LAB 6B – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL CICS/DB2 application (60 minutes)

Updated December 22, 2020 by Regi – Created by Regi/Wilbert

Acknowledgments:
We would like to thank the following for their assistance:
Suman Gopinath & Nathan Cassata.

This lab will take you through the steps of using the automated unit testing (zUnit) capabilities of IBM Developer for z (IDz) to create a unit test case for a COBOL CICS/DB2 program. This enables the testing of just a single program within a CICS transaction without the need to run the whole transaction. This is done by stubbing out CICS calls, enabling the program to be tested without a CICS environment and without the need to deploy to CICS after a code change. This enables a developer to test early without impacting other developers that share the same CICS environment.

In this lab you will record interaction with a COBOL CICS/DB2 program (program under test) and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the CICS COBOL program, and rerun the unit test.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

1. **Get familiar with the application using the 3270 terminal**
   → You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be recording.
2. **Import a z/OS project**
   → You will import a z/OS project with the required resources added to the project.
3. **Record interaction with the application.**
   → You will record an interaction with the COBOL CICS/DB2 program.
4. **Generate, build and run the unit test**
   → You will compile and link-edit the generated unit test program, followed by running the unit test.
5. **Modify the program and rerun the unit test**
   → You will modify the program under test, rerun the unit test, and observe the failure of the test case.
6. **Run the unit test from a batch JCL .**
   → You will run the unit test from a Batch JCL and observe a similar test case result.

Section 1. Get familiar with the application using the 3270 terminal

You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be interacting with.

___ 1.1 Connect to z/OS and emulate a CICS 3270 terminal

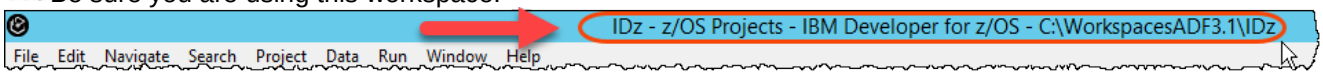
You will use IDz to emulate a 3270 terminal to run the CICS transaction.

1.1.1 ⏭ Start *IBM Developer for z Systems version 14* if it is not already started

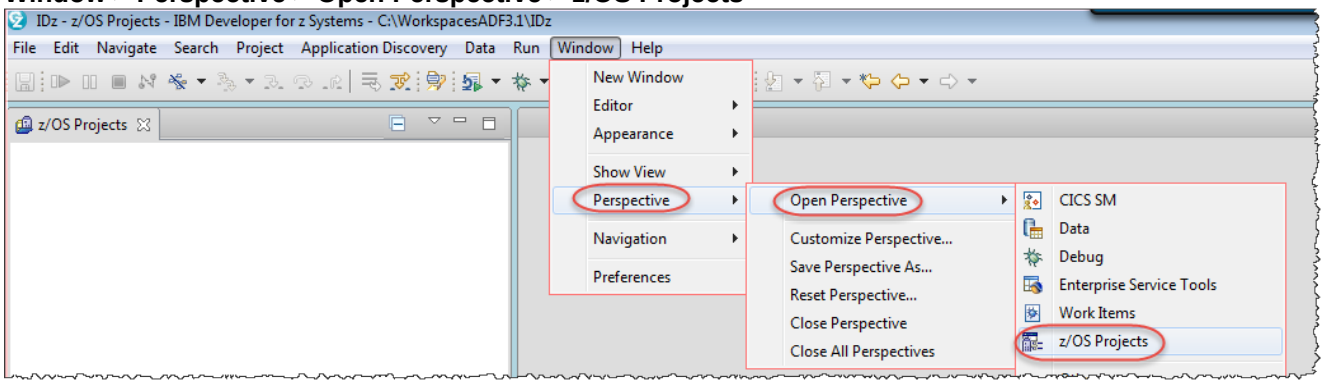
⏭ Using the desktop double click on **IDz 14.2.2** icon.

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.

⏭ Be sure you are using this workspace:

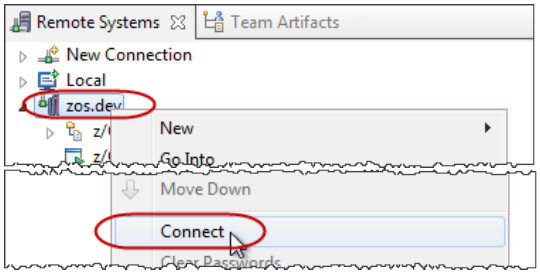


1.1.2 ⏭ Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

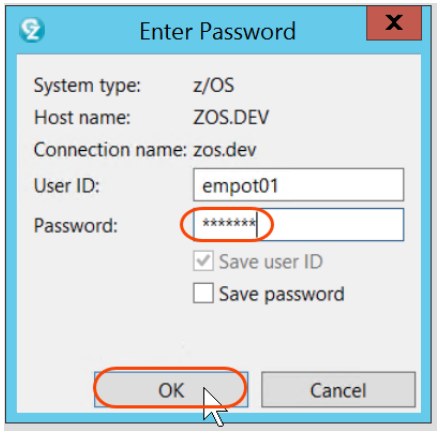


1.1.3 On this lab you will use userid **empot01** .
If you are connected as empot01, jump to step 1.1.5 Otherwise**.**

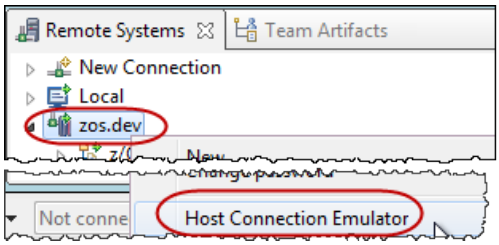
⏭ Right click on **zos.dev** and select **Connect**



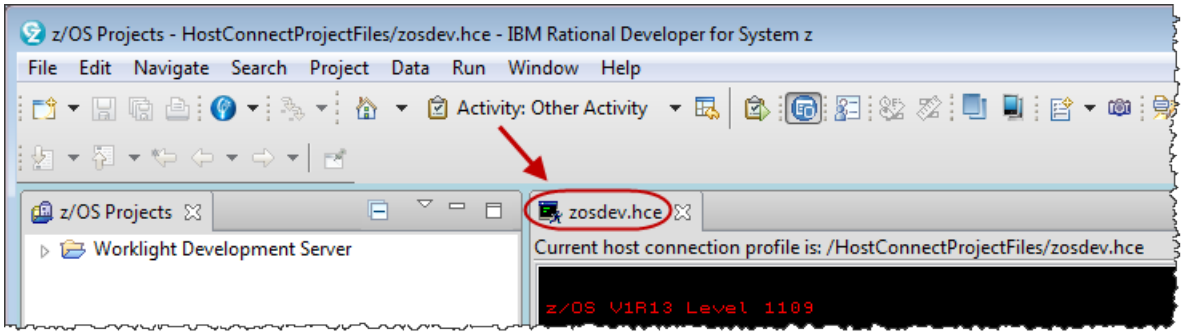
1.1.4 Type **empot01**as userid and **empot01** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.



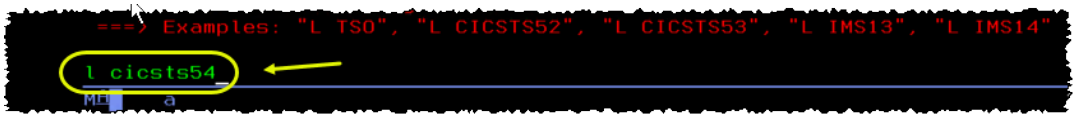
1.1.5 Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator.**



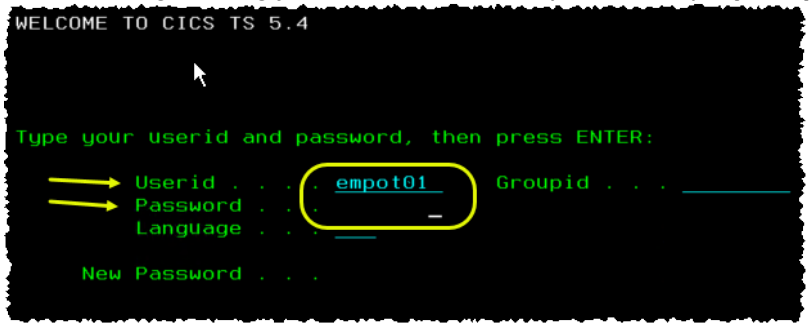
1.1.6 Since you will need more space, **double-click** on the ***zos.dev.hce*** title



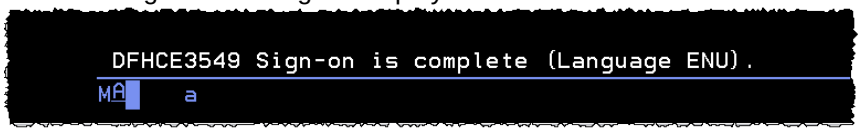
1.1.7 Type **l cicsts54**. (where "l" is the lower case of letter "L") and press **Enter key**.



1.1.8 Logon using your z/OS user id and password (**empot01)** and press **Enter**.



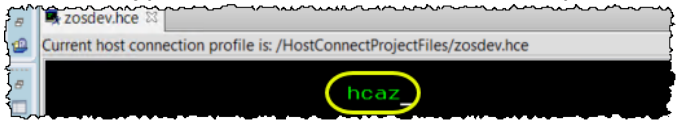
1.1.9 The sign-on message is displayed



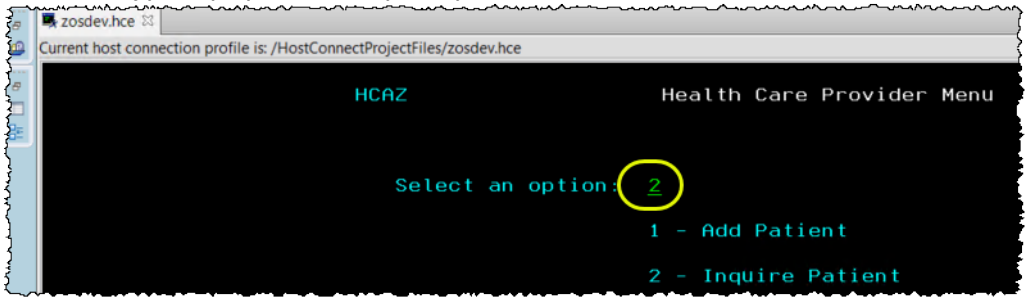
___ 1.2 Run CICS transaction HCAZ

You should now be in the z/OS CICS region named *CICSTS54*. This is the CICS instance where you will make the recording of your interaction.

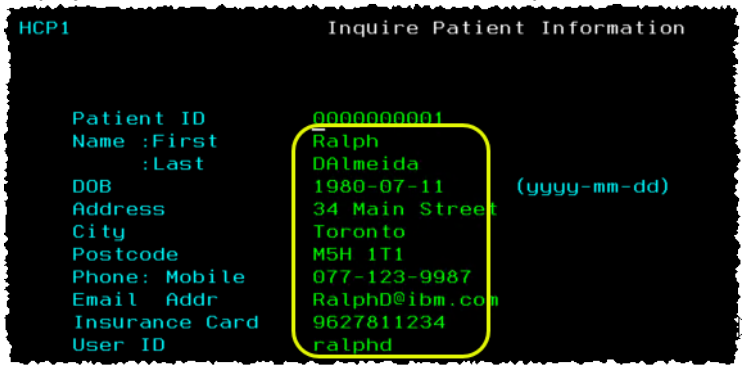
1.2.1 Type the CICS transaction **hcaz** and press the **Enter** key.



1.2.2 Type **2** (Inquire Patient) and press **Enter** .

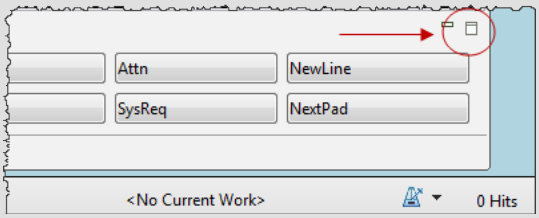


1.2.3 Type **1** for Patient ID and press Enter. The program will read the patient from a DB2 table and display the customer details. Remember the patient first name: **Ralph**

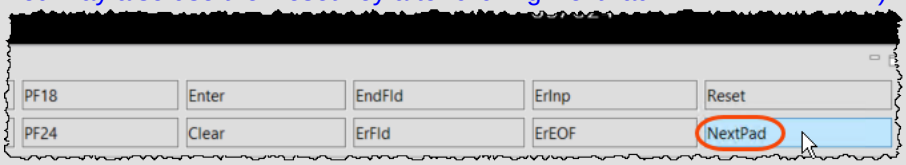


IF you need to use functions like Clear or Reset

*If you need to use the **clear** function use the key **Esc**.*
.Also you may look in the right lower corner, select this icon . This will display possible keys, including the clear button.



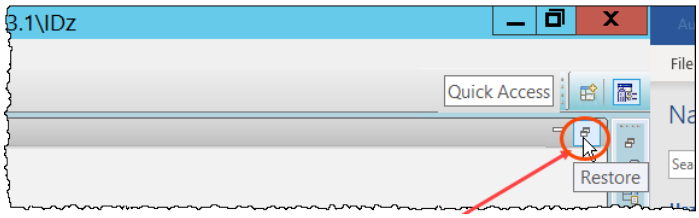
You may also use the Reset key after clicking NextPad)



1.2.4 Press **F3** to end the application.

1.2.5 **Close** the terminal emulation clicking on → . Or pressing **CTRL + Shift + F4**.

1.2.6 You may need to restore the **z/OS Projects** perspective by clicking on the icon on top right:



What have you done so far?
You emulated a 3270 terminal using IBM Developer for System z.
You also executed the CICS transaction **HCAZ** and verified a simple interaction with the Hospital application. The objective here was to show the code that you will update.

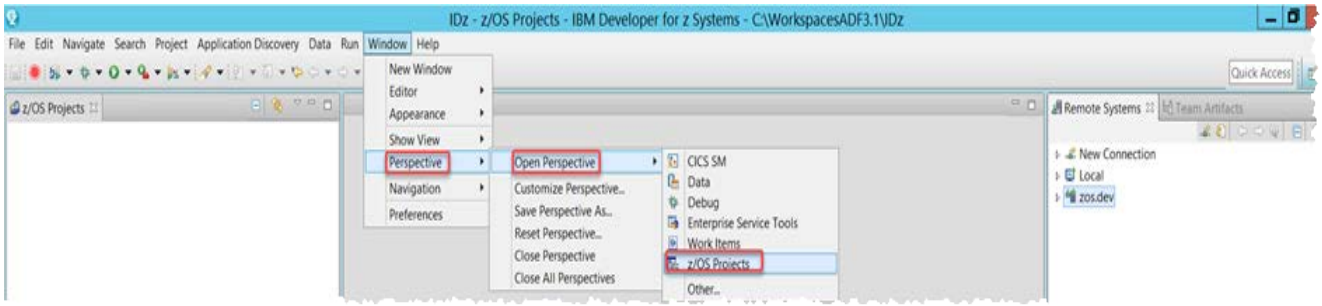
Section 2 – Import a z/OS project

You will import a z/OS Project that contains the resources needed to generate a unit test program for a COBOL CICS application.

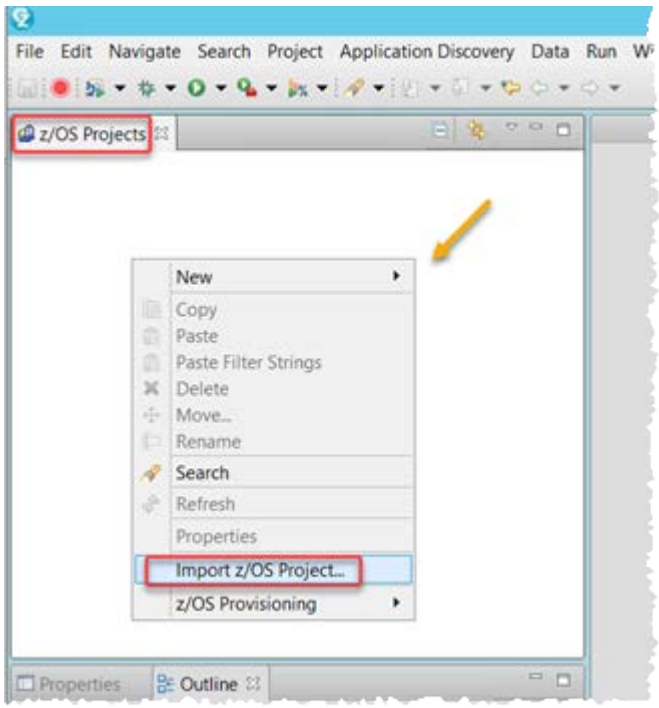
___ 2.1 Importing the LAB6B z/OS Project

The Hospital Application used in this lab has its source code in a *PDS* that's been added to a z/OS Project. You must be connected to the zos.dev system (see step 1.1).

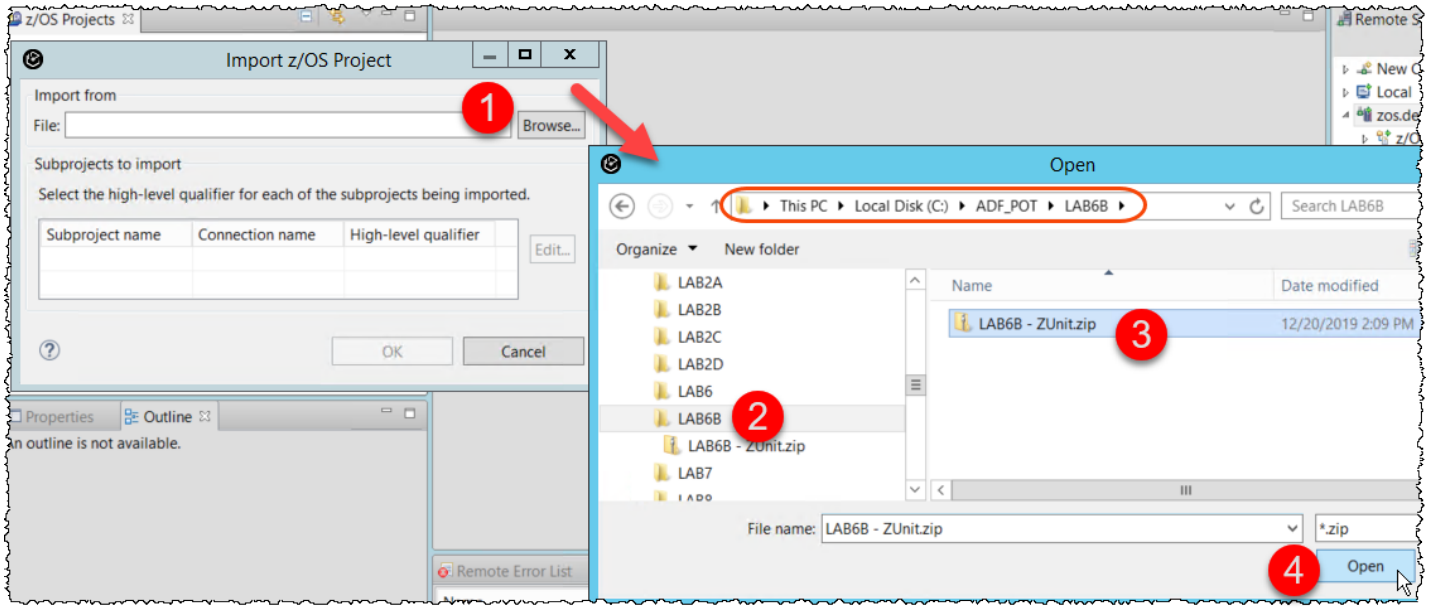
2.1.1 ⏭ If not already in the z/OS Projects perspective, open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



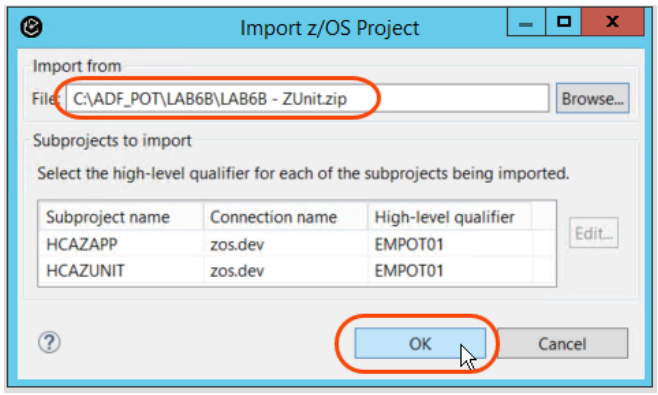
2.1.2 ⏭ Using the **z/OS Projects** view on top left, position the cursor anywhere in the view. Right mouse click and select the **Import z/OS Project** action.



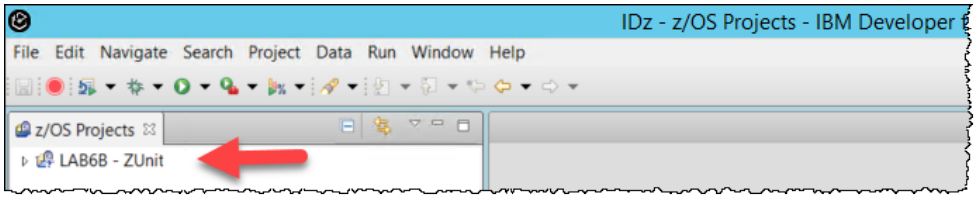
2.1.3 In the **Import z/OS Project** dialog, click the **Browse** button, navigate to **C:\ADF_POT\LAB6B** and select **LAB6B - ZUnit.zip**. Click **Open**.



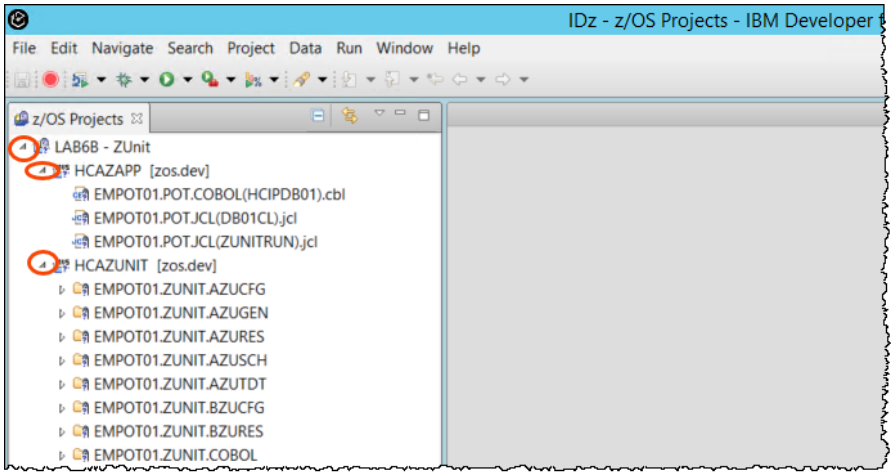
2.1.4 In the dialog that opens, click **OK**.



You should see something like this in your z/OS Projects view:



2.1.5 **Expand the nodes** by left clicking on the icon as shown below:.



Section 3 – Record data interaction using the CICS application.

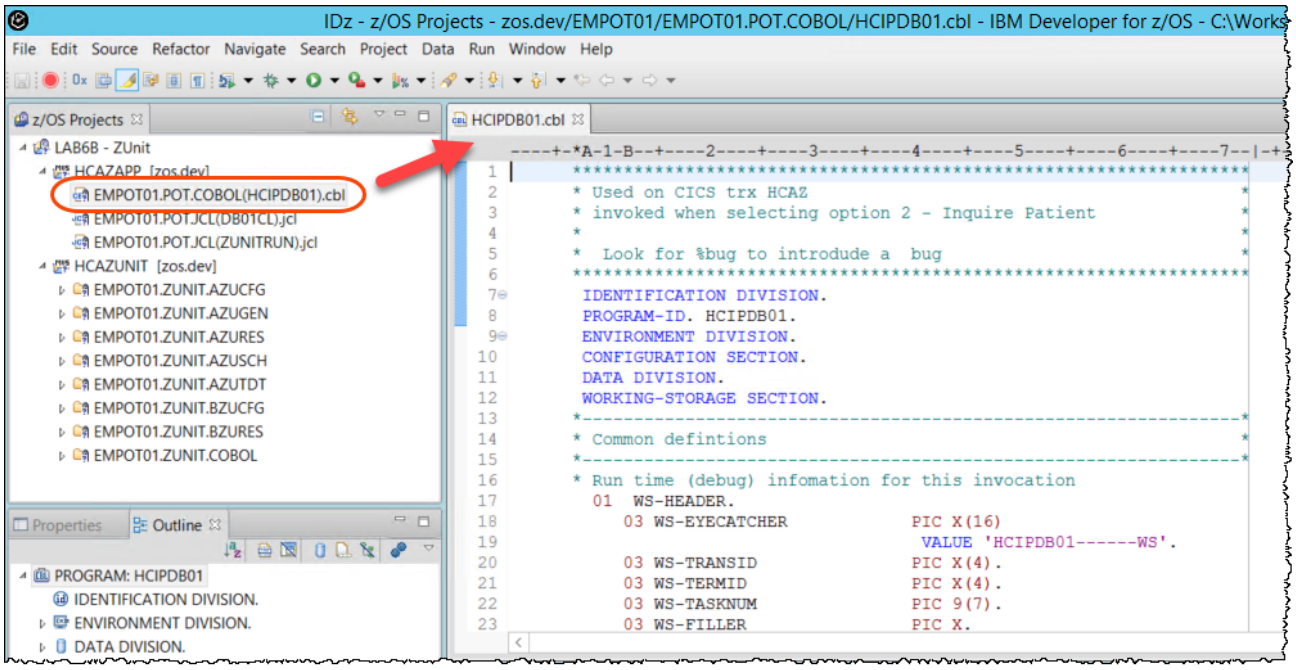
Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

___ 3.1 Understanding the COBOL program that reads from DB2 table

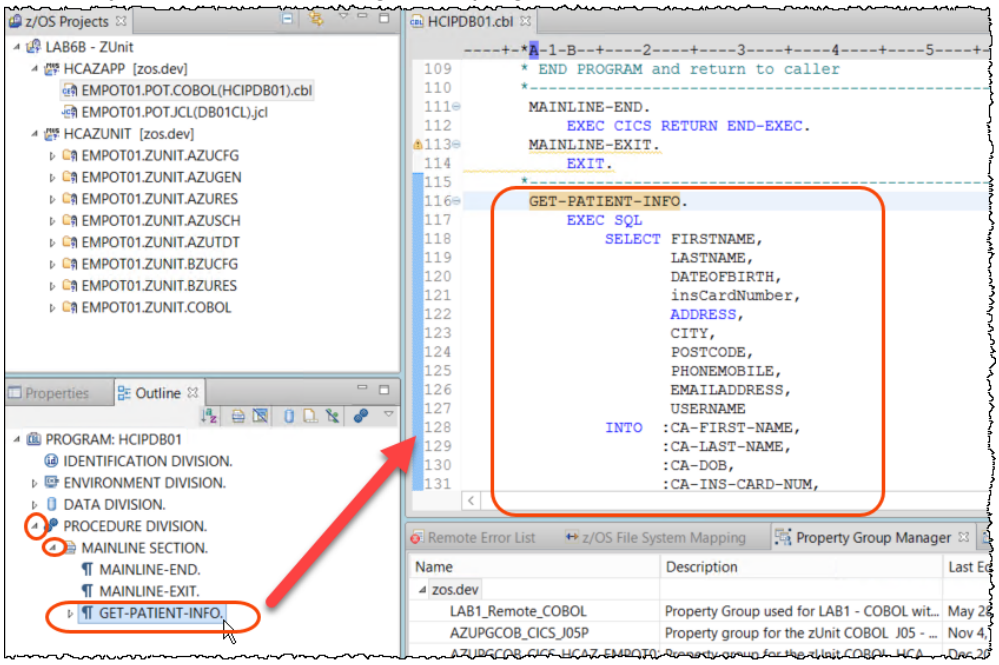
The COBOL code that reads the patient from the DB2 tables is the program ***HCIPDB01***

3.1.1 Using *z/OS Projects* view double click **EMPOT01.POT.COBOL(HCIPDB01).cbl** under **HCAZAPP**.

This is the program that you will update later on.

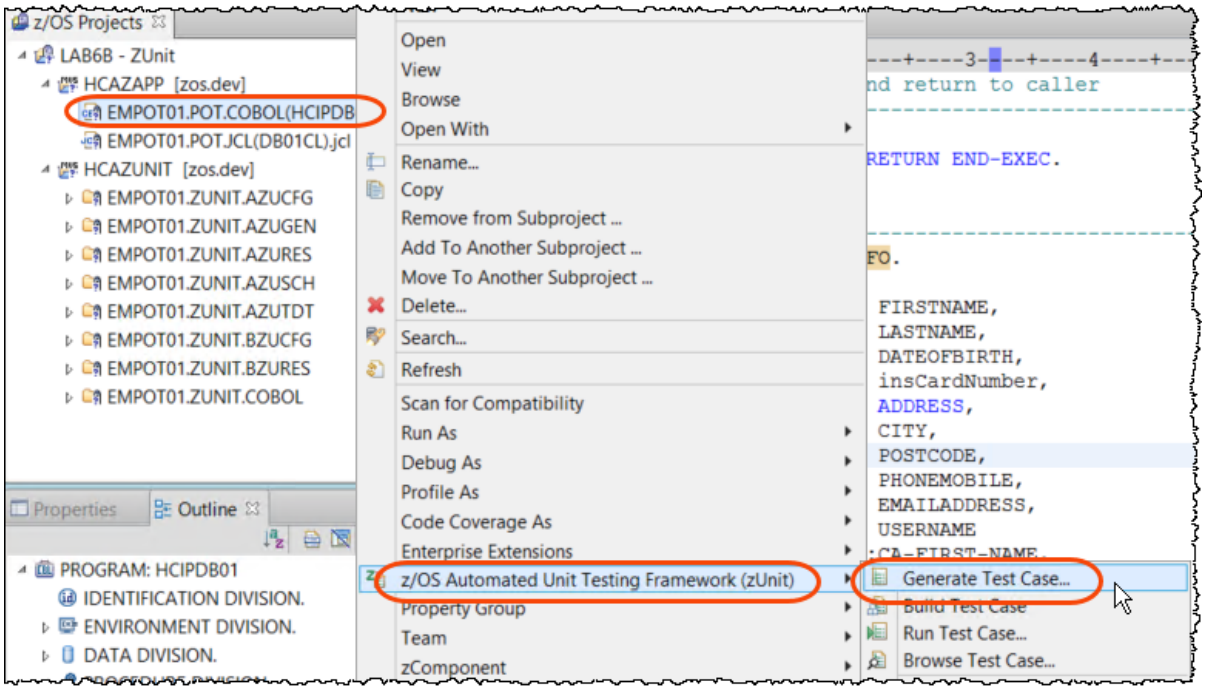


3.1.2 Using the Outline view on left expand **PROCEDURE DIVISION** and click on **GET-PATIENT-INFO.** This is where the patient data is read from the DB2 table.
Later on you will introduce a bug in this program..

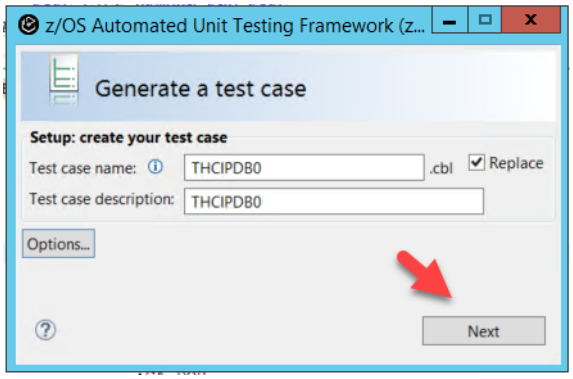


___ 3.2 Recording the COBOL program that sends the message

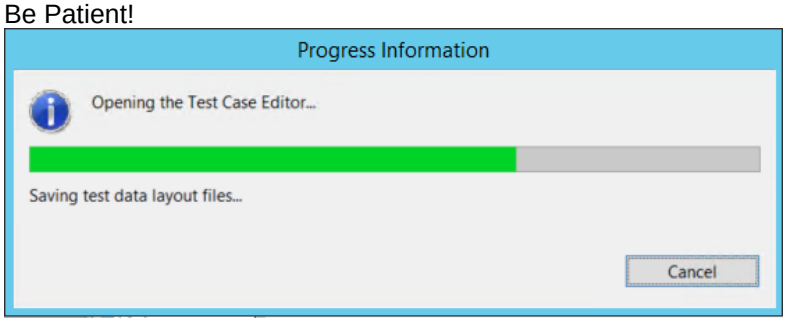
3.2.1 To start the recording, right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)->Generate Test Case..**



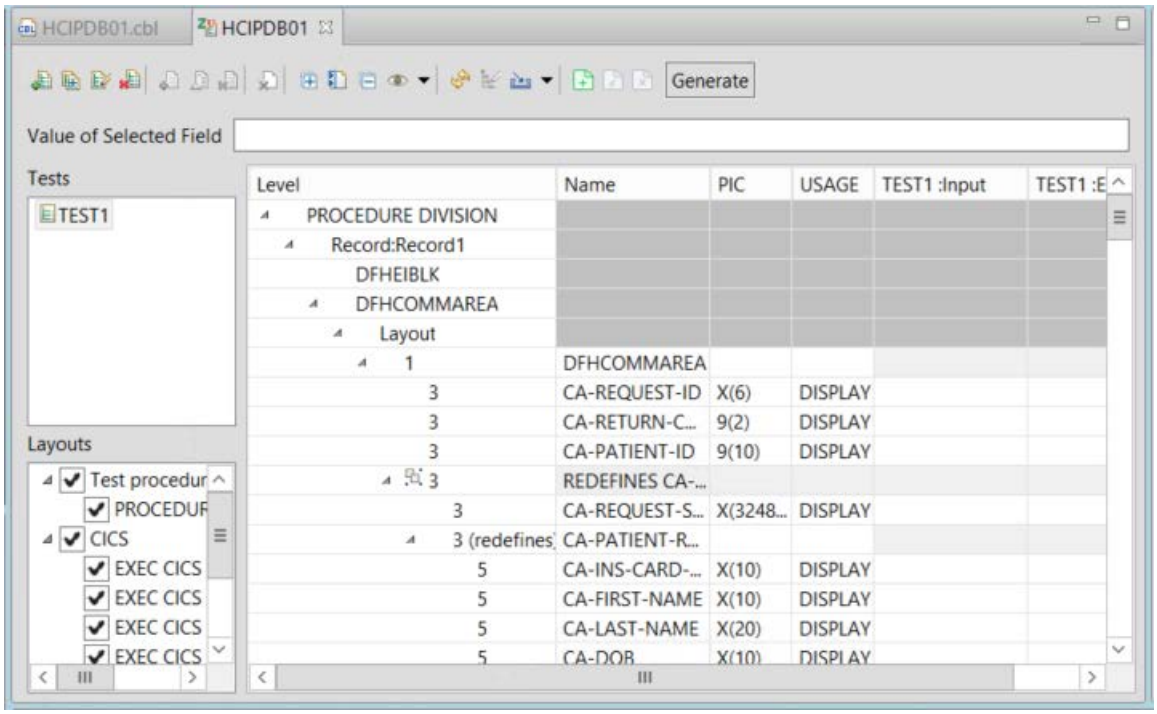
3.2.2 This opens a dialog where you can name your test case.
Accept the auto-generated name and click **Next**.



3.2.3 This operation may take a while since members are being created on z/OS PDS **EMPOT01.ZUNIT.***
Be Patient!

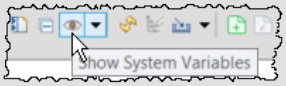


3.2.4 This will open the **Test Case Editor**, as shown below. TEST1 may or may not be on your screen.
If TEST1 is there you will delete on next step.



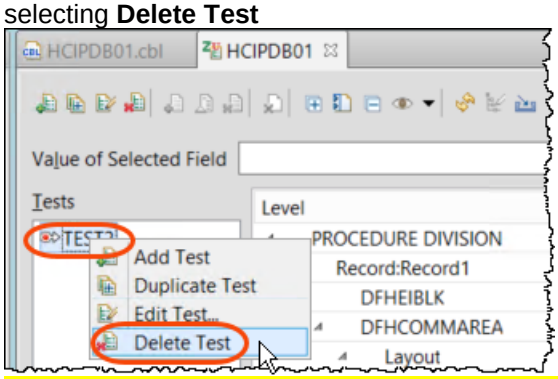
Understanding the test case editor

- The bottom left box summarizes all the input output variable structures – In this example, the program has a COMMAREA exchanged via the PROCEDURE DIVISION, five EXEC CICS statements and one EXEC SQL statement.
- The COMMAREA and the CICS statements are also accompanied by the CICS DFHEIBLK variables. To see those values you must select the icon



The variables that are returned by the program are the output from the program logic that a developer should be checking

3.2.5 Since we will be recording the test data, delete the **TESTx** entry (if it exists) by right clicking and selecting **Delete Test**

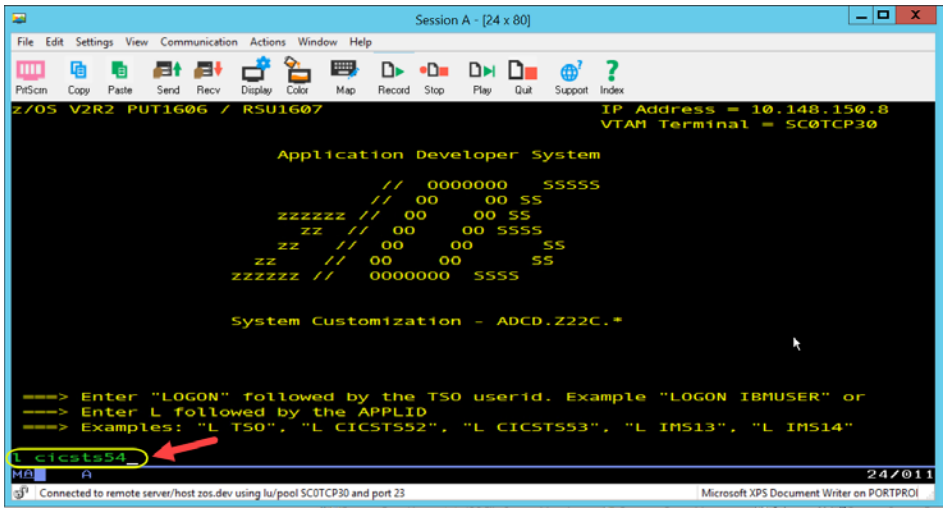


Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

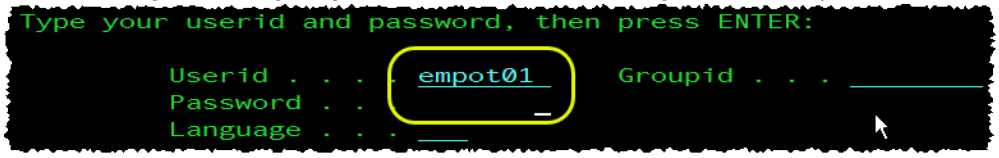
3.2.6 Bring up a 3270 terminal emulator clicking on the **host emulator icon** on the Windows task bar.

This opens the host emulator.

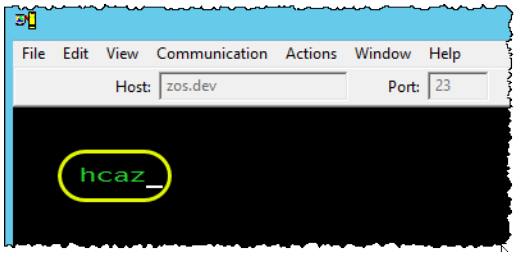
3.2.7 Type **l cicsts54**. (where l is L lower case) and press **Enter key** or the **right Ctrl** key.



3.2.8 Sign on using **empot01** as the userid and **empot01** as the password.

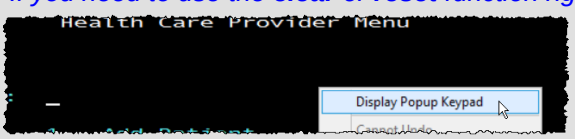


3.2.9 Once you see the sign-on is complete message, enter **hcaz** and press the right **Ctrl** key.

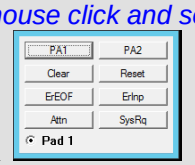


IF you need to use functions like Clear or Reset

*If you need to use the **clear** or **reset** function right mouse click and select as below*



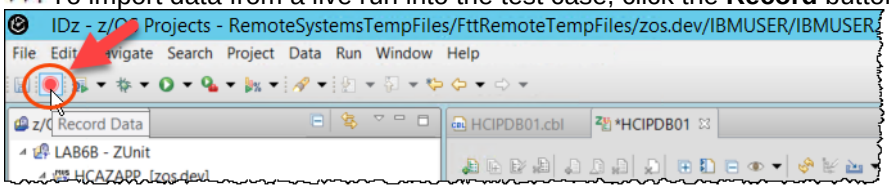
and



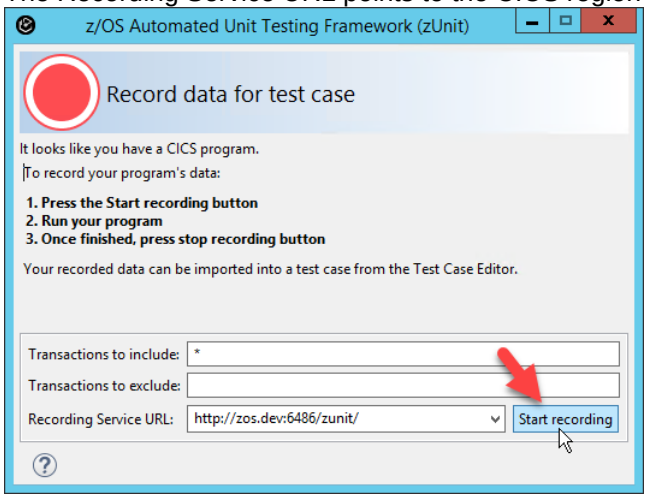
You are now ready to start recording and import data into the test case editor.

3.2.10 Minimize the terminal emulator and go back to IDz dialog.

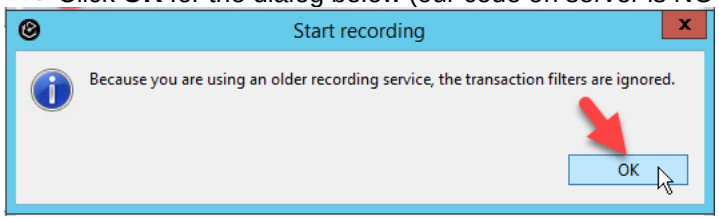
To import data from a live run into the test case, click the **Record** button on the IDz toolbar.



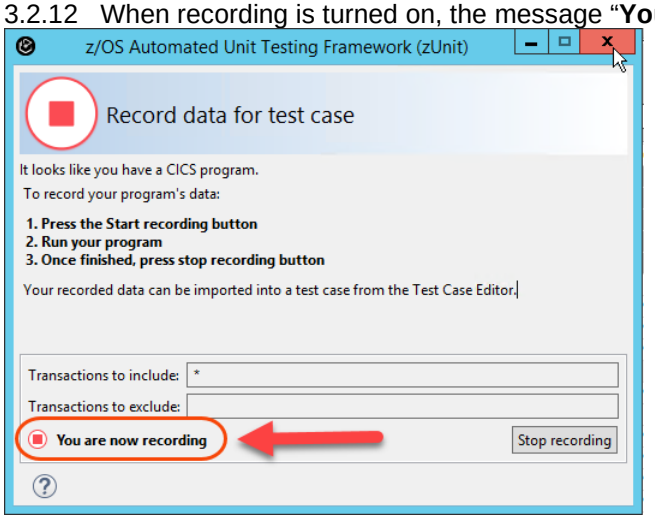
3.2.11 In the dialog that comes up, click on **Start recording**.
The Recording Service URL points to the CICS region where the live run is recorded.



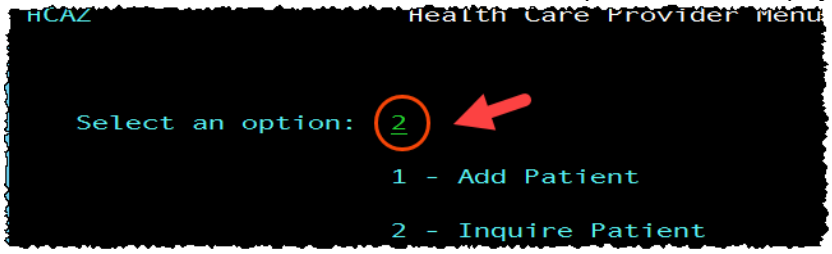
Click **OK** for the dialog below (our code on server is NOT the latest level).



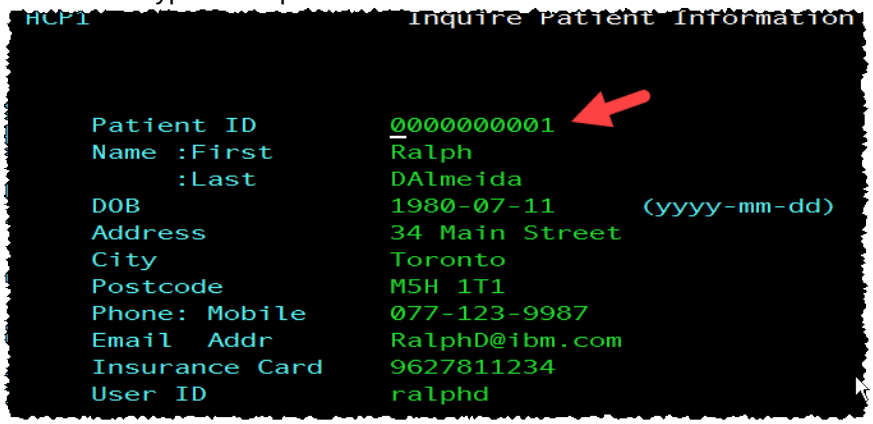
3.2.12 When recording is turned on, the message "**You are now recording**" appears.



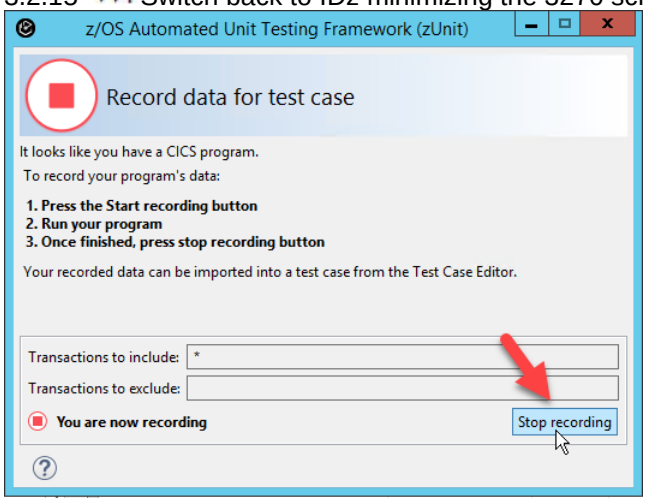
3.2.13 Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**.



3.2.14 Type **1** and press **enter**

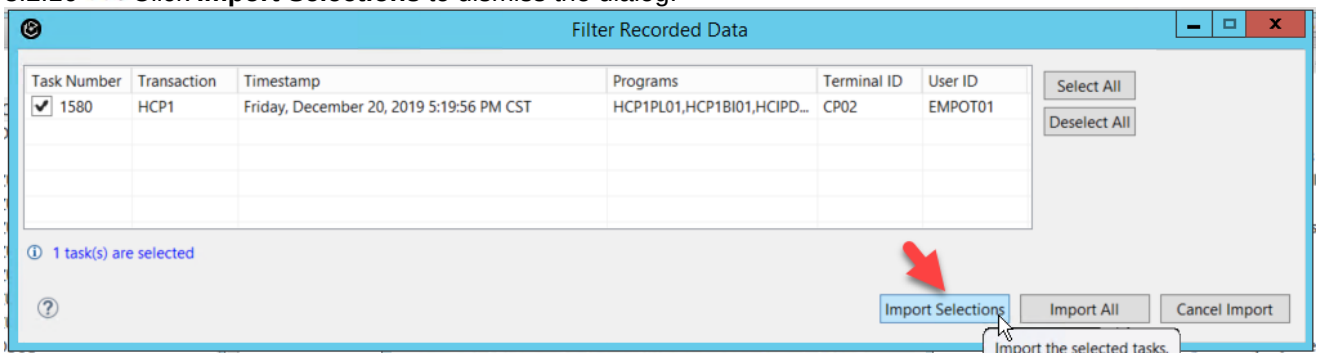


3.2.15 Switch back to IDz minimizing the 3270 screen and click **Stop recording**.



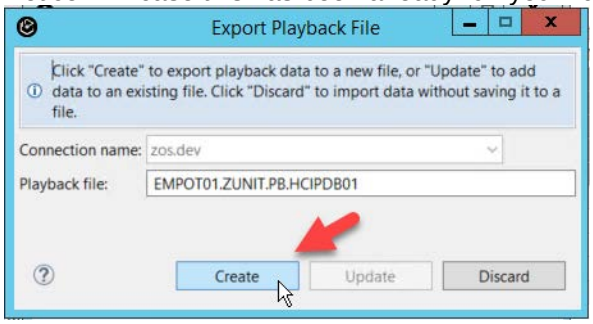
The dialog Filter Recorded Data is displayed.

3.2.16 Click **Import Selections** to dismiss the dialog.

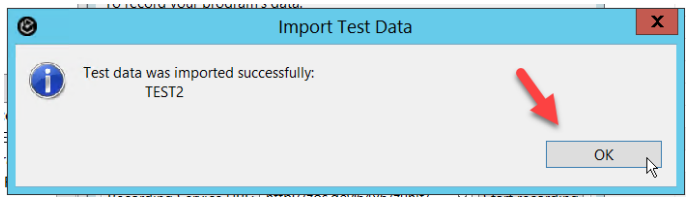


3.2.17 Click **Create** to export playback data to a z/OS data set

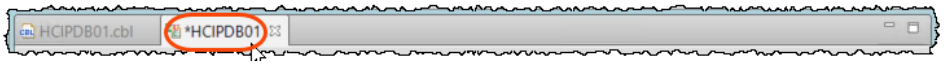
Notice – In case this has been already run you must select **Update** instead.



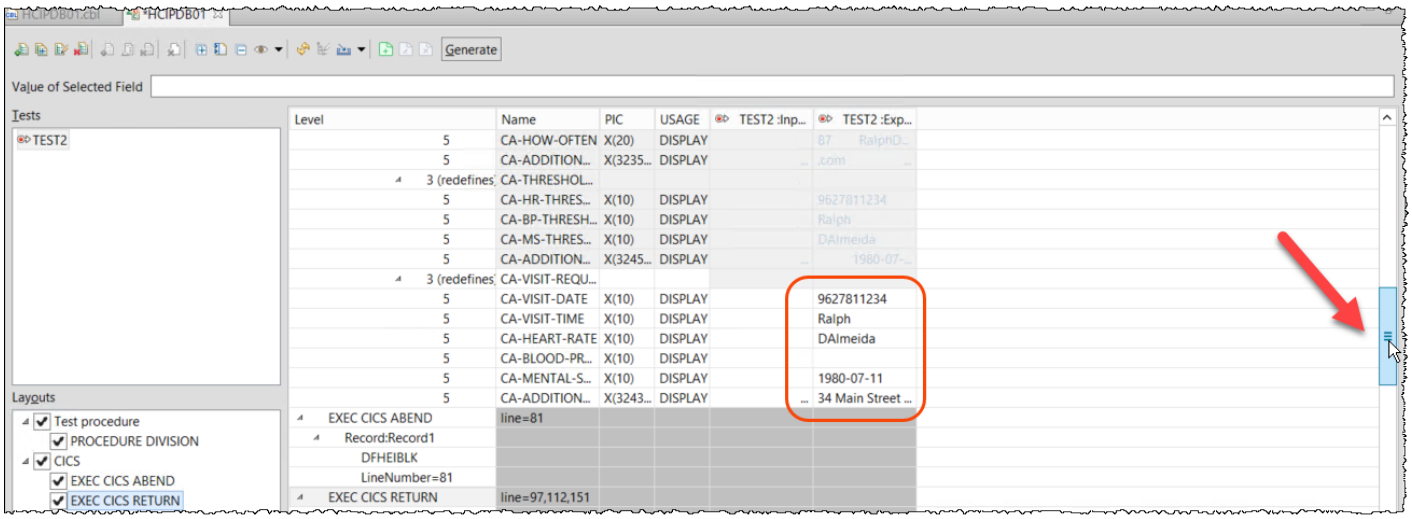
3.2.18 Click **OK** to dismiss the dialog



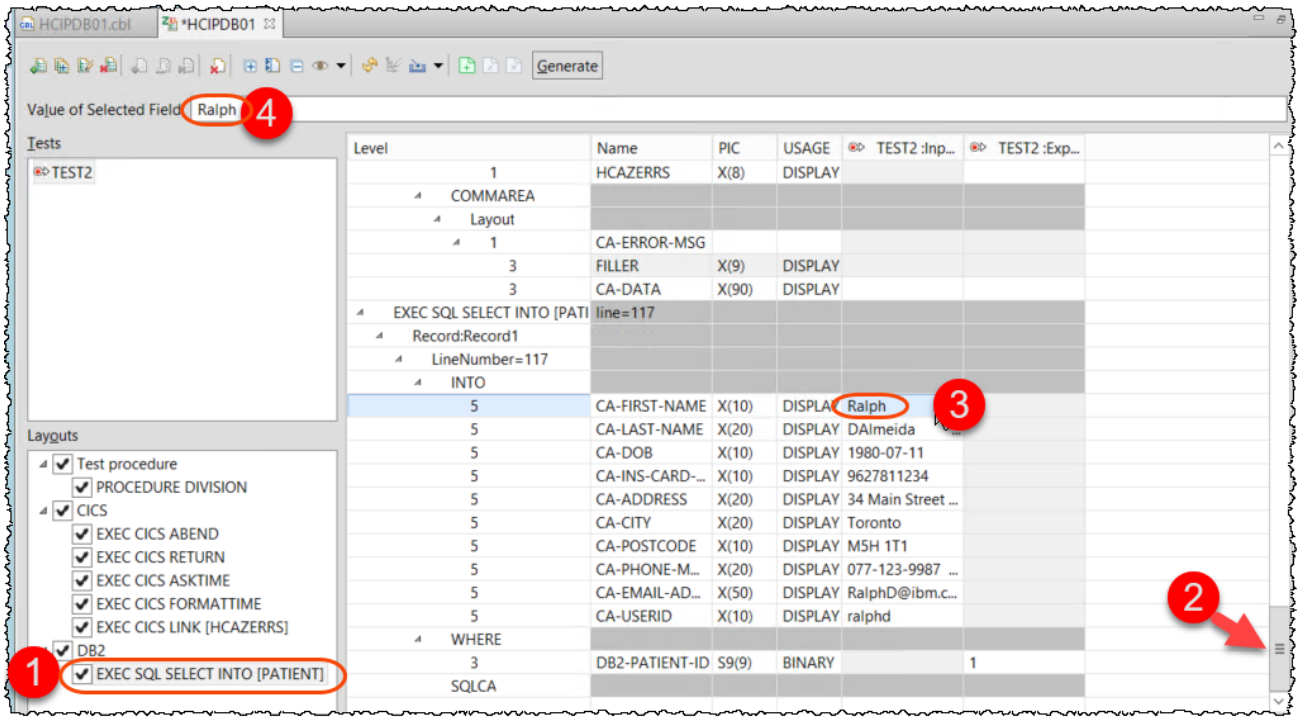
3.2.19 Double click on the **HCIPDB01** title to enlarge the view.



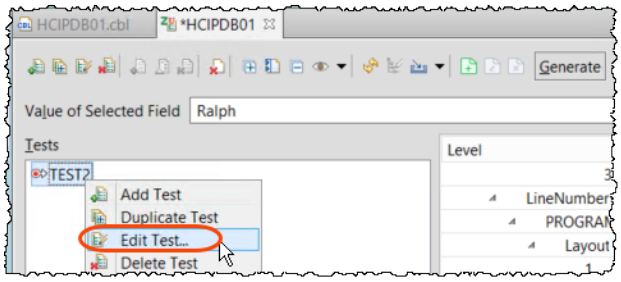
3.2.20 You now see a new test created in the editor and populated with the values from the live run.
Scroll down the editor and notice the data displayed on the screen that was captured..



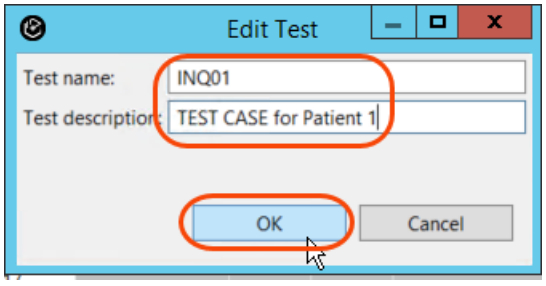
3.2.21 1 Select **EXEC SQL SELECT INTO (PATIENT)**to position the data layout for this statement.
2 Use the scroll bar to scroll down until the end,
3 Click on **Ralph** . and notice that value is displayed on the line 4 ”



3.2.22 ⏭ Right click on **TEST2** and select **Edit Test**



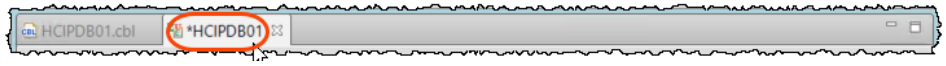
3.2.23 It is a good practice give a name for the test case.
⏭ Use a name like **INQ01** and a description like "**TEST CASE for Patient 1**": . Click **OK**



3.2.24 ⏭ Press **Ctrl+S** to save any changes.

Notice this save takes a while since values are saved on z/OS

3.2.25 ⏭ **Double click** again on the ***HCIPDB01*** title to shrink the view.

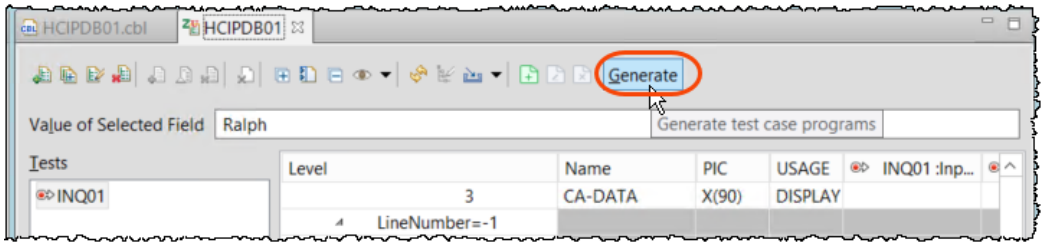


Section 4. Generate, build and run the unit test.

You will generate, build, and run the unit test.

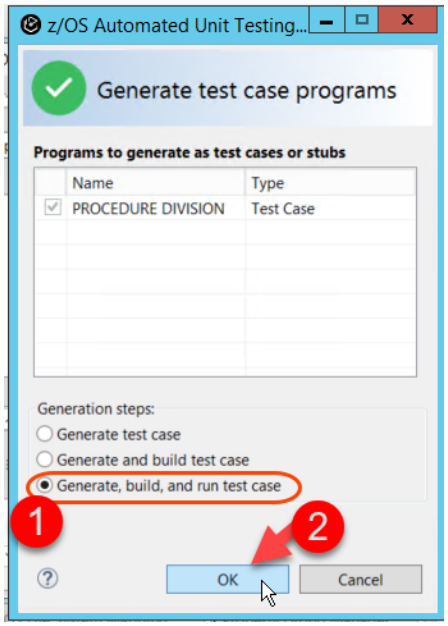
___ 4.1 Generating, building and running the test case program.

4.1.1 ⏭ Now that the expected input and output values have been set in the test case editor from the recorded run, **click** on **Generate** to generate the test case program.

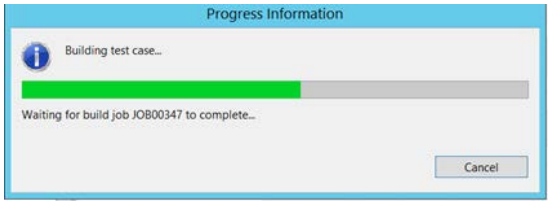


On the **Generate test case programs** dialog,

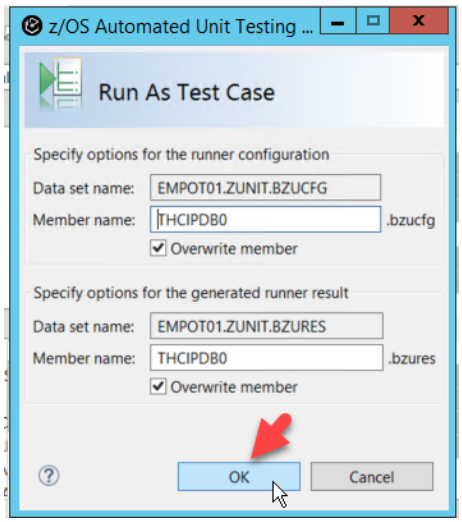
4.1.2 Select **Generate, build, and run test case** and then click **OK**.



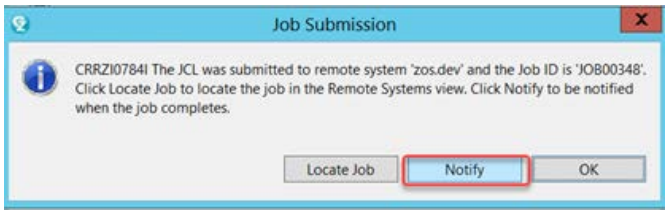
A **Progress information** dialog on the building will open.
This generation will take a while and a job is submitted to z/OS for execution. The COBOL test cases programs were compiled/link edited and are ready to run.



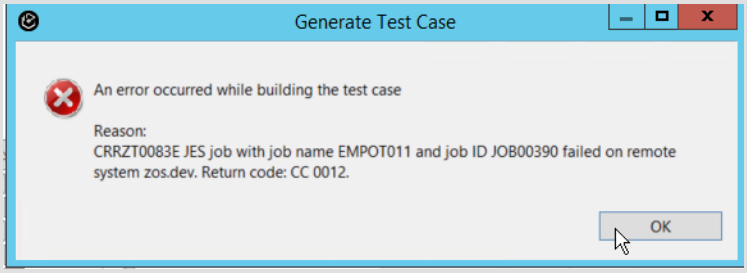
4.1.3 Once the building is complete, a **Run As Test Case** dialog will open, click **OK** to continue.



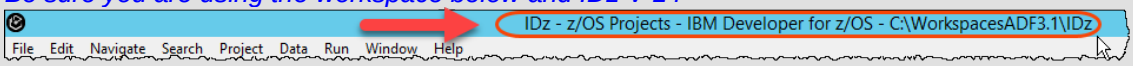
4.1.4 A **Job Submission** dialog opens, click **Notify** to be notified of job completion.



#1. Had a return code 12 as below?



You might be using a wrong IDz Workspace…
Be sure you are using the workspace below and IDz V 14

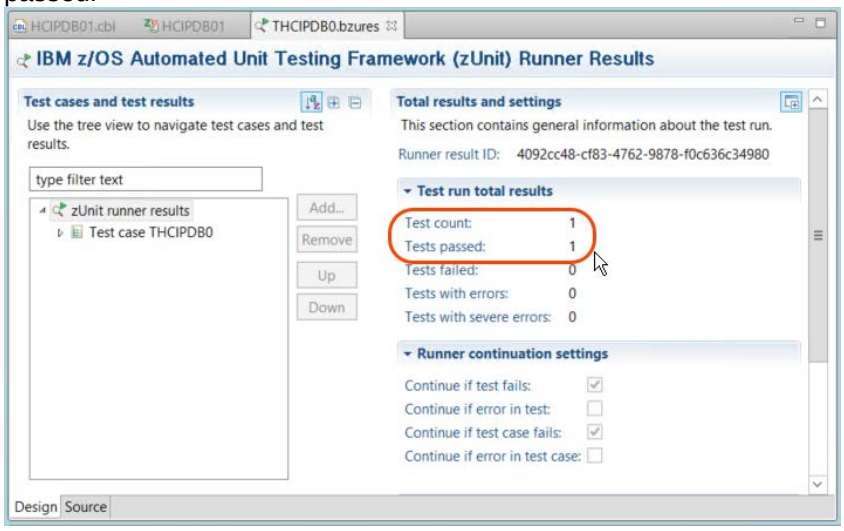


Call the instructors and
1. Close all opened editors.
2. Delete the data set EMPOT01.ZUNIT.PB.HCIPDB01 *and go back to record and try again*

#2. Had a "strange Test fail" ?

Call the instructor..
1. Delete the data set EMPOT01.ZUNIT.PB.HCIPDB01
2.All members from EMPOT01.ZUNIT.* *need to be deleted and the record started again.*

4.1.5 Once the unit test run has completed, the results screen is displayed showing test cases ran and passed.



You have now created a test case with data imported from a recorded run and have been successful in testing a CICS program without the need of a CICS environment.

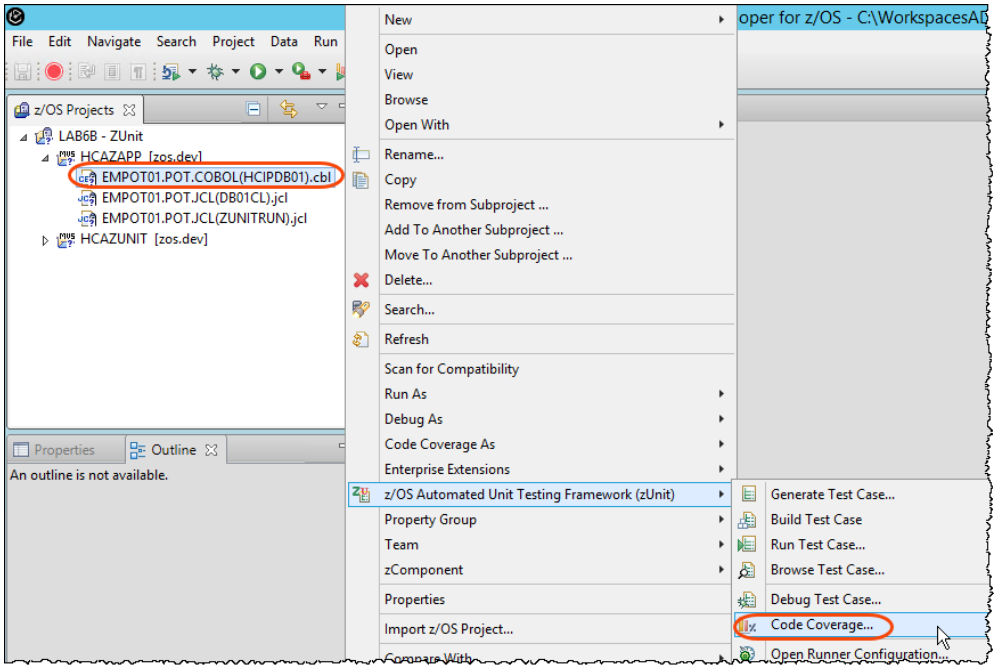
4.1.6 Use **Ctrl + Shift + F4** to close all opened editors.

___ 4.2 Running zUnit test case with Code Coverage

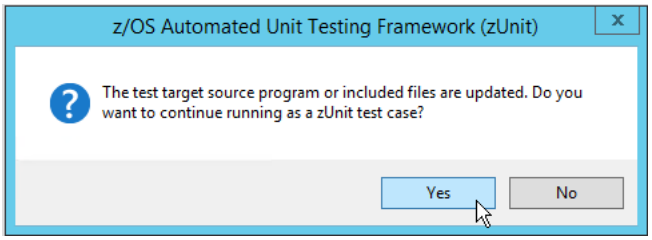
Code coverage analyzes a running program and generates a report of lines that are executed, compared to the total number of executable lines. Sometimes you need to be sure that your test is covering all capabilities of your program and that you are having the correct test cases.

Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, what is very handy..

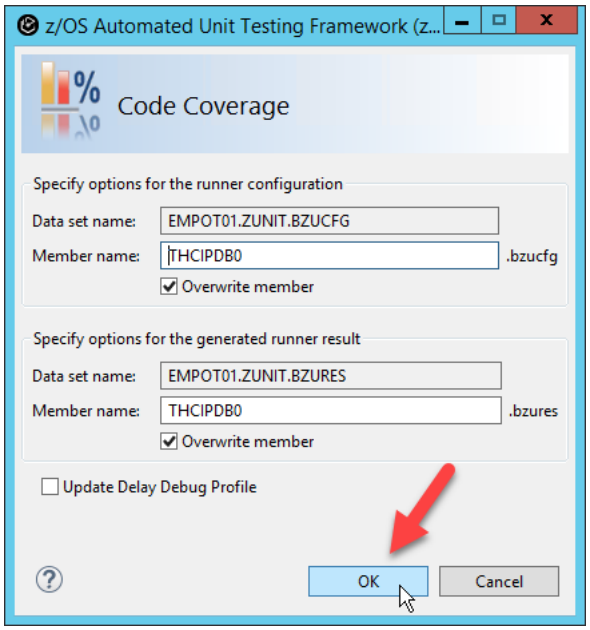
4.2.1 Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)->Code Coverage…**



4.2.2 Click **Yes** if the dialog below opens

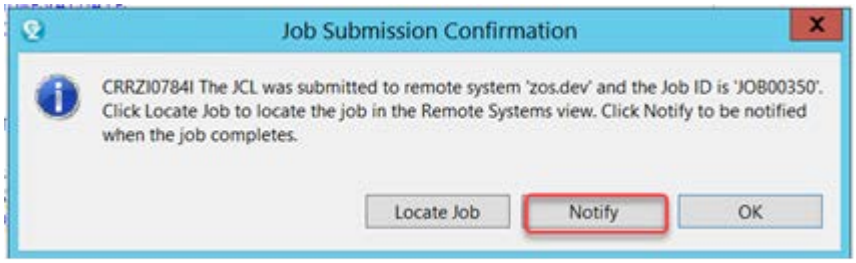


4.2.3 ⏭ Click **OK** for this dialog. Be sure that *Update Delay Debug Profile* is un-checked.

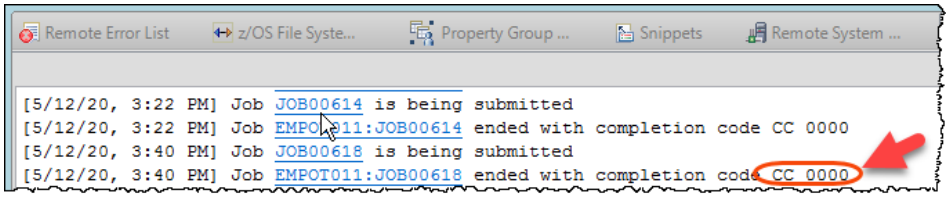


4.2.4 A Job Is submitted for batch execution..

⏭ Click **Notify** on the Job Submission Confirmation dialog.

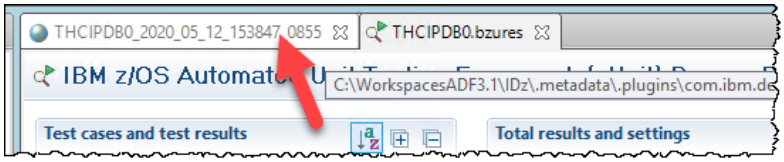


⏭ Make sure the job completes with completion code of **0**.



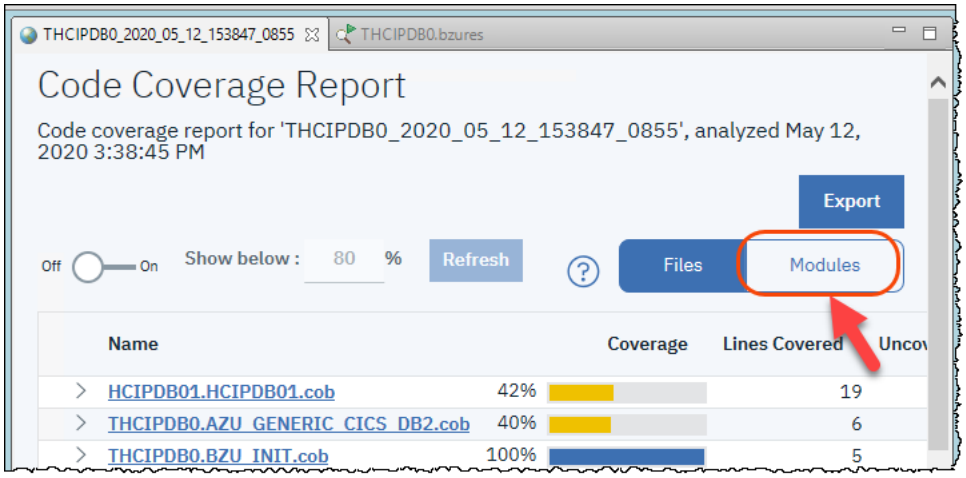
4.2.5 Once again the zUnit Results are displayed and the Test passed.

⏭ Click on **THCIPDB0_yyyy_mm_dd_xxxxxxx** tab to see the *Code Coverage* report

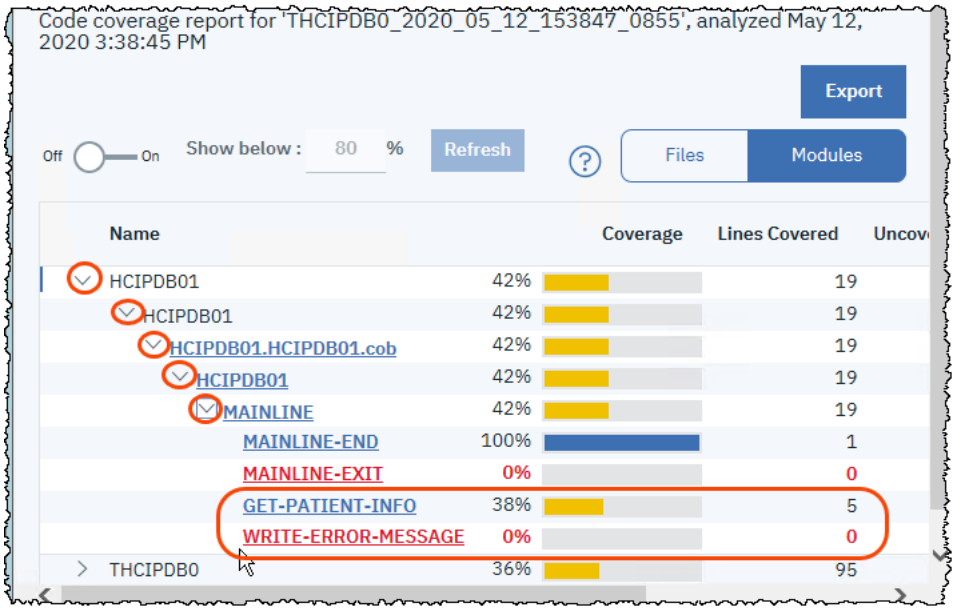


4.2.6 The code coverage report shows all COBOL programs executed for this specific test case.
It shows the coverage of the COBOL programs generated for zUnit to perform the test as well our *HCIPDB01* COBOL program..

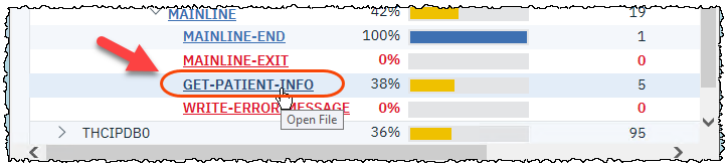
⏭ Since we are just interested in the code coverage of program being tested click on **Modules**:



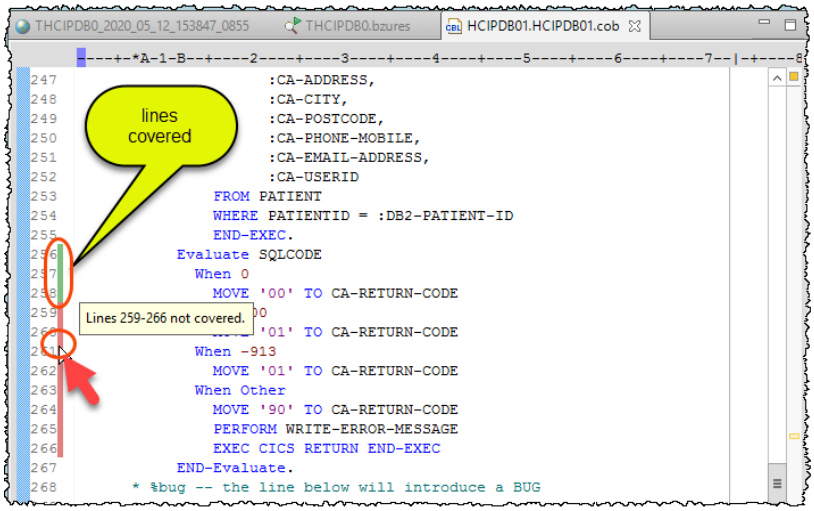
4.2.7 ⏭ **Expand the nodes** as below to see the COBOL paragraphs.
Notice that *GET-PATIENT-INFO* has 38% of coverage and that *WRITE-ERROR-MESSAGE* was not covered at all on this test case.



4.2.8 ⏭ Click on **GET-PATIENT-INFO** to see more details



4.2.9 **Scroll down** and **move the mouse to the colored areas on left**.
The lines covered by this test case are in **green** and the lines not covered are in **red.**
Also from the previous image we can see the WRITE-ERROR_MESSAGE paragraph is not covered at all.



4.2.10 From this report we can see that the test cases created for this program are not sufficient ..
The developer could go back to the test cases editor (step 3.2.19 above) and manually add test cases that cover other paragraphs. Due the lack of time we will not cover that here, but if you have extra time you can try it. Ask the instructor for guidance.

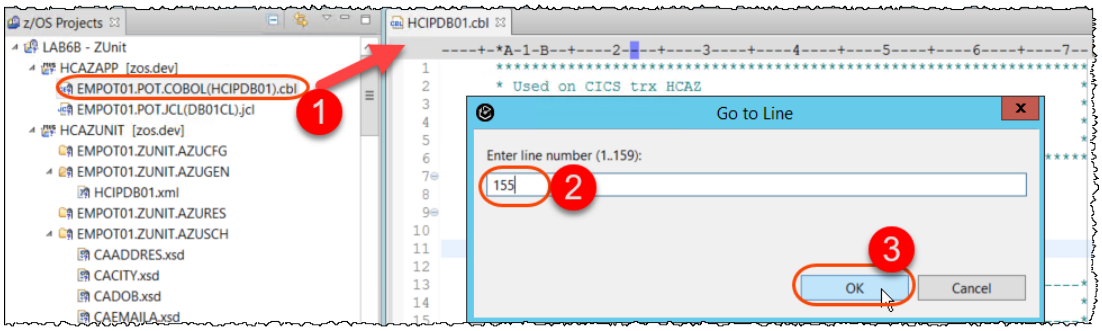
Section 5. Introduce a bug in the program and rerun the unit test.

Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

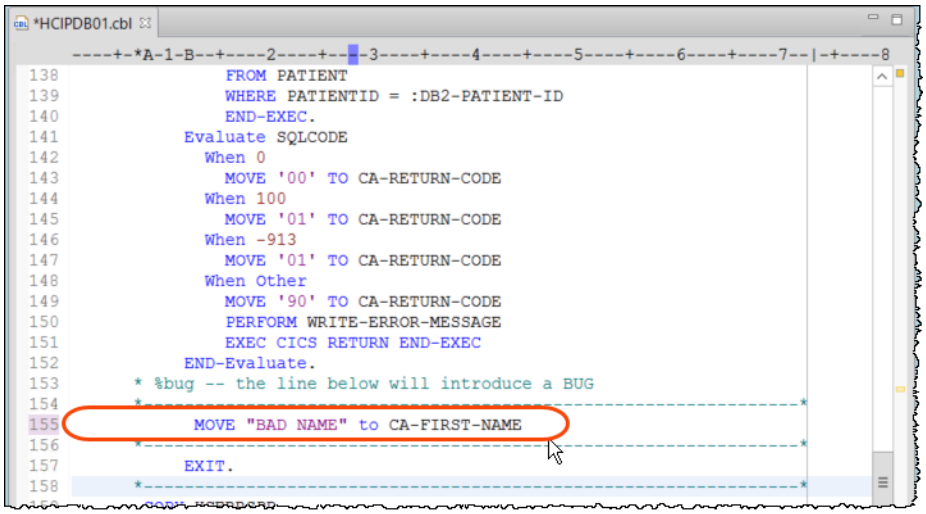
___ 5.1 Modifying the program and introduce a bug

.

5.1.1 On IDz, close all active windows by pressing **Ctrl+Shift+F4**
Open **EMPOT01.POT.COBOL (HCIPDB01).cbl** by double clicking on it in the z/OS Projects view.5.1.2
On the editor, go to line 155 by pressing **Ctrl+L** and entering **155** and clicking **OK**.



5.1.3 ⏭ Change the line 155 removing the * from the statement that moves “BAD NAME”,
⏭ Press **Ctrl+S** to save the changes.

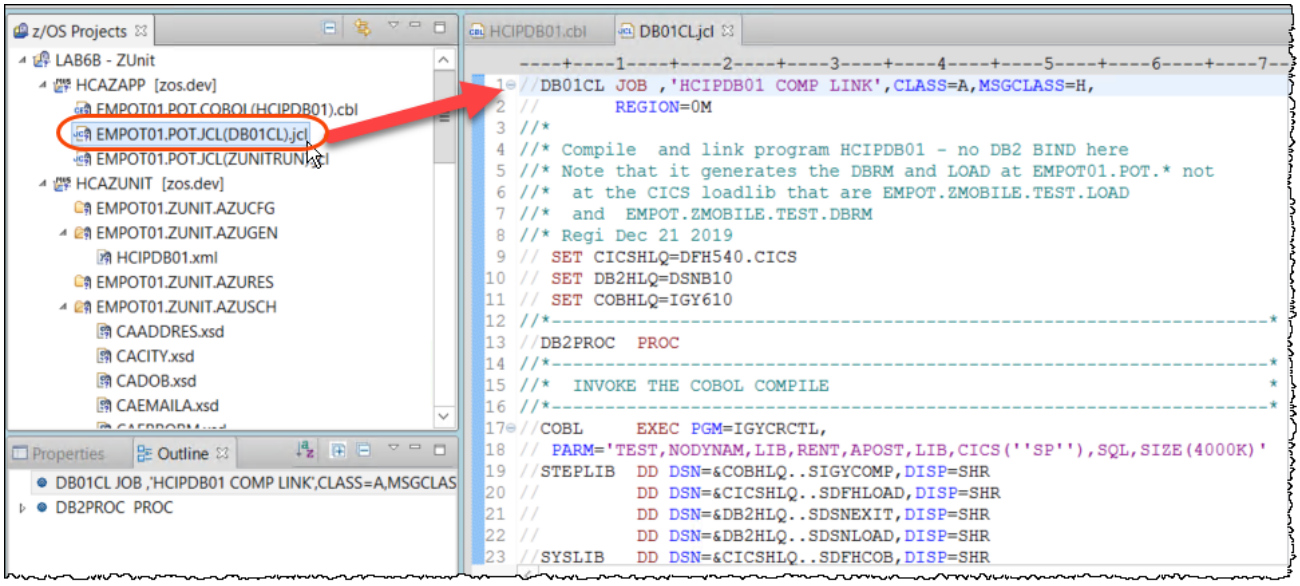


___ 5.2 Rebuilding the changed program without deploying to CICS

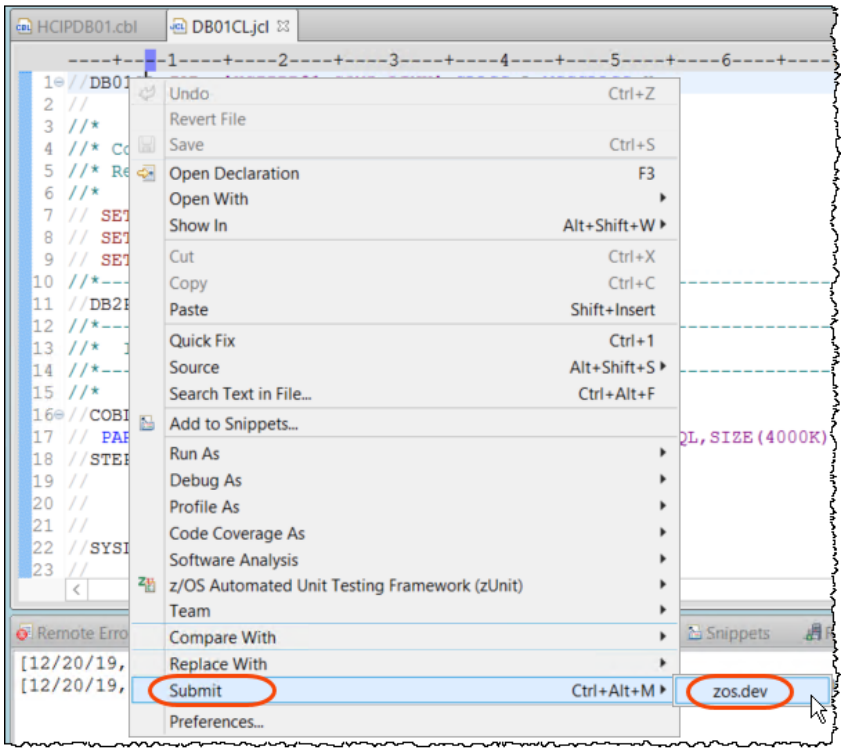
.

5.2.1 ⏭ On the z/OS Projects view, double click **EMPOT01.POT.JCL (DB01CL).jcl,** to edit .

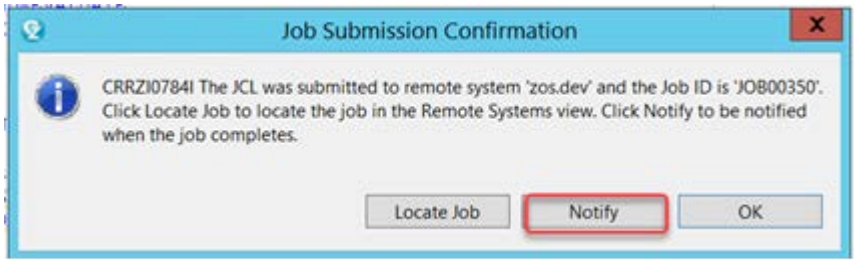
This JCL will compile and link the changed COBOL program *HCIPDB01* using a working PDS to store the load module.
Notice that DB2 bind is not necessary since DB2 SQL calls will be intercepted when running the generated zUnit test cases.



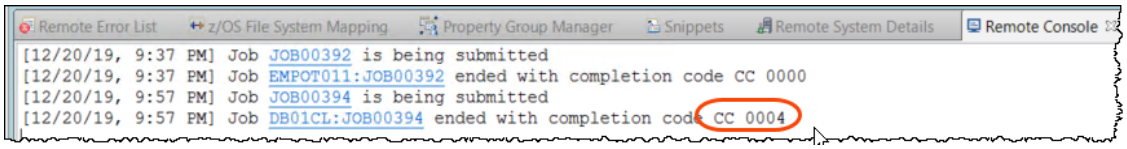
5.2.2 Right click and select **submit > zos.dev** to submit this job for execution



5.2.3 Click **Notify** on the Job Submission Confirmation dialog.

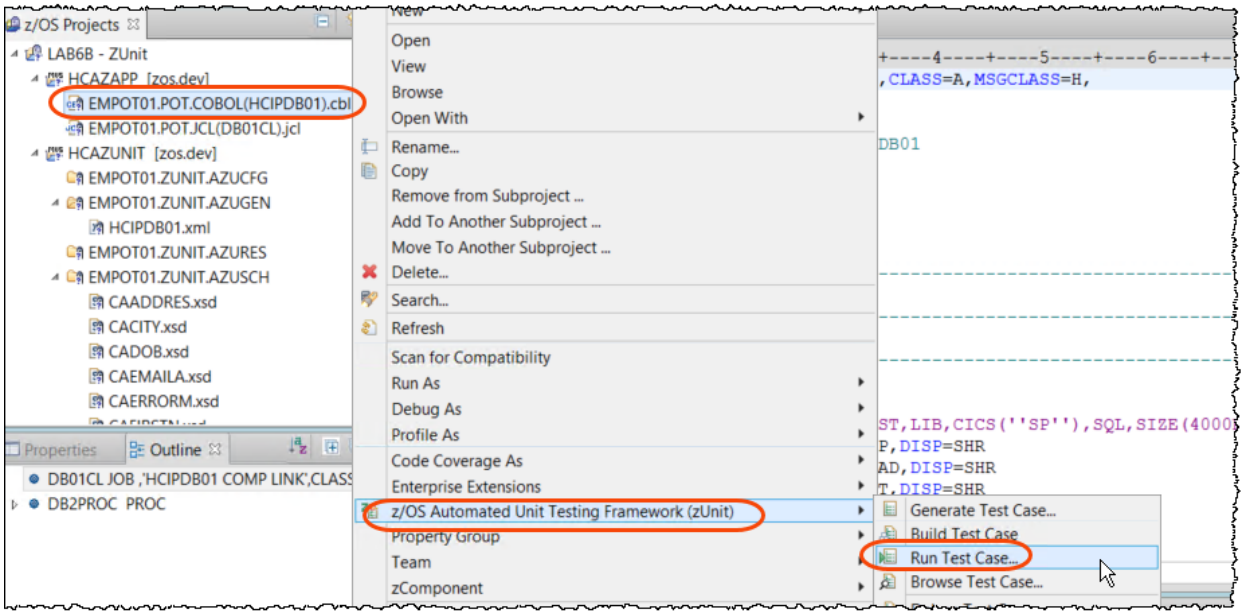


Make sure the job completes with completion code of **4**.

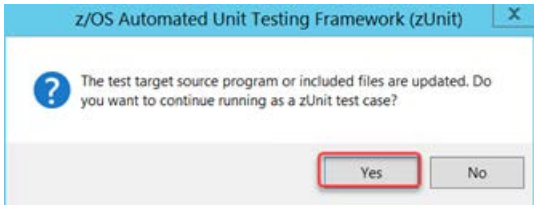


___ 5.3 Rerunning the test case

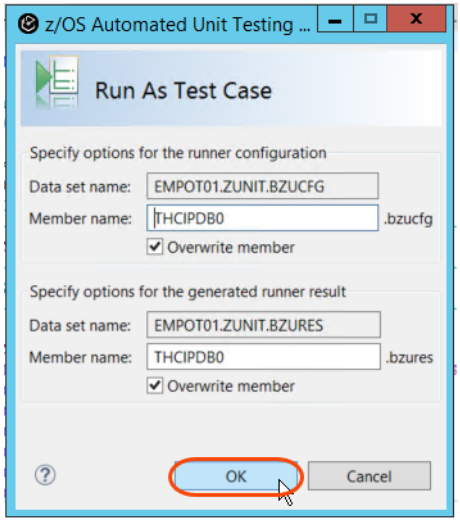
5.3.1 On the z/OS Projects view, select **EMPOT01.POT.COBOL(HCIPDB01).cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)->Run Test Case..** action.



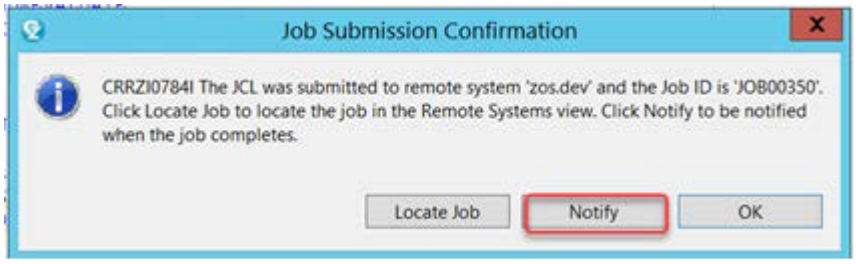
5.3.2 Click **Yes** to the following dialog to continue running the test case.



5.3.3 Click **OK** to the **Run As Test Case** dialog.

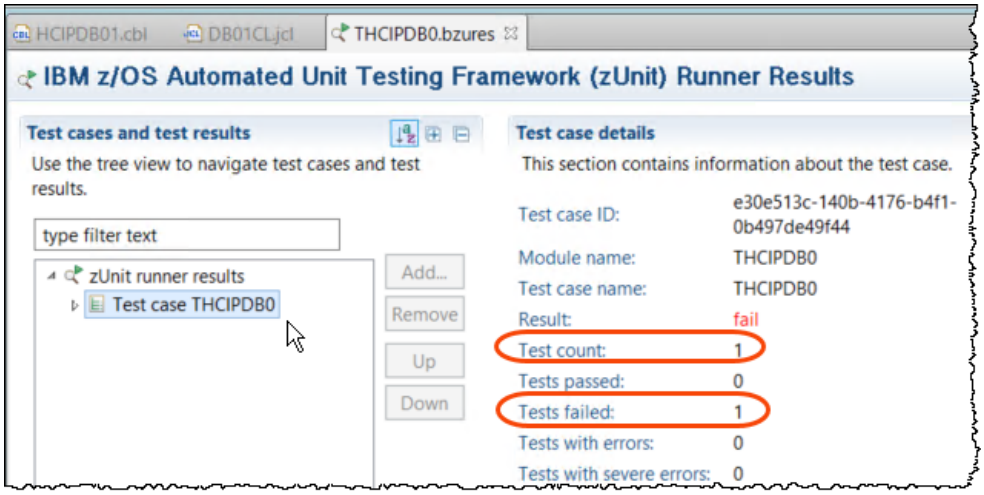


5.3.4 ⏭ Click **Notify** on the Job Submission Confirmation dialog.

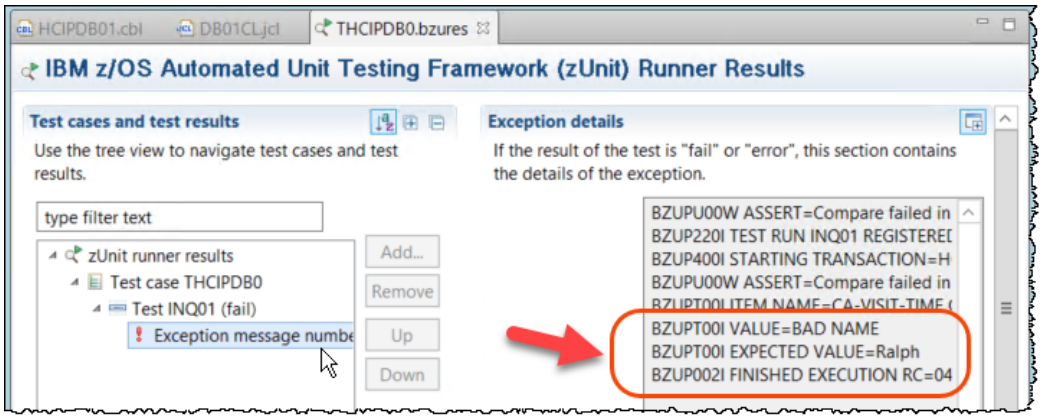


5.3.5 When the test run completes, the test results will be displayed, and it showed 1 test was ran and it failed.

⏭ Click **Test case THCIPDB0.** The failure is expected because the expected value of the error message is different from the actual value.



5.3.6 ⏭ Expand **Test case THCIPDB0, Test INQ01 (fail)** and click **Exception message number** to verify the value received versus the expected value and verify the failure



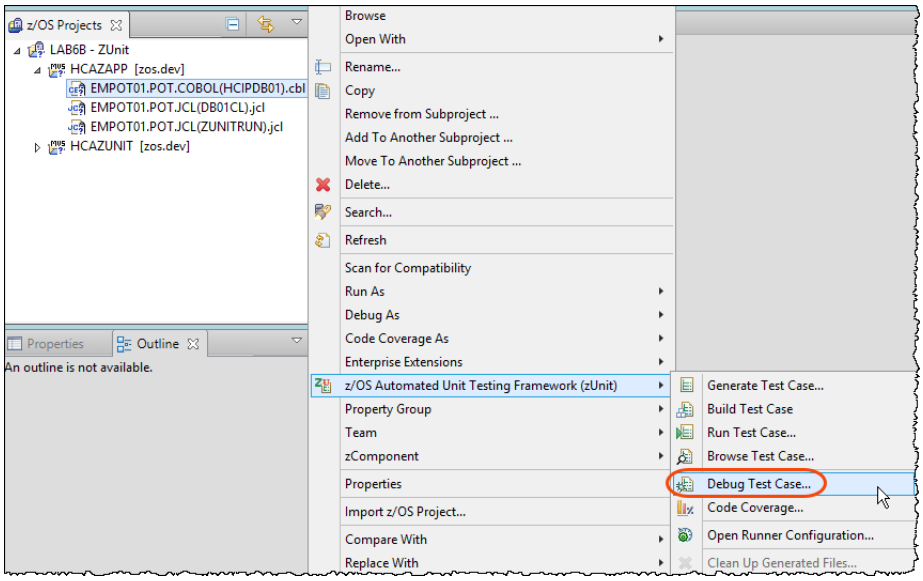
___ 5.4 Running zUnit test case with debugging

The Debug dialog will debug the test case. Note that will debug the zUnit programs, the generated COBOL and also the COBOL program that you are testing. Below one example.

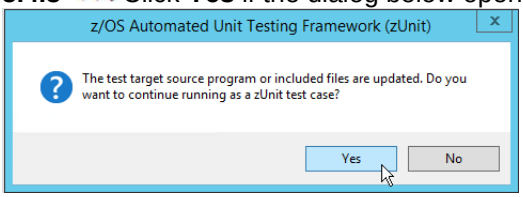
Notice that you are debugging a batch job without need to have CICS and DB2 active, what is very handy.

5.4.1 ⏭ Close all active windows by pressing **Ctrl+Shift+F4**

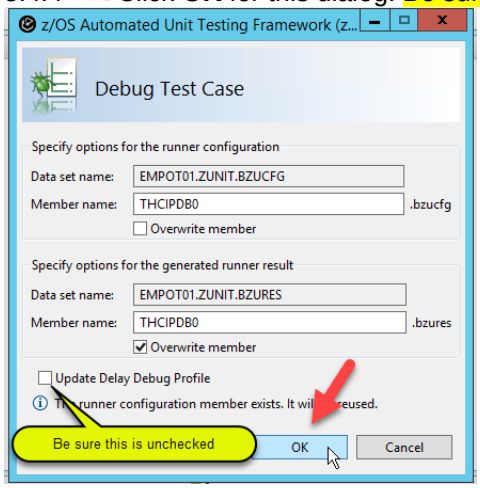
5.4.2 ⏭ Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl**
and select **z/OS Automated Unit Testing Framework (zUnit) > Debug Test Case…**



5.4.3 ⏭ Click **Yes** if the dialog below opens

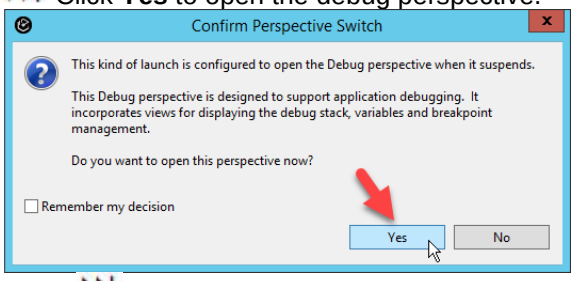


5.4.4 ⏭ Click **OK** for this dialog. Be sure that Update Delay Debug Profile is un-checked.

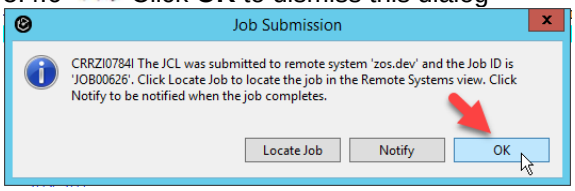


5.4.5 A Job Is submitted for batch execution..

Click **Yes** to open the debug perspective.



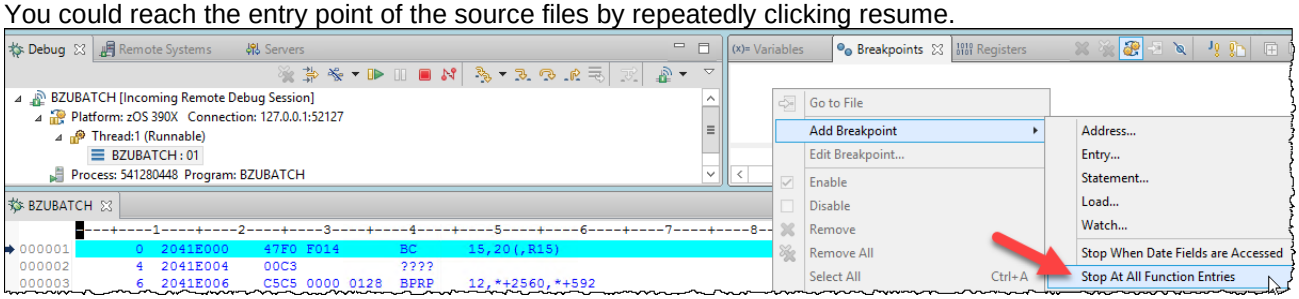
5.4.6 Click **OK** to dismiss this dialog



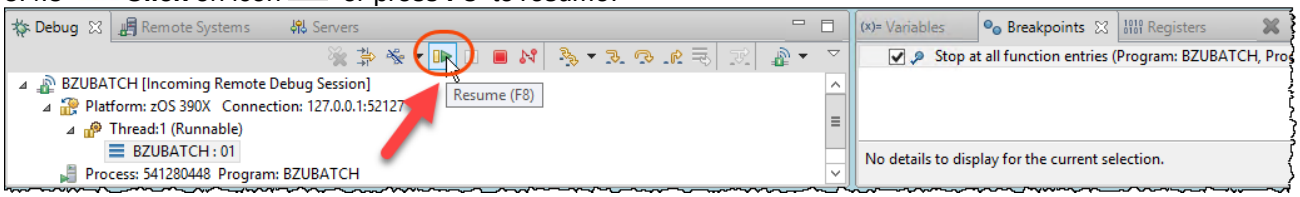
5.4.7 Notice that the first program being debugged is the zUnit engine (BZUBATCH) that is not a COBOL

Using the Breakpoints view, right click on it and be sure that **Add Breakpoints > Stop At All Functions Entries**. is selected, If not you must select it

You could reach the entry point of the source files by repeatedly clicking resume.

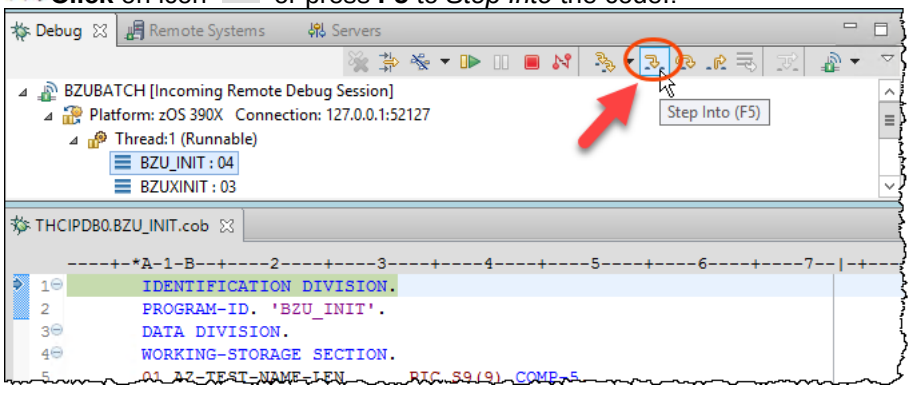


5.4.8 **Click** on icon or press **F8** to *resume*.

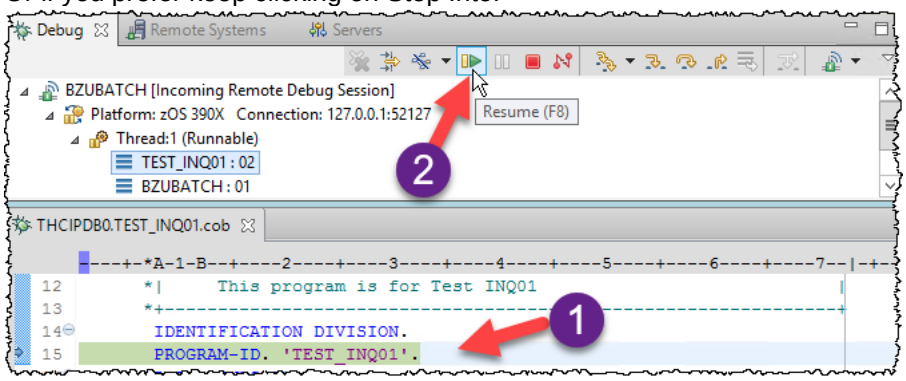


5.4.9 This COBOL generated program **BZU_INIT** will start in debug mode..

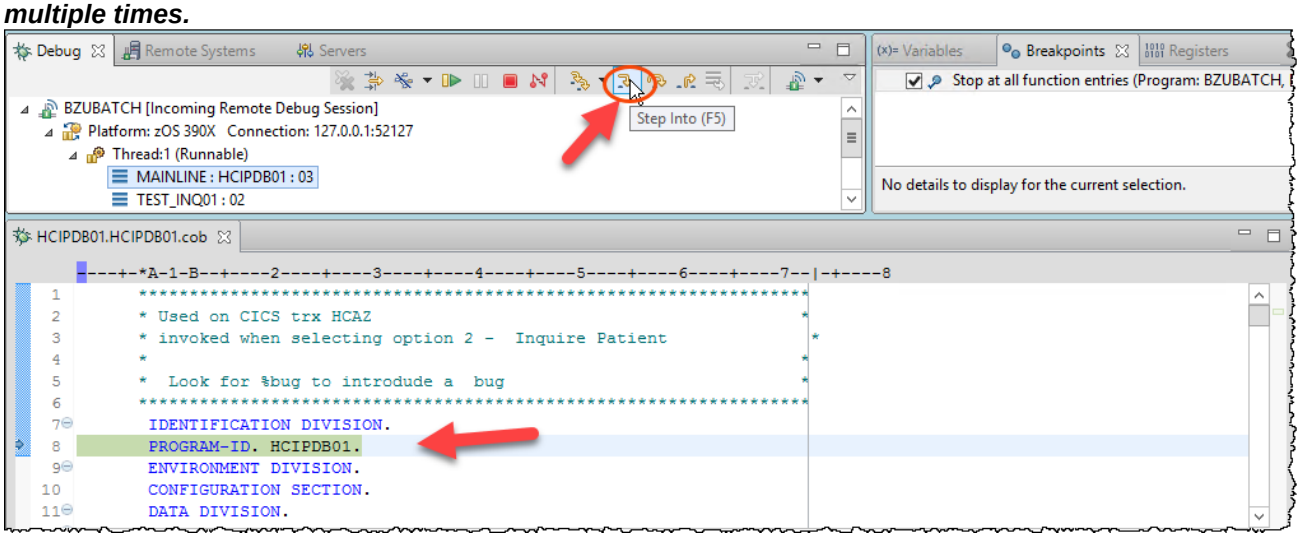
Click on icon or press **F5** to *Step Into* the code.:



5.4.10 **Click** on icon or press **F5** to *Step Into* the code until you reach the program **TEST_INQ01**
Click on icon or press **F8** to *resume*..
Or if you prefer keep clicking on Step Into.

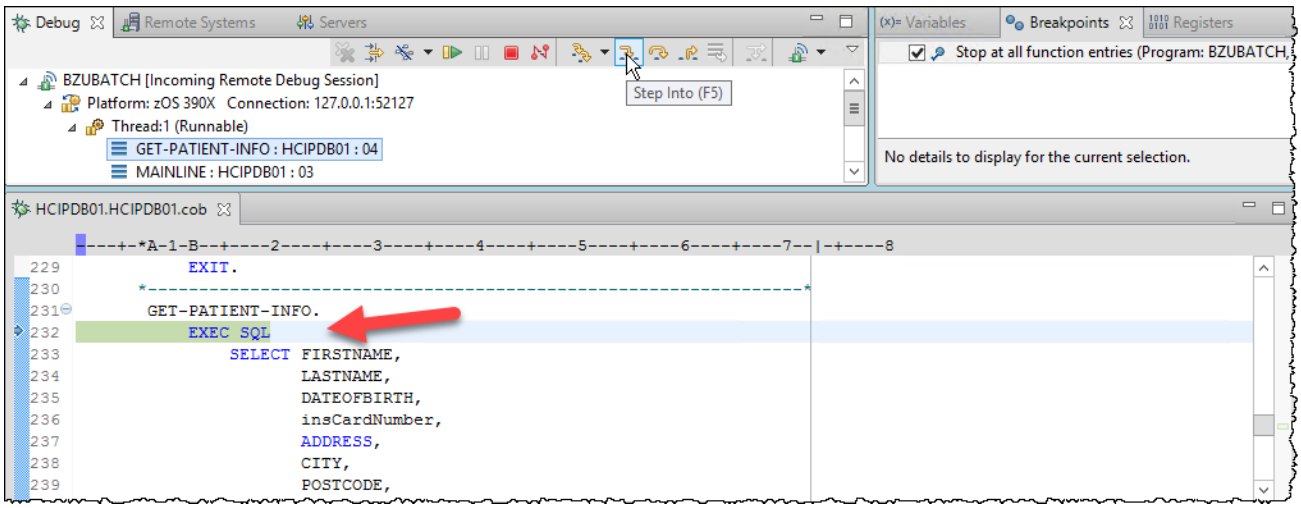


5.4.11 Once the debug stops on your program **HCIPDB01** click on icon (or **F5)** to *Step Into* the code ***multiple times.***

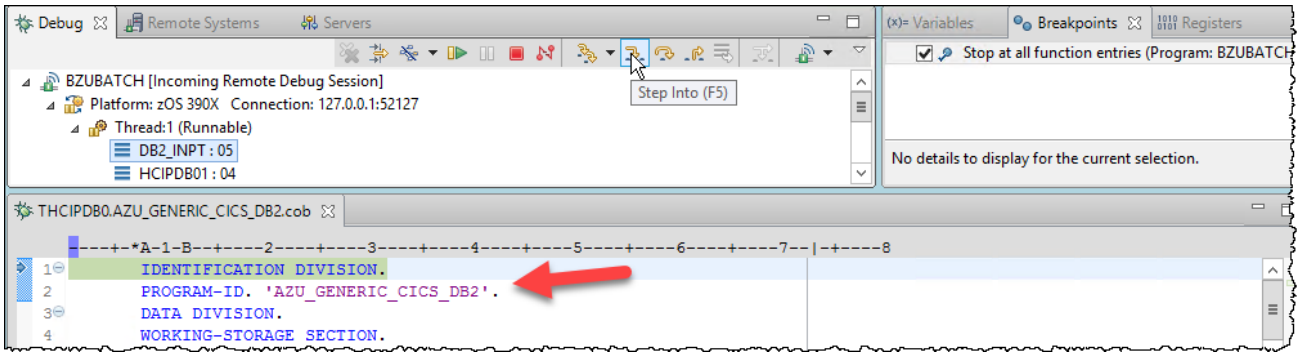


5.4.12 When you see the ***SQL SELECT*** statement you may use the step into to verify that you are using "stubbed code to get the data from DB2..

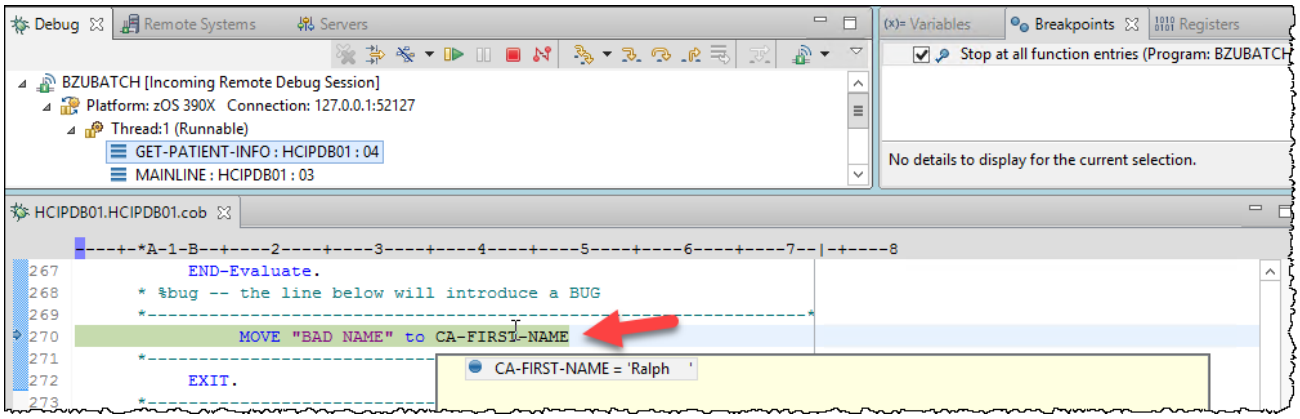
...



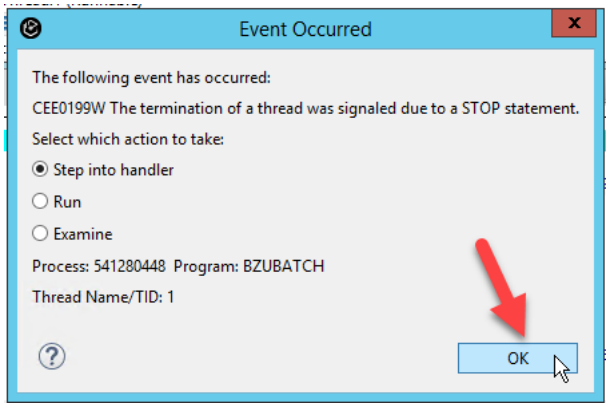
5.4.13 Keep clicking on icon (or **F5)** to *Step Into*
Noticed that the DB2 is not being invoked to access the DB2 table a program named **AZU_GENERIC_CICS_DB2** is invoked to get the test case data that you recorded.



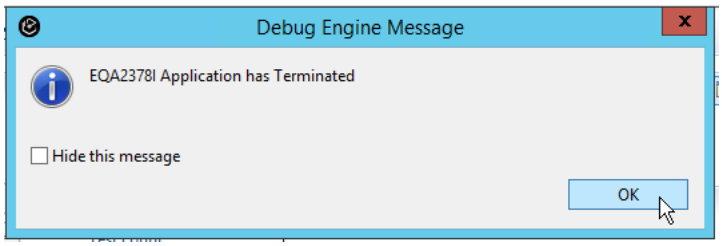
5.4.14 You also can see the BUG that you introduced. Move the mouse to **CA-FIRST-NAME** to see the field value (Ralph).



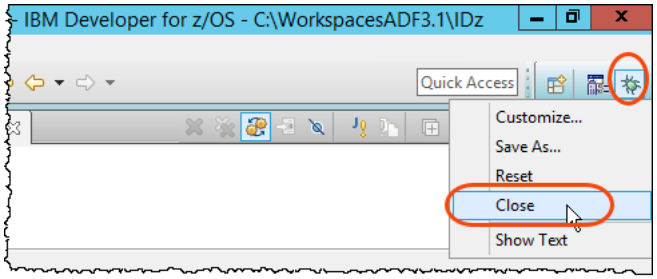
5.4.15 Fell free to continue debugging until the end of the test case execution Or just keep clickin on icon (**F8**) to *resume*..
You will see that the debug ended when you have the dialog below. Click **OK**



5.4.16 Click **OK** to terminate the debug



5.4.17 On top right corner, right click on icon and select **Close** to close the debug perspective



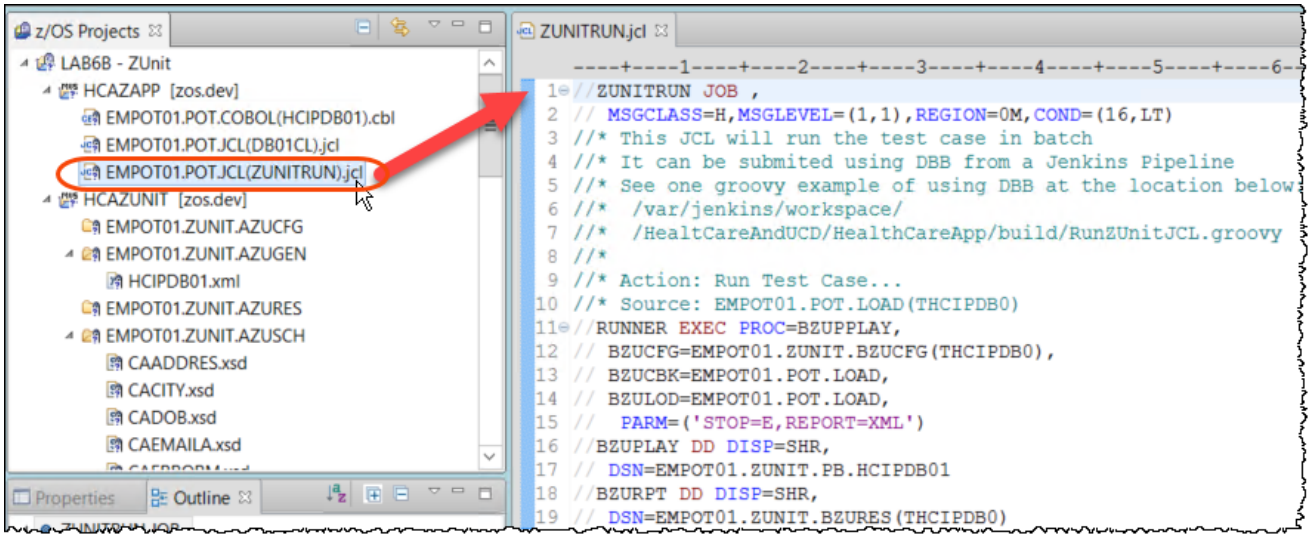
Section 6. Run the unit test from a batch JCL.

Once a test case has been generated and built, it can be executed using a batch run via JCL.
This would enable it to be ran as part of an automated process or pipeline..

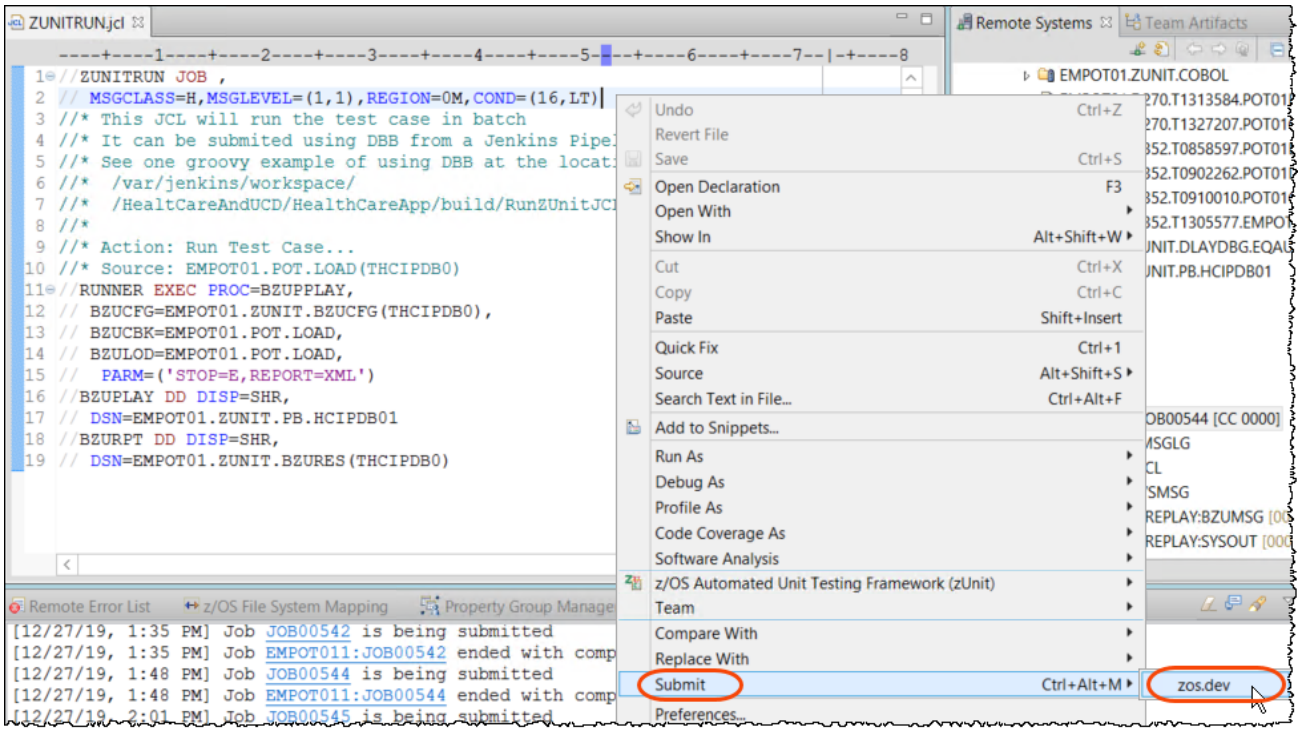
___ 6.1 Running the unit test from a batch JCL

6.1.1 Close any active windows by pressing **Ctrl+Shift+F4**.

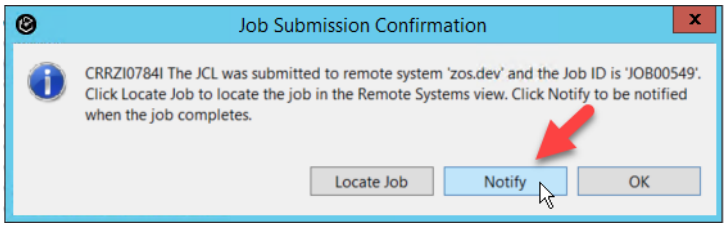
6.1.2 On the IDz **z/OS Projects** view, double click **EMPOT01.POT.JCL(ZUNITRUN).jcl** to open it.
This JCL will run the test case that you created using a batch job.



6.1.3 Right click the editor and select **Submit > zos.dev**.

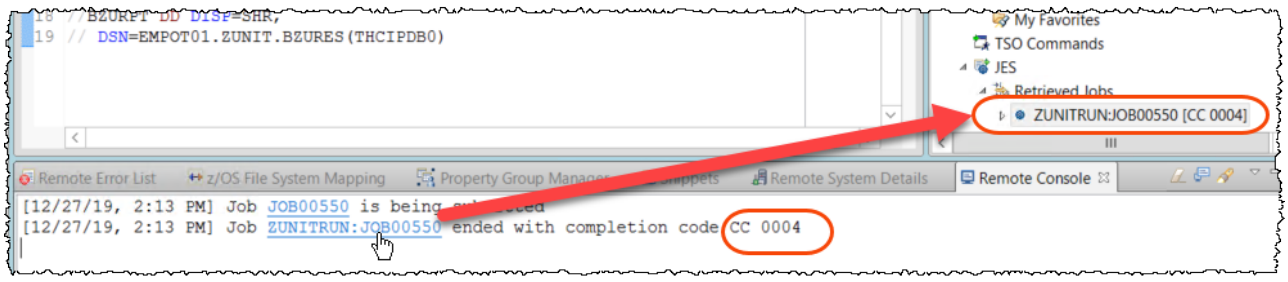


6.1.4 Click **Notify** on the Job Submission Confirmation dialog.

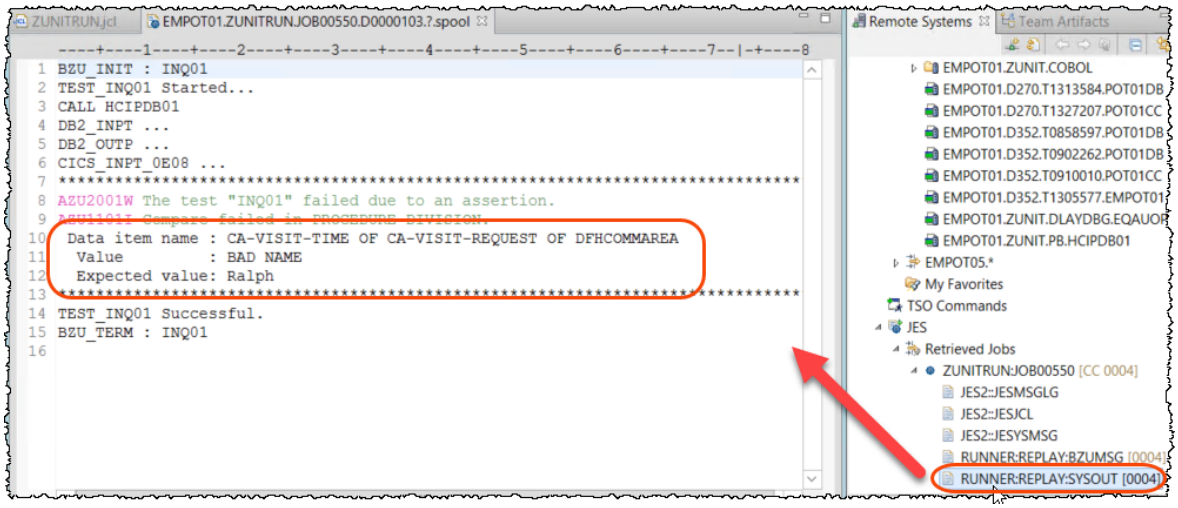


6.1.5 You MUST have **004** as return code.

Click on **ZUNITRUN:JOB00xxx**

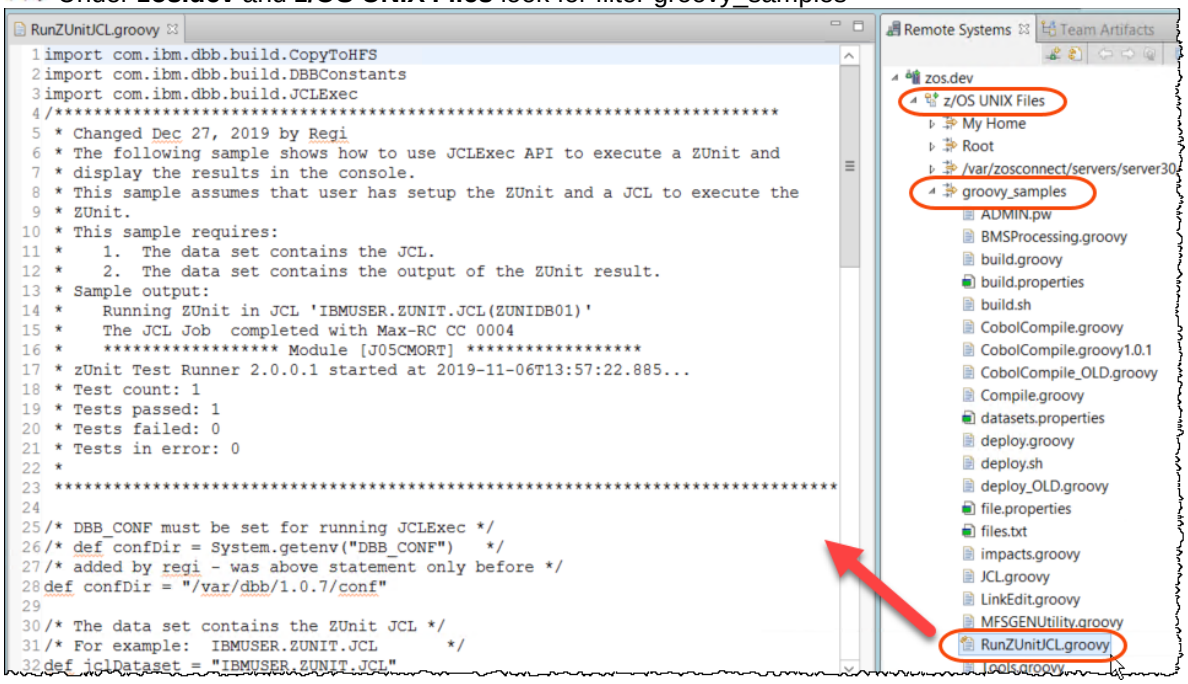


6.1.6 ⏭ Expand **ZUNITRUN:JOB00xxx** and verify the results double clicking on **RUNNER:REPLAY:SYSOUT.**



6.1.7 This JCL could be executed using DBB and part of a Jenkins Pipeline.
You may see an example of a groovy script used by DBB at the USS file below:
/var/jenkins/workspace/HealtCareAndUCD/HealthCareApp/build/ RunZUnitJCL.groovy

⏭ Under **zos.dev** and **z/OS UNIX Files** look for filter groovy_samples



Notice that this capability allows to invoke zUnit using pipelines like Jenkins.

Congratulations! You have completed the Lab.