile, firstName and lastName.

Notice that the argument reference names are different from the class property names.

This is a common dispute among developers.

Some prefer that the names are kept distinct in order to avoid confusion.

Other developers prefer to match the constructor argument reference names to the class property.

This, however, can lead to confusion in the code because you cannot distinguish between the arguments in the constructor and the class properties.

If you choose to name the argument and the class properties the same, you must use the "this" operator to explicitly reference the class properties.

I am adding the "this" operator to each of my class property references within the constructor, even though my current code avoids any confusion.

The "this" operator is a self-reference to the specific instance of the class that is being created.

Therefore, it refers to the class properties not the argument reference names in the constructor.

In other words, these variables on the left of the assignment reference the public variable definitions defined in the class while the variables on the right of the assignment refer to the constructor arguments.

Now that I've created the ActionScript class that represents all Employee data, I will use it to create instances for each employee.

To use the class, I must first import it into my application using the import keyword and referencing the package and class name.

I will also name the instance and data type it to the class.

Lastly, I will use the new keyword to call the constructor and pass in the arguments.

I'm saving my class file and then returning to the main application file.

I am creating a Script tag block below the Script comment.

I am typing firstEmployee and using the quick assist tool – CTRL+1 – to create a variable. The variable generated is a private variable named firstEmployee, data typed to the Object class.

I am leaving this variable private because this data will only be used in this application, but I will use the content assist tool – CTRL+Space – to change the data type to the Employee class that I created here in the ActionScript file.

Note that, since I am using the provided content assist options, Flash Builder will automatically import the class, referencing the correct package.

If you do not use content assist, you must remember to type the import statement manually.

To instantiate this object, I am using the new keyword and then referencing the Employee constructor.

Between the parentheses, I am passing the data for this employee in the same order that it is defined in the constructor's arguments list.

"aparker.jpg" is the first value.

"Athena" is the second value, and

"Parker" is the final value.

I am now creating a second set of employee data.

This will also be a private variable named secondEmployee, data typed to the Employee class.

The constructor arguments are "stucker.jpg,' "Saul" and "Tucker."

These two instances of the Employee class contain all of the data that I need to pass to the EmployeeDisplay custom component instances.

I will replace the data in the visual components with the data from my ActionScript class instances.

I am populating the imageFile property for the first component with a binding to the firstEmployee.imageFile value.

When I save the file, which will cause Flash Builder to re-compile, you can see a warning message that state that the data binding cannot detect assignments to the firstEmployee and imageFile variables.

Remember that variables created in ActionScript and used in bindings to MXML components must use the Bindable metadata tag.

I am using content assist to add this tag above both instances of the employee variables.

However, when I save the files, the warning message for the binding still exists for the imageFile property.

In this case, the message is referring to the class definition of the imageFile property itself.

I am returning to the Employee.as file and placing the Bindable metadata keyword above the imageFile property definition.

When I save the file, you can see that the warning message disappears in the main application.

A class property must be bindable in order to be used in a binding to a UI control.

If all of your properties in a class will be used in binding statements, you can make the entire class bindable by moving the Bindable metadata keyword above the class definition.

Be wary, though, of overusing bindings in this way.

Only define the entire class as bindable if you intend to use all the properties in binding statements.

They are extremely useful, but creating them does create overhead because the application then must watch all of these properties for changes.

I do not need a class binding for this application, so I am undoing that change.

Back in the main application, I am adding the secondEmployee.imageFile property as the value to pass into the second EmployeeDisplay instance.

When I save the application and run the file, you can see that the data is properly passed to the two component instances because their images display properly.

Note, however, that I will still need to pass in the first and last name values because they are currently hard coded.

Another way to handle the data passed into the custom component is to treat is as one object.

I just defined all of the employee data for each employee as objects named firstEmployee and secondEmployee.

I can pass this entire complex object to the EmployeeDisplay custom component rather than passing the values one at a time.

In order to do this, I must first modify the EmployeeDisplay custom component to accept one complex object rather than these two separate arguments.

I'm going to Control + click on the component name to open the file.

Then, I'm going to comment out these two bindable public variables using the Flash Builder keyboard shortcut Control + Shift + C, or command + shift + c on the Mac.

Note that if you use this shortcut in a Script block, then you will get script comments.

If you use it around an MXML tag, then you will get HTML-like comments.

By the way, you can find keyboard shortcuts in the  Help > Key Assist menu.

Next I will import the components.Employee class for use in this MXML file.

The one Bindable public property of this custom component class will now be called employeeData and will be data typed to the Employee class.

I am updating the binding in the Image control's source property to use the new employeeData object.

Specifically I am referencing, still, the imageFile property stored in the object.

Next I am removing the binding from the Label control that displays the employee name.

I will discuss this further in a moment.

I am saving the file and returning to the main application file, where you can see the compiler errors that alert you to the two missing class properties.

I am removing the imageFile and fullName properties from both custom component instances.

You can see that the content asssist for the component includes the new employeeData component property.

Rather than passing two separate values for the employee data, I am passing the entire Employee data instance at once.

For the first instance of the EmployeeDisplay custom component, I am binding the employeeData property to the firstEmployee data instance.

The second EmployeeDisplay instance is being bound to the secondEmployee data instance.

Now when I save the file and run the application, you can see both employee images without their employee names.

In the final section of this video, I will populate the employee names in the EmployeeDisplay custom component by calling a method that I will create in the Employee ActionScript class file.

If you recall from our oop discussion, methods are actions or behaviors accessible to instances of a class.

All properties of an instance are available to the methods so methods are commonly used to manipulate and control properties without changing their underlying values.

Like properties, you should assign access modifiers to the methods to define how other code can access the methods.

You may often hear the terms function and method used interchangeable.

While this is a common practice there is a distinction between the two.

A function is a generic behavior.

For instance, the throw() function can be applied to almost anything.

You can throw a TV, a lamp, a chair, or even a car if you're strong enough.

A method, however, is a behavior specifically tied to a class.

For instance, the throw() method for the Football class may define that you place your fingers along the football's laces before you throw it.

Once you define the method, you will add the logic to determine its actions between the curly braces of the method definition.

Back in the Employee.as class file, I am adding a function below the constructor.

This is a public function that I am naming createFullName().

When this function is called, it will return the full name of the employee.

Earlier I mentioned that the constructor function does not have a return type.

Class methods always have a return type.

If the method does not return a value, you would use the void keyword after a colon in the function declaration.

Since this function is returning the name of the employee as a String, I am typing String for the data type.

Inside the curly braces for the method, I am typing the return keyword which specifies the string, in this case, that is returned when the method is called.

I want to return the firstName and lastName class properties separated by a literal space so I am typing firstName plus the literal space between double quotes plus the lastName property.

You may have noticed that I am not using the “this” keyword to reference the firstName and lastName properties.

Inside of the constructor you must clarify whether you are referencing the class property or the constructor argument.

The “this” operator helps you to make the distinction.

Outside of the constructor, the constructor arguments don’t exist, so when you refer to firstName or lastName, the application knows that you are referencing the class properties.

I am saving the file and then returning to the EmployeeDisplay custom component.

Remember that employeeData is an instance of the Employee class in which we just added the createFullName() method.

Since this is a public method, we can access it from this EmployeeDisplay custom component instance.

In the text property of the Label control, I am adding binding curly braces and calling the employeeData.createFullName() method.

I am saving the file, and when I run the application, you can see that both employees are displayed with the proper image and names.

For your next step, work through the exercise titled “Creating an ActionScript class and instances”.