Flex in a Week, Flex 4.5

Video 2.10: Implementing value objects and a typed data model

In the last exercise you created an ActionScript class for employee data.

In this video, you will learn how that Employee class can be implemented as a value object to type data.

You will also learn how to turn a collection of generic objects into a typed data model.

A value object is a typed object with properties.

Consider the Employee class that you created in the last exercise.

It could be considered a value object class because its instances are objects that are typed as employees and contain properties like name, email address and phone number.

Value objects are generally free of any implementation detail or business logic and simply contain value data related to an object.

Some of you may be familiar with the Java transfer object or data transfer object.

Those are server-side versions of a value object.

In a Flex application, you might use a DTO and a value object together.

For instance, the server-side transfer object might be an employee object created in Java that is passed to the Flex application and then implemented as a value object in ActionScript for populating a form.

In Flex, you will create value objects from an ActionScript class.

You will often see the classes created in a directory named vo or valueObjects.

Each piece of value data is created as a class property, where the individual property or the entire object can be bindable.

By making the value data into class properties, you will have access to them in the Flash Builder code hinting interface.

In this video, I will work with the Employee Portal: Vehicle Request Form.

This is the solution file for this video and the associated exercise.

Remember that this file implements the MVC pattern, where the data that is retrieved from this HTTPService call populates the employees model, the UI components are separated into the VehicleRequestForm custom component view, and the main application is the controller that handles the communication between the model and view.

In the valueObjects folder of the Package Explorer, I am opening the Employee.as class file.

There are only two differences between this file and the one that you created in the last exercise.

First, this one is in the valueObjects package instead of the components package.

Second, there are a lot more properties in this file, which happen to match the data coming from the XML file.

Note that the package doesn't really matter.

We could have left the class in the components package.

I am just following the convention that we have chosen for this training series.

In the main application file, you can see that the HTTPService call has a result event defined.

I am holding down the Control key and then clicking on the handler to locate it in the code.

I am adding a breakpoint to the closing curly brace of the handler so that I can look at the returned data.

I am clicking on the Debug button.

The application displays briefly in the browser and then I am prompted, in Flash Builder, to switch to the Debugging perspective.

I am double-clicking on the Variables tab to maximize the view and then drilling down into the event object to look at the result property.

You can see that the repeating employee XML node is automatically converted by the Flex framework into an ArrayCollection object with each employee filling a numbered index.

Each index is a generic ActionScript object with multiple name-value pairs of data.

Handling the data generically like this is fine, but data has meaning so it is better to create a model that contains explicitly typed data.

Specifically, I am going to turn this ArrayCollection instance of generic data into an ArrayCollection instance of employee objects.

Remember that you create instances of a class in ActionScript by first importing the class and then instantiating it using the new keyword.

Since I did not create any arguments for my class constructor, I cannot pass the values for the instance in the constructor.

However, I can simply reference the instance name and then the class property using dot syntax to populate the values.

Once I have created my Employee instances, I can create a model of data by simply adding them, one at a time, to my ArrayCollection instance using the addItem() method.

You saw earlier that I currently have an ArrayCollection of generic objects being returned from the HTTPService request.

To turn this array of untyped data into an array of typed data, I will simply loop over the untyped data and use the information to populate instances of my value object class.

As I create each instance, I will simply add it to my model using the addItem() method of the ArrayCollection class.

In the main application file, note that the returned data in the result object is populating an ArrayCollection object named employees.

This property now holds the ArrayCollection of generic data that was returned from the server.

I want to manipulate the returned data before I populate this employees class property, so I am going to add a var keyword before this variable to make it a local variable and then I am changing the name to employeeData.

I must data type this new variable to the ArrayCollection class since this is the type of data being returned from the server.

After this assignment, I am invoking the content assist tool by pressing CTRL+Space and typing fore and pressing Enter to create a for each loop from a code template provided in Flash Builder.

I am modifying the template by changing I to create an iterant named emp and changing int – using code hinting with CTRL+space - so that it is data typed to the Object class.

Notice that the in operand automatically specifies the employeeData ArrayCollection variable as the data to loop over.

This code will loop over the employeeData object and reference each item of data in the array as emp on every iteration of the loop.

As I loop over each item in the employeeData ArrayCollection, I want to convert the data from a generic object into a typed data object.

Explicitly, I want to create instances of the Employee value object.

Above the loop, I am declaring a local variable named employee, data typed to the Employee class.

Note that Flash Builder automatically imports the class for me.

Within the for each loop, I am instantiating the employee variable using the new keyword.

I have called its constructor.

Next, I am populating the firstName property for the instance with the emp.firstName value.

Remember that employee is the typed Employee object that I am creating, while emp is the untyped data from the server.

To save time, I am pasting all of the other properties.

Currently, this code just creates a new employee object on every instance of the loop.

Now, I need to take that employee object and populate the employees ArrayCollection class property with it.

As the last line in the for loop, I am typing employees dot addItem() and then I'm adding the employee object.

So, let's review this code.

When the result data is returned from the server, I am using it to populate a local variable named employeeData, which I then loop over.

Each item of data from the server will be referenced as emp in the loop.

As the loop iterates, a new instance of the Employee class is instantiated and populated from the associated server data.

The typed value object, employee, is then added to the employees class property, which is bound to the VehicleRequestForm custom component's employees property.

That, in turn, is bound to the DropDownList control in the view.

When I run the application, you can see that the data populates the DropDownList control as before.

However, if I debug the application, you can see that the employees data now contains an ArrayCollection of Employee value objects.

For your next step, work through the exercise titled "Creating an ArrayCollection of value objects".