**Prepare for Unit Test**

**Module Description**

Following the updates to your testing work products, you will begin to write test code based on the detailed designs.

**Participant Presentation**

Click the link to launch the [Module 34 Presentation](https://accenture.desire2learn.com/content/enforced/9406-Pending/W3_Participant_Materials/Module_34/ADF_2_0_Java_M34_TimeWarp_PD.pptx?_&d2lSessionVal=YnoL4daZGC8ne2RyGf0cWMrk8).

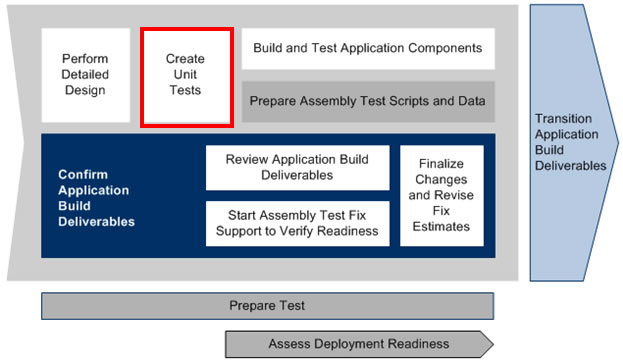
**Scenario**

You will use the created Unit Test Documentation to create automated test scripts that will execute and check the correctness of the application code. You will also update the stub code to support the test script execution (e.g., provide the appropriate outputs or exceptions for the test). Use the Test Cycle Control Sheet document to determine which scripts are used to test which conditions and what the name of the test scripts should be.

As mentioned in the orientation, team status is assessed periodically through a Team Status Meeting. The Team Lead will host this meeting and gather the team status from you and the rest of your team. At the end of the meeting, the Team Lead will communicate your team status to the management. This helps them evaluate how well things are progressing and whether intervention is needed.

**ADM Activity Context Diagram**

* Application > 4100 Build Application > 4183 Create Unit Tests



**Roles**

|  |  |  |  |
| --- | --- | --- | --- |
| **Role** | **ADM Task** | **ADM Responsibility** | **Description** |
| Developer | 4183 - Create Unit Tests | Primary Performer | Creates automated tests to be used on the implemented Java code. |
|  | Team Status Meeting | Primary Performer | Provides status on assigned tasks and activities including: what is completed, in-progress, delayed.  Communicates any issues that impact progress or productivity. |
| Team Lead | Team Status Meeting | Primary Performer | Leads the Status meeting and compiles feedback. Addresses and corrects any issues where possible. Escalates issues when appropriate. |
| Project Manager | Team Status Meeting | Management Oversight | Provides guidance on the Team Status Report. |

**Participant Instructions**

1. Confirm that the JUnit library is installed in your project. If not, review the [Tools Primer](https://accenture.desire2learn.com/content/enforced/9406-Pending/Tools%20Primer/ADF_2_0_Tools_Primer.docx?_&d2lSessionVal=YnoL4daZGC8ne2RyGf0cWMrk8) document for the steps to install the library.

2. Open your latest versions of the following work products:

TE583 Test Scenarios

TE584 Test Conditions and Expected Results

TE585 Test Cycle Control Sheet

3. Start the Eclipse development environment.

4. Open the FestivalPortal Project.

5. Write JUnit test scripts to test the test conditions that are part of the documented test scenarios as defined by the Test Cycle Control Sheet.

6. Since the application code to be implemented later must be isolated (i.e., it can't call other classes that are being built, but must be able to be executed), write stub code for the class methods that will be called.

**Hints and Tips**

* Use the AP333 Component/Class Design documents for the other classes to be implemented to understand what type of output the classes provide. Write your stub code based on the outputs that are being expected by your code.
* Remember that the JUnit test scripts are Java programs! They need to be written so that they follow the project development standards documented in the TA243 Application Development Standards document.
* Make sure to confirm that you have sufficient code coverage based on the detailed design in each (based on the pseudo code that is provided - inputs, output, and possible exceptions).
* There should be tests that cover negative as well as positive tests.
* Try to make your stubs as simple as possible by:
  + Returning a value (e.g., if an integer is returned, the stub code in a method can be one statement, such as "return (-999);")
  + Using control-flow statements like if-else, while loops do loops and switch-case statements to allow your tests to pass parameters that generate the specific results desired (even throwing exceptions). This is a more flexible approach.