

# Using IBM Watsonx Code Assistant for Z for Application Discovery & COBOL Refactoring

## Hands-on lab guide

**David Hawreluk**

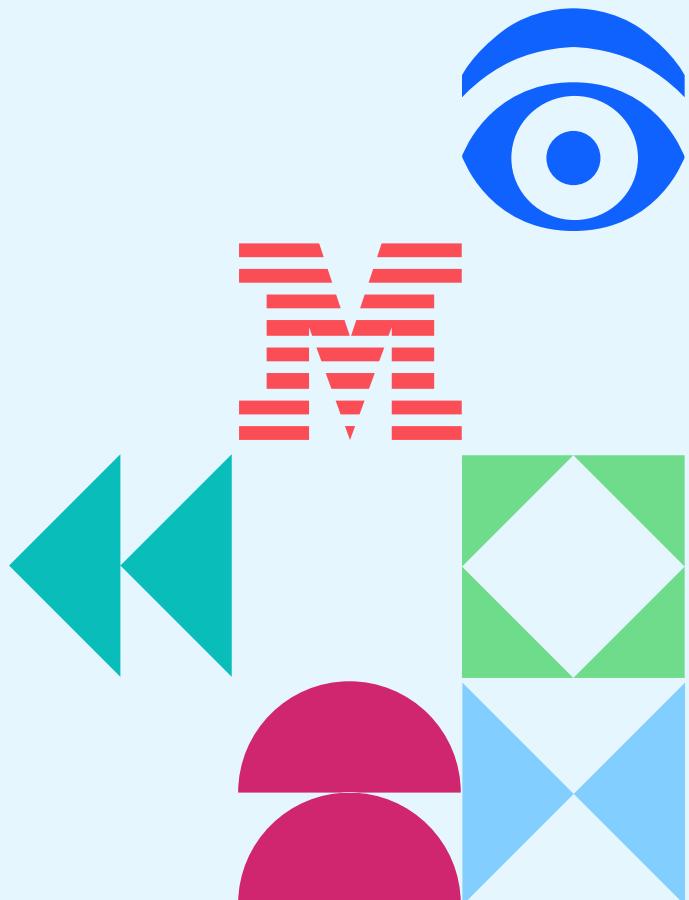
[dhawrel@us.ibm.com](mailto:dhawrel@us.ibm.com)

Brand Technical Specialist - Z DevOps, ADDI

**Frank Hernandez**

[Frank.Hernandez1@ibm.com](mailto:Frank.Hernandez1@ibm.com)

Brand Technical Specialist – Z AIOPS, WCA4Z



# TABLE OF CONTENTS

---

**00** OVERVIEW

---

**01** UNDERSTAND

---

**02** REFACTOR

---

**03** TRANSFORM

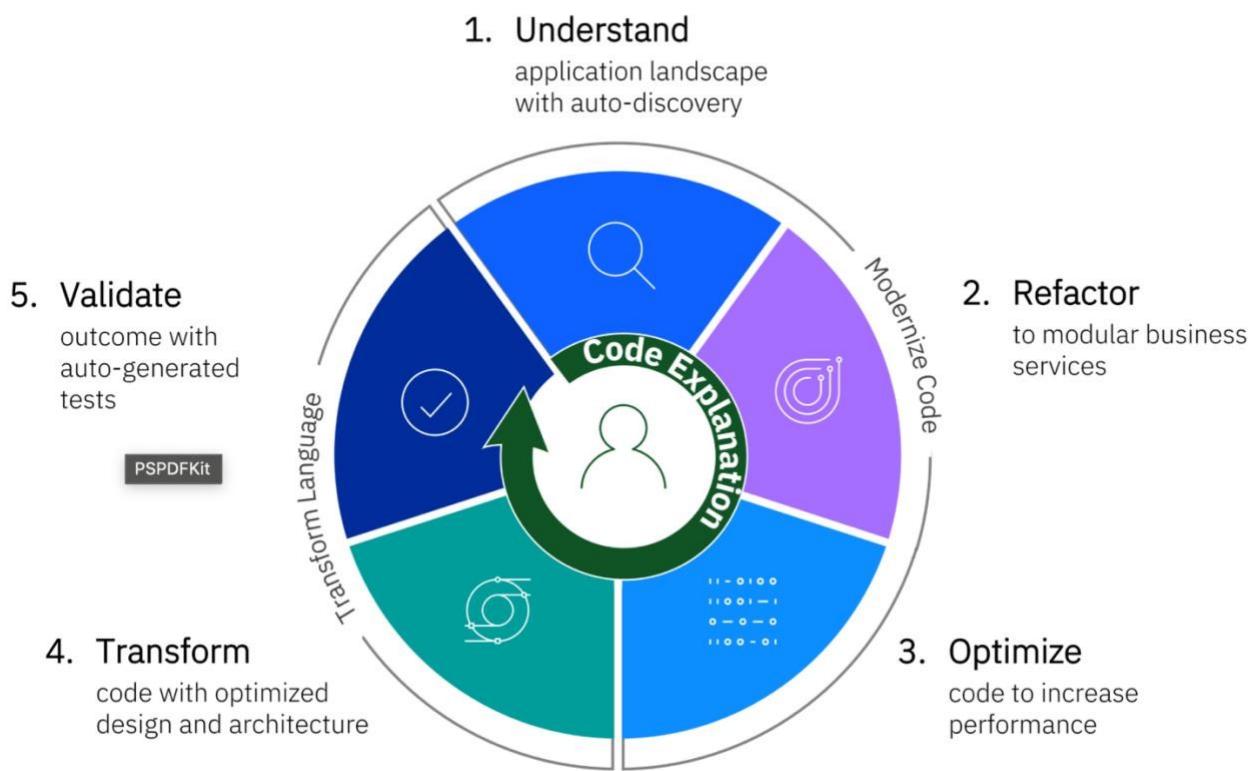
---

**04** Validation

# 1 00 Overview

---

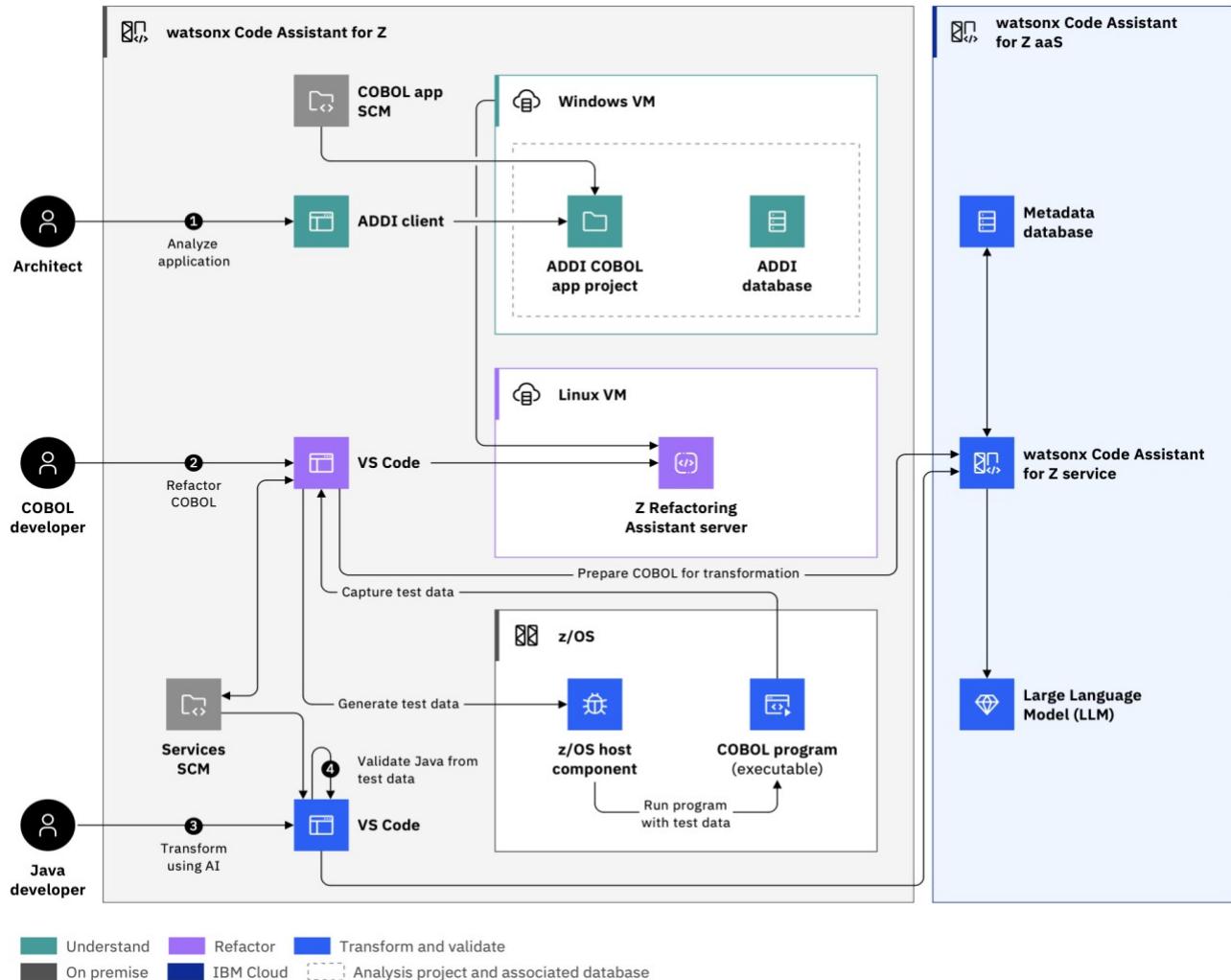
## IBM watsonx Code Assistant for Z modernization experience



If the modernization strategy involves converting the COBOL service to Java, the COBOL developer can prepare the COBOL code for transformation, sending the necessary metadata to the Db2 database in IBM Cloud for Watsonx Code Assistant for Z. After reviewing and compiling the COBOL code, the COBOL developer can use the IBM Watsonx Code Assistant for Z Validation Assistant to generate COBOL test data. This test data can be used later in the process by the Java developer to validate that the Java service is semantically equivalent.

The Java developer uses the IBM Watsonx Code Assistant for Z extension in Visual Studio Code (VS Code) to connect to the Watsonx Code Assistant for Z service in IBM Cloud. This service is provisioned by using a Cloud account and provides access to the AI foundation model. This foundation model

also called a large language model (LLM), is trained on many programming languages and is fine-tuned for converting COBOL to Java. Each developer accesses the service by using an individual API key and connects securely to use the AI. The Java developer uses the COBOL service in the SCM repository to generate Java classes and business logic, and can review and compile the code to complete the Java service. The Java developer can then use the Z Validation Assistant to generate JUnit tests for each Java method to check that the code is semantically equivalent to the COBOL service.



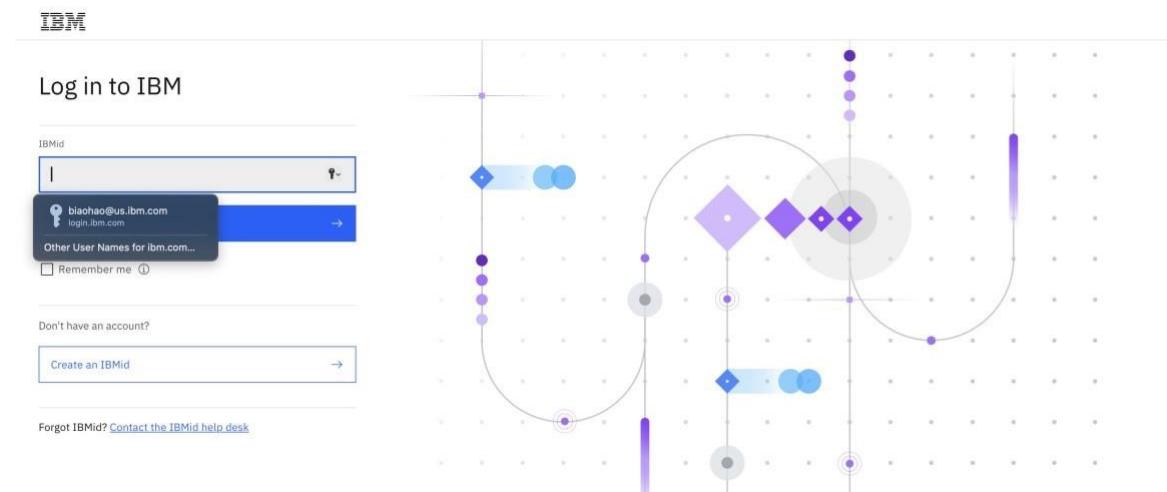
These images have been configured to communicate with each other, so you do not need to do that yourself. These configurations include:

- WCA4Z Service (1 of 3) – the LLM Service to connect to DB2 Meta Data for code transformation; the API Key for VS Code to access the LLM Service
- WCA4Z Discovery (2 of 3) – the network connection to WCA4Z Refactoring (3 of 3) via the private network interface; the SSH connection to WCA4Z Refactor (3 of 3) to start the Refactoring Assistant service; DB2 configuration to DB2 Meta Data on WCA4Z Service (1 of 3); and more

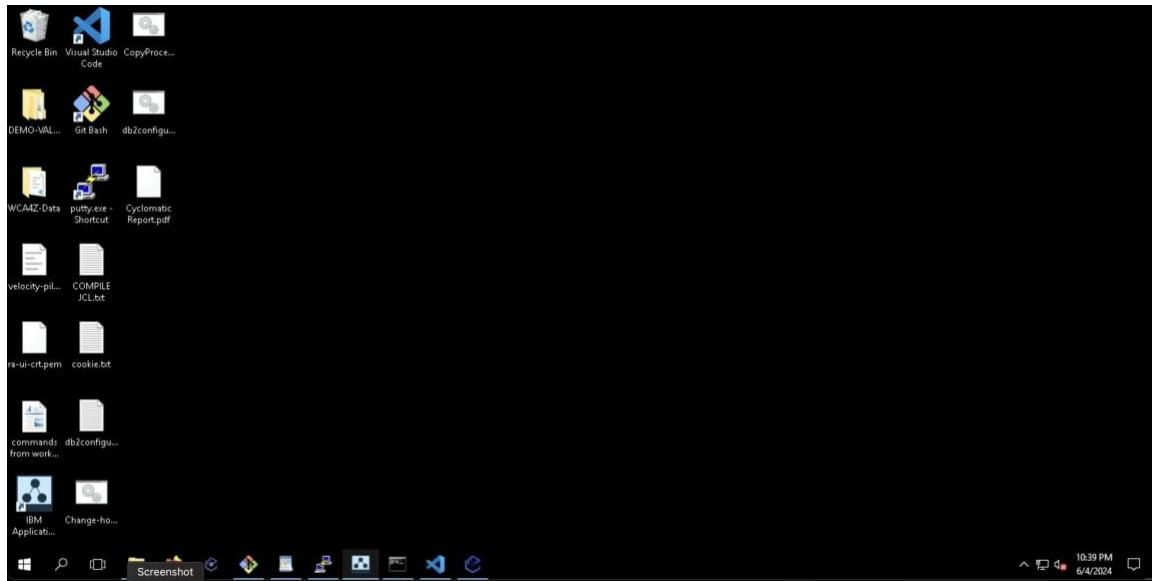
- WCA4Z Discovery (2 of 3) – the ADDI clientCOBOL project is built with either IBM sample code (or your own code reviewed by IBM); the ADDI refactoredCOBOL project is built and configured to use the DB2 for Meta Data
- WCA4Z Refactor (3 of 3) – the network connection to WCA4Z Discovery (2 of 3) via the private network interface
- WCA4Z Validation – Validate the outcome with auto-generated tests.
- VS Code – use the API key to invoke the LLM Service; and more

All the tasks will be performed for the training using the tools in the ADDI Server image.

You will log in using your IBM ID and the authentication credentials.



As indicated in the diagram below, you will be connected to the WCA4Z Discovery (2 of 3) or ADDI Server image. You will use the tools in the image, such as the ADDI Client and VS Code, to perform the tasks in the following Understand, Refactor, and Transform sections.



All the tasks will be performed for the training using the tools in the ADDI Server image.

## 2 01 Understand

---

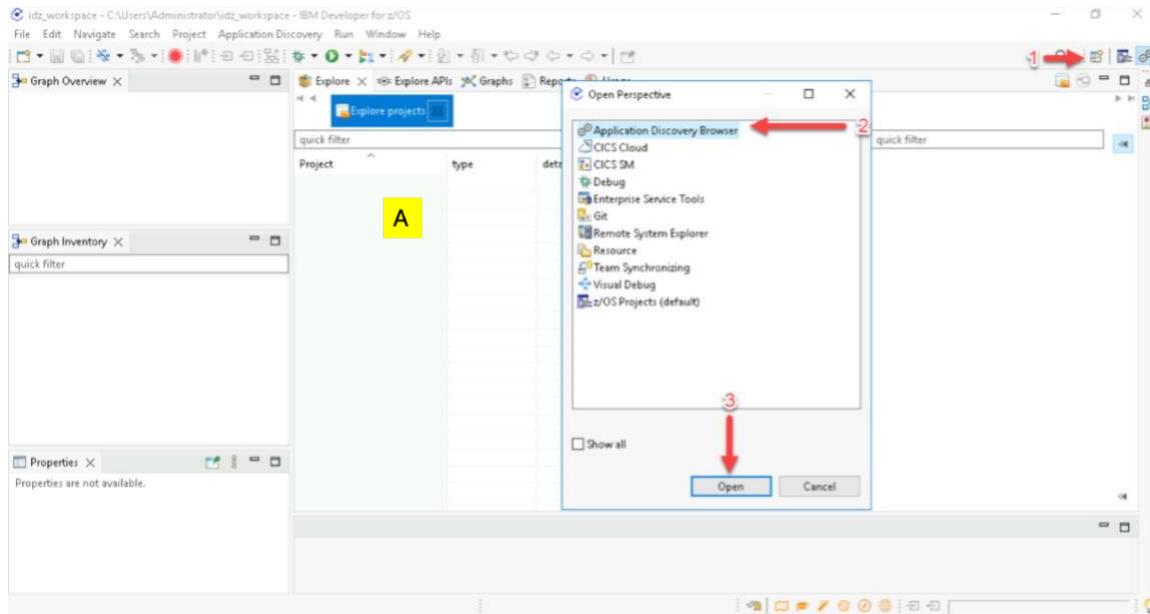
All the steps below are performed within the ADDI Server image.

### Step 1 – Open the ADDI Client

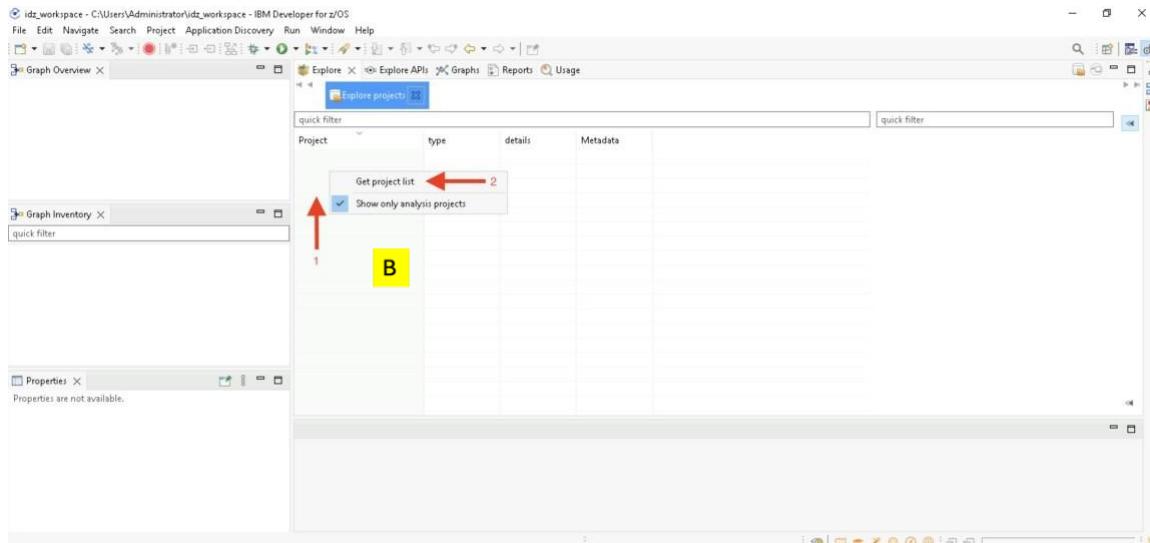
In your ADDI Server Image, open IBM Developer for z Systems.



Once opened, navigate to the Application Discovery Browser (A) perspective. Follow the instructions (1, 2, and 3, as indicated in the diagram below) to change to the Application Discovery Browser perspective if you are not in that perspective.

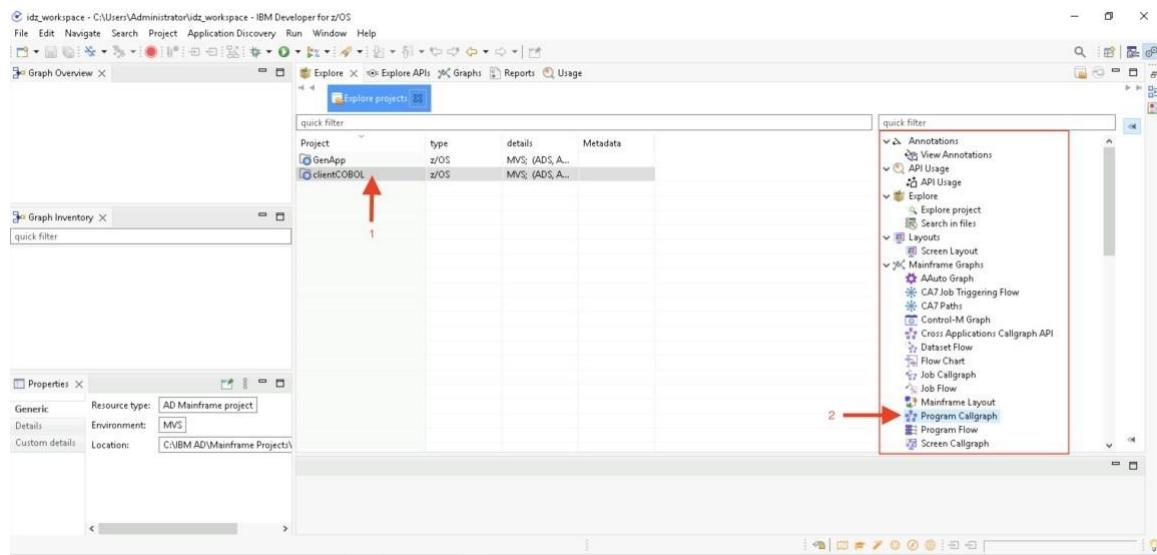


If no project is in the *Project* pane (B), right-click the empty space under *Project* to bring up the menu and select the menu item “*Get project list*”, as indicated in the diagram below.

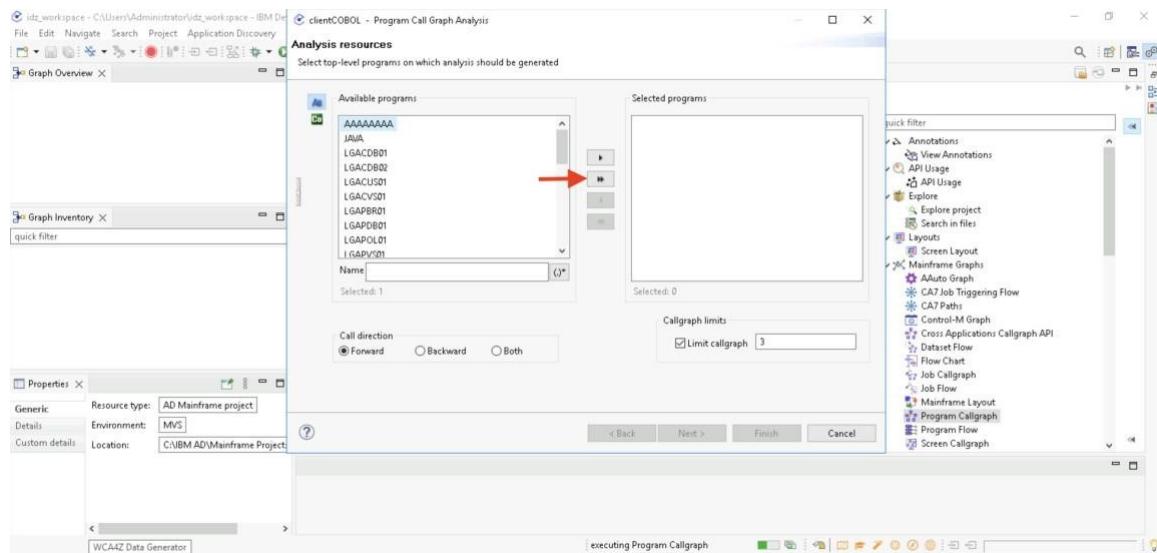


## 2.1 Step 2 – Build Program Callgraph

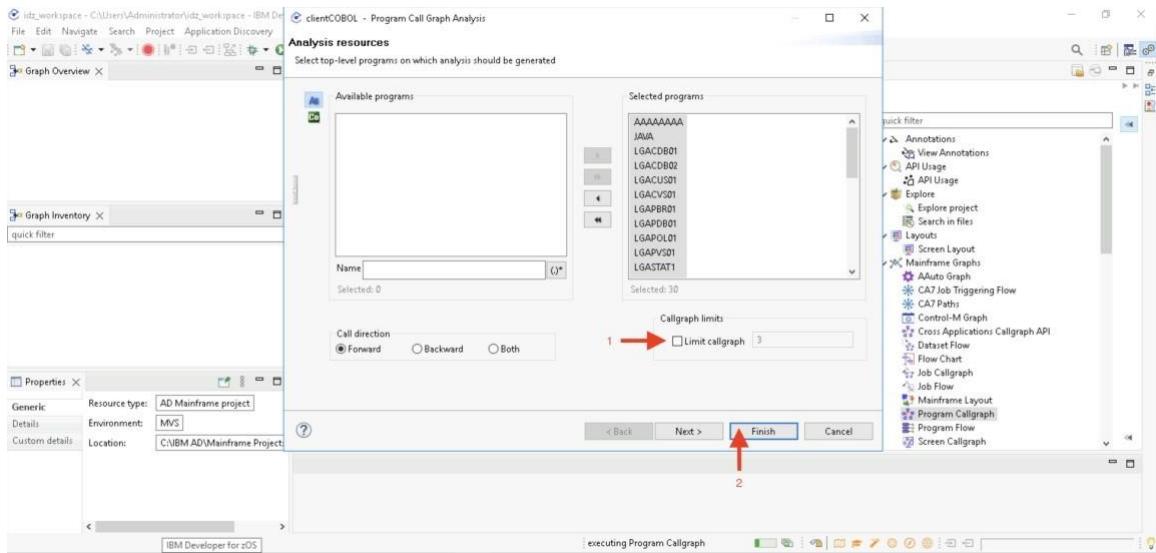
Select GenApp or *clientCOBOL* in the project list, the ADDI menu on the right side appears. Find and double-click “*Program Callgraph*” under “*Mainframe Graphs*”, as indicated in the diagram below.



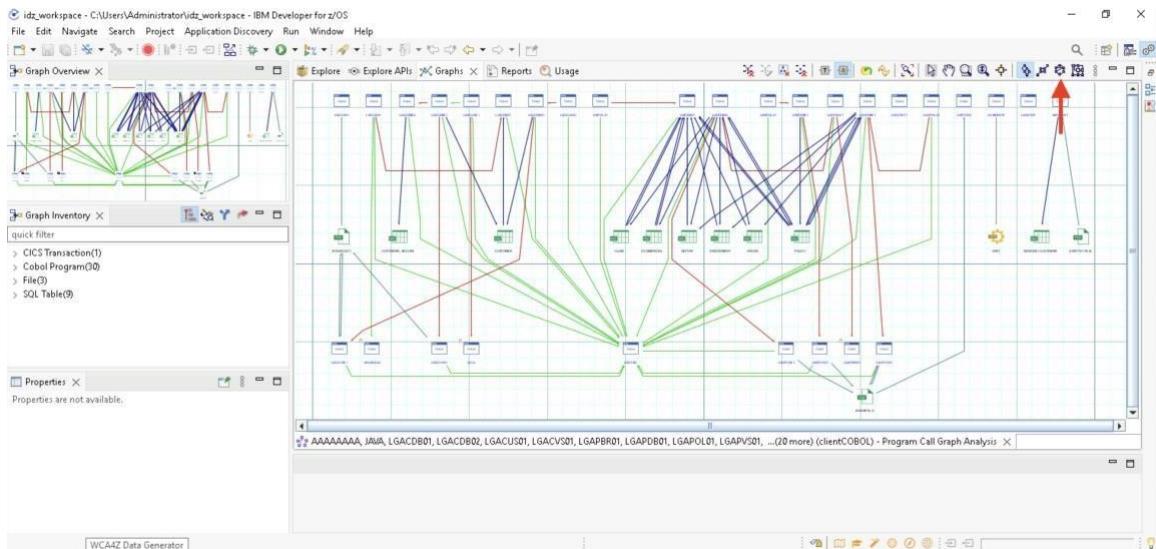
Click “>>” to select all in *Available programs*, as indicated in the diagram below.



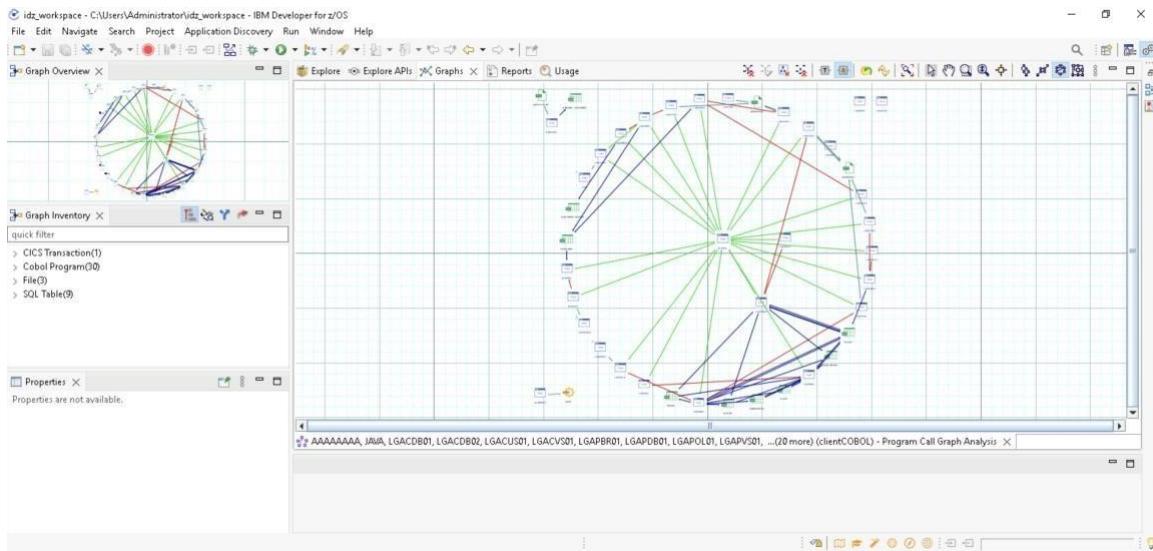
Uncheck the *Limit callgraph* and click *Finish*, as indicated in the diagram below.



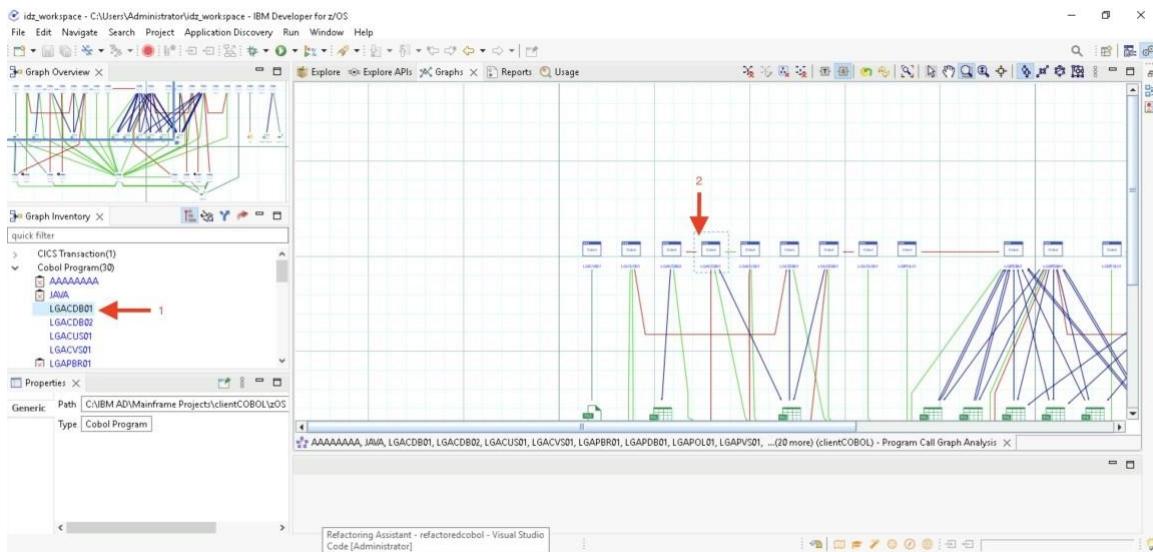
The diagram below is the program call graph. Click the icon “*Change layout to circular*”, as indicated in the diagram below.



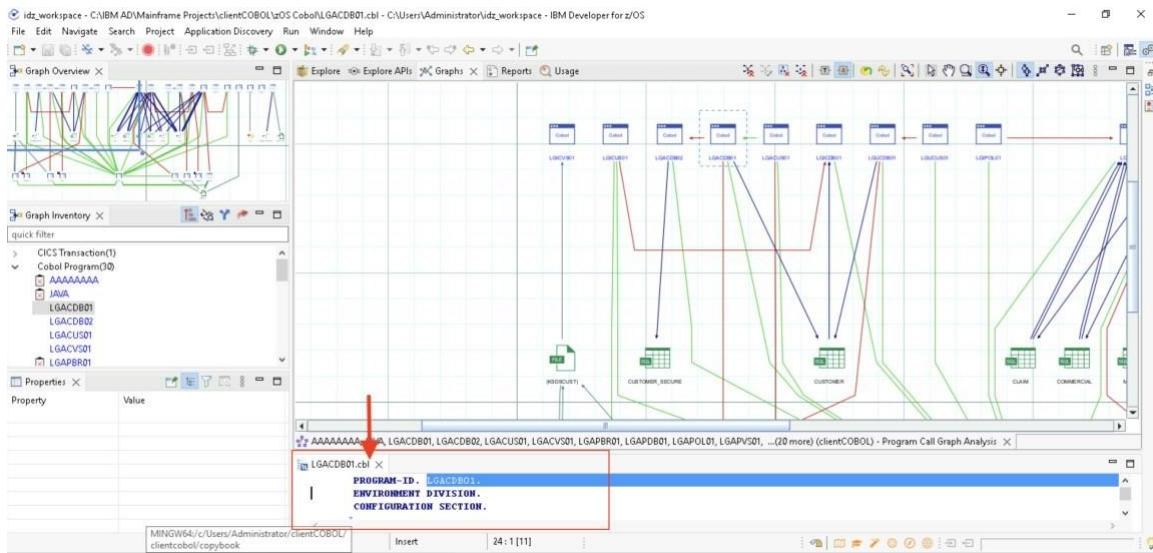
The diagram below is the circular program call graph. By clicking the different layout icons, you can change it to a different layout, such as *hierarchical*, *symmetric*, or *orthogonal*. Click the icon “*Change layout to hierarchical*” to revert back to the initial *hierarchical* layout.



Expand on the *Cobol Programs* in the *Graph Inventory* pane on the left side and select *LGACDB01*. The diagram below shows the selected program centered in the main graph pane. Double-click the program icon to show the COBOL code in the pane below.



The diagram below shows the COBOL code. Double-click on the program tab *LGACDB01.cbl* to maximize the program pane.



Review the COBOL code. Double-click on the program tab *LGACDB01.cbl* to restore the program pane.

The screenshot shows the IBM Application Discovery tool window. The code for LGACDB01.cbl is displayed in the main pane. An arrow points to the top of the code, specifically to the first line: `PROGRAM-ID. LGACDB01.`

```

PROGRAM-ID. LGACDB01.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
*
DATA DIVISION.

WORKING-STORAGE SECTION.

* Common definitions
* Run time (debug) information for this invocation
01 WS-HEADER.
 03 WS-EYECATCHER PIC X(16)
    VALUE 'LGACDB01-----BS'.
 03 WS-TRANSID PIC X(4).
 03 WS-TERMID PIC X(4).
 03 WS-TASNUM PIC 9(7).
 03 WS-FILLEN PIC X.
 03 WS-ADDR-DFHCOMMAREA USAGE IS POINTER.
 03 WS-CALEN PIC 99(4) COMP.

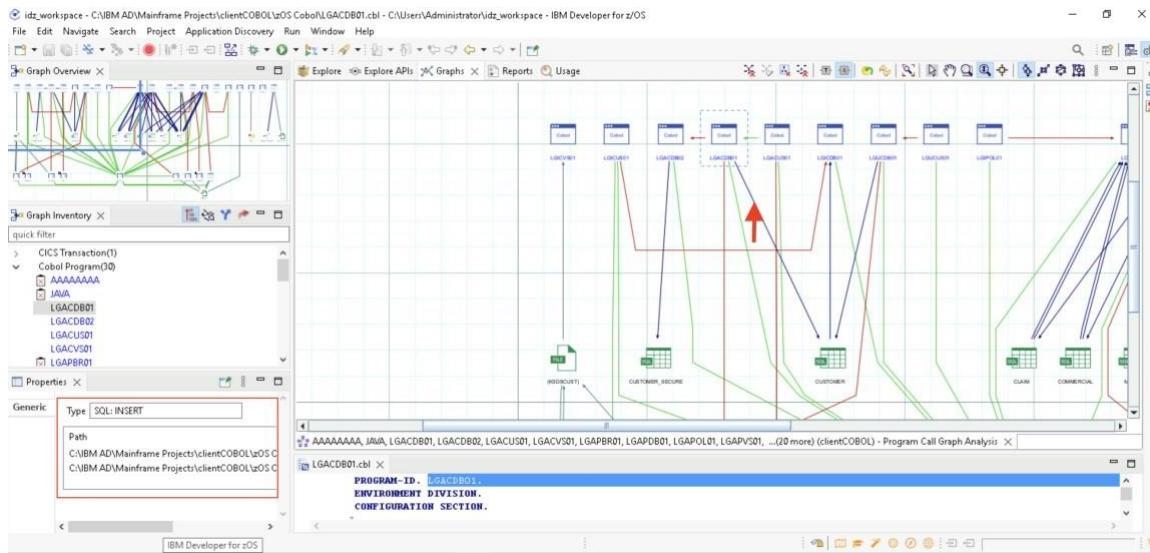
*
01 WS-RESP PIC 99(8) COMP.
01 LastCustNum PIC 99(8) COMP.
01 GENAccount PIC X(16) Value 'GEN&CUSTNUM'.
01 GENApool PIC X(8) Value 'GENA!'.

* Variables for time/date processing
01 WS-ABSTIME PIC 99(8) COMP VALUE +0.
01 WS-TIME PIC X(8) VALUE SPACES.
01 WS-DATE PIC X(10) VALUE SPACES.

* Error Message structure
01 ERROR-MSG.

```

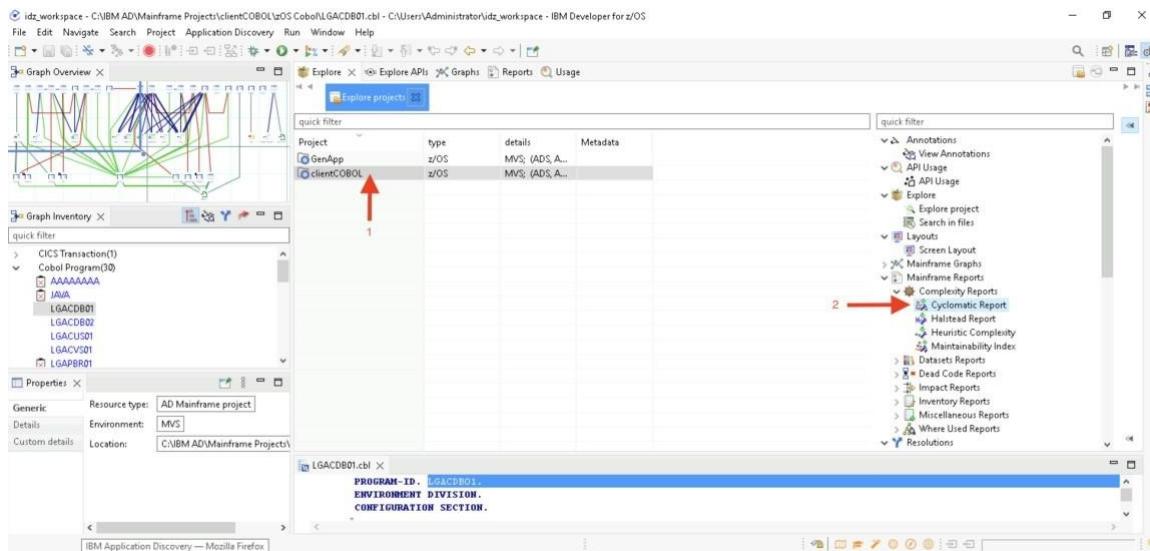
Click on the blue line between the program icon for *LGACDB01.cbl* and the table icon for *CUSTOMER* table; the Properties pane on the bottom left side shows it's an SQL INSERT statement, as indicated in the diagram below.



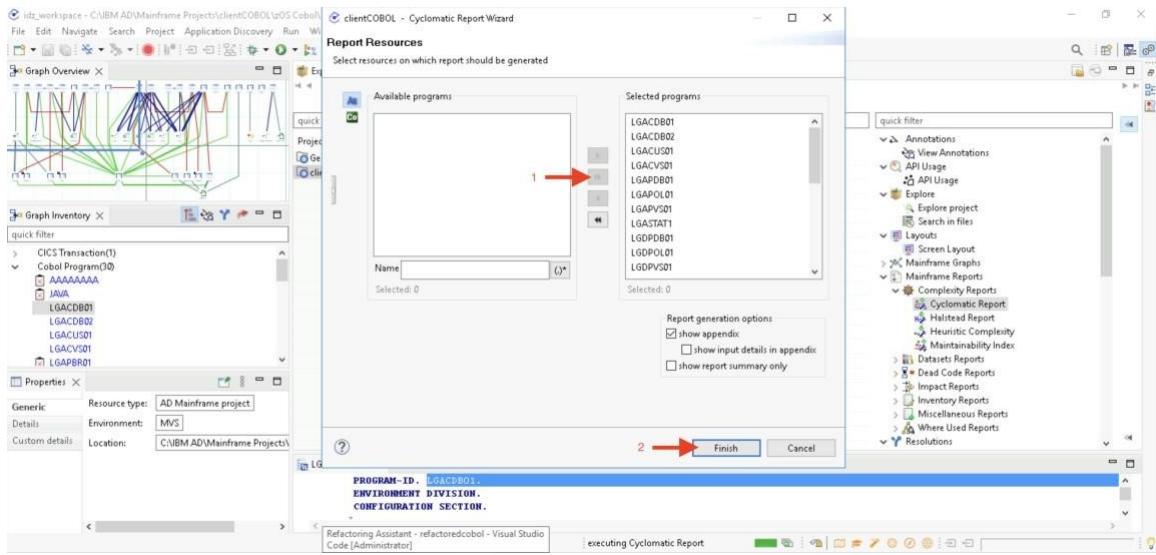
## 2.2 Step 3 – Create Mainframe Report

Back to *Explore projects*, select the *clientCOBOL* project, then expand on “Mainframe”

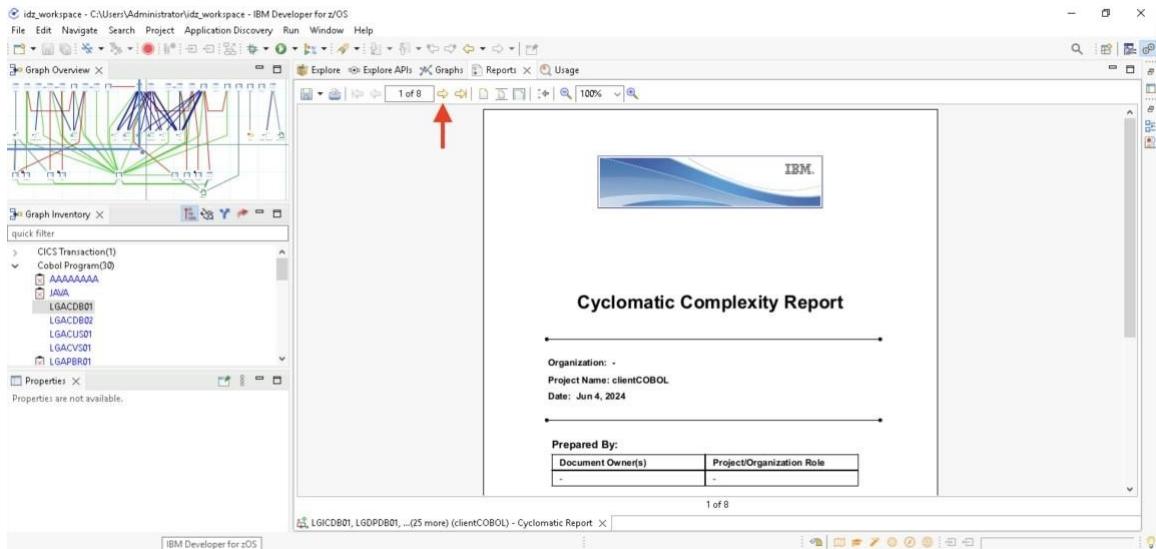
*Reports > Complexity Reports*" on the right side, and double-click the "*Cyclomatic Report*", as indicated in the diagram below.



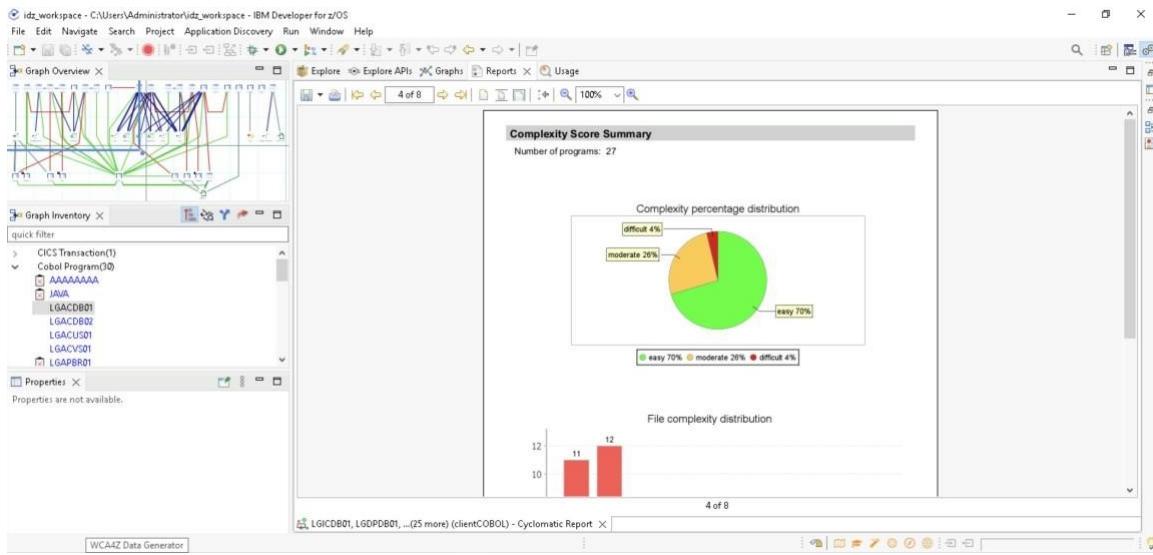
Click “>>” to select all of the *Available programs* and then click *Finish*.



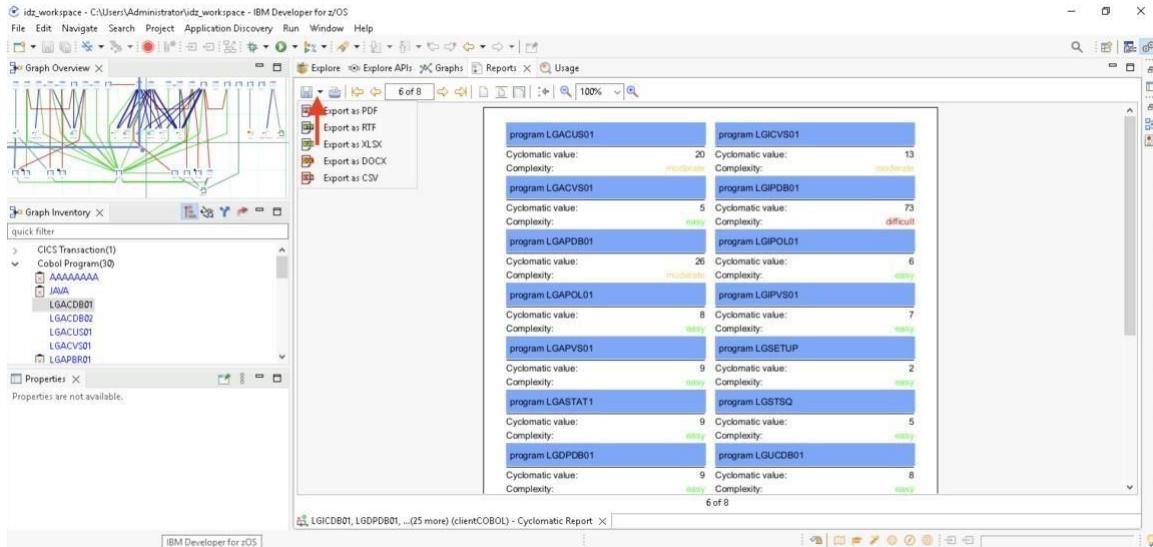
The Cyclomatic Complexity Report is shown in the diagram below. You can review it by clicking the page navigation icons, as indicated in the diagram below.



The diagram below shows another page of the report.



The diagram below shows another page of the report. You can export the report in various formats, as indicated in the diagram below.



You can generate other types of reports that are of interest to you.

## 2.3 Step 4 – More About ADDI

Program Callgraph and *Cyclomatic Report* above are just two examples of many ADDI functionalities. To learn more about ADDI, look at the ADDI documentation here:

<https://www.ibm.com/docs/en/addi/6.1.2?topic=analyze-ui-reference>

## END OF UNDERSTAND SECTION

## 3 02 Refactor

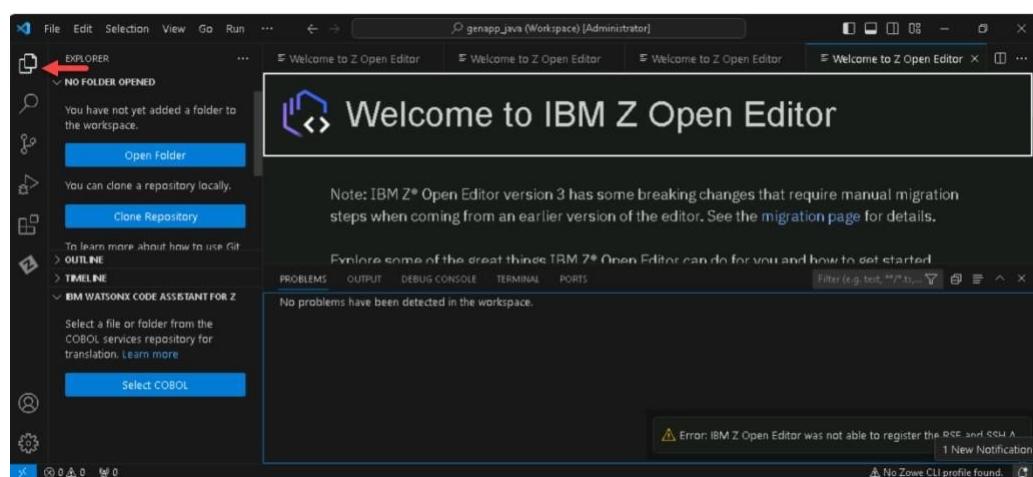
---

All the steps below are performed within the ADDI Server image.

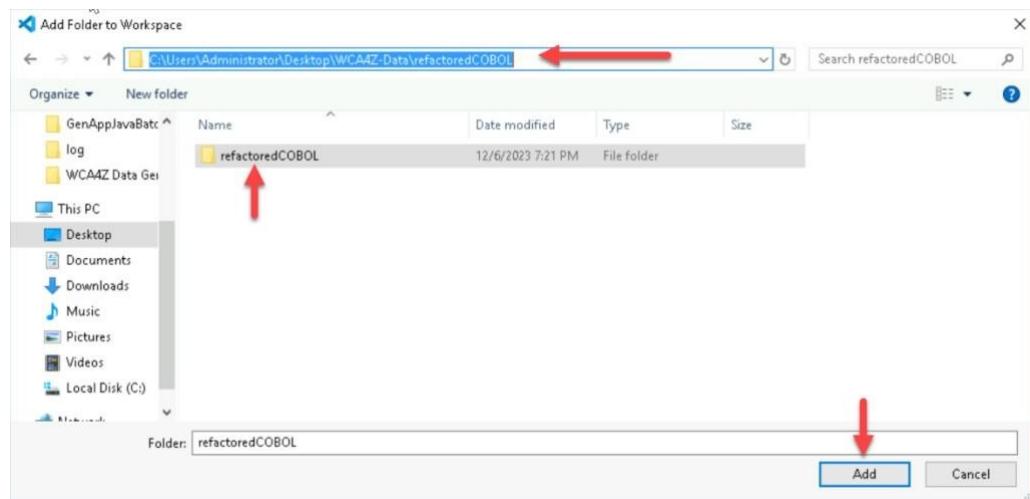
### 3.1 Step 1 – Select Folder for Refactored Code

You will now connect your refactored COBOL repo to VS Code, where you will later store your refactored COBOL code, ready for transformation to Java.

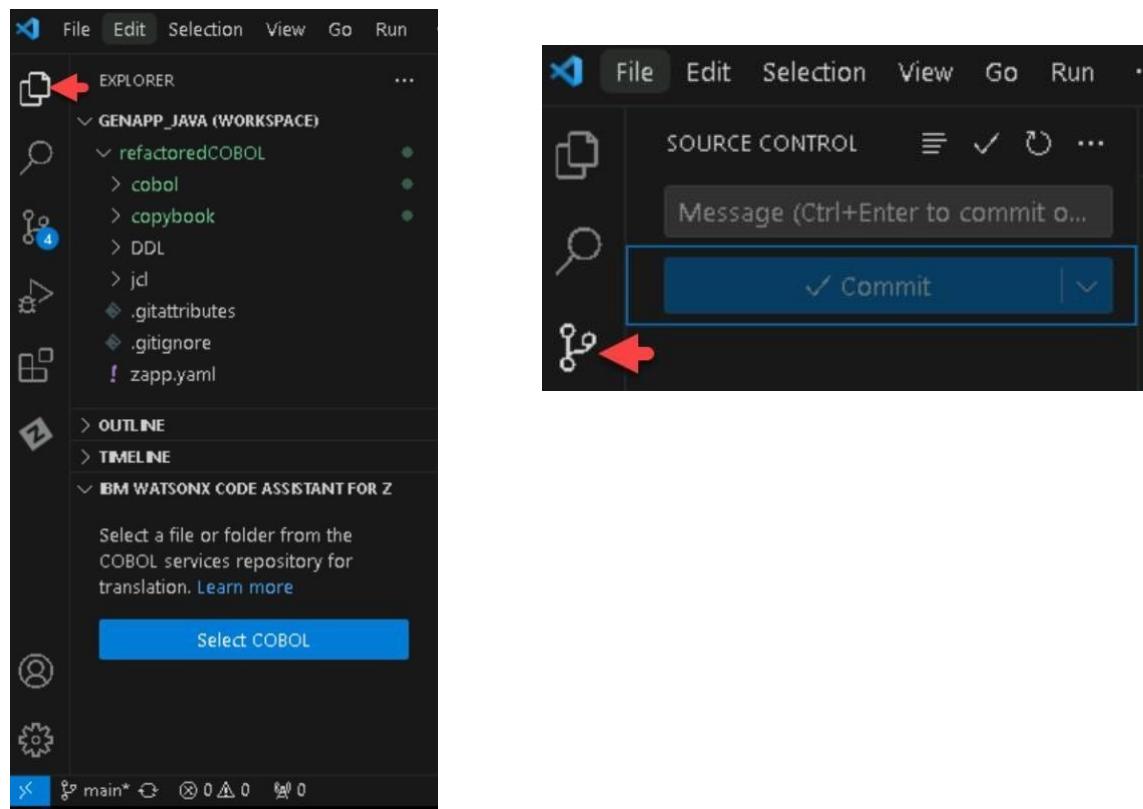
Navigate to the *Explorer* tab.



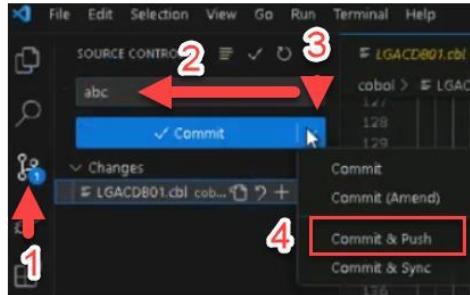
Select “Open Folder” and navigate to  
C:\Users\Administrator\Desktop\WCA4ZData\refactoredCOBOL, select the folder *refactoredCOBOL*, and then *Add*.



Confirm that you can see this in the *Explorer* tab and the *Git* tab in your VS Code environment.

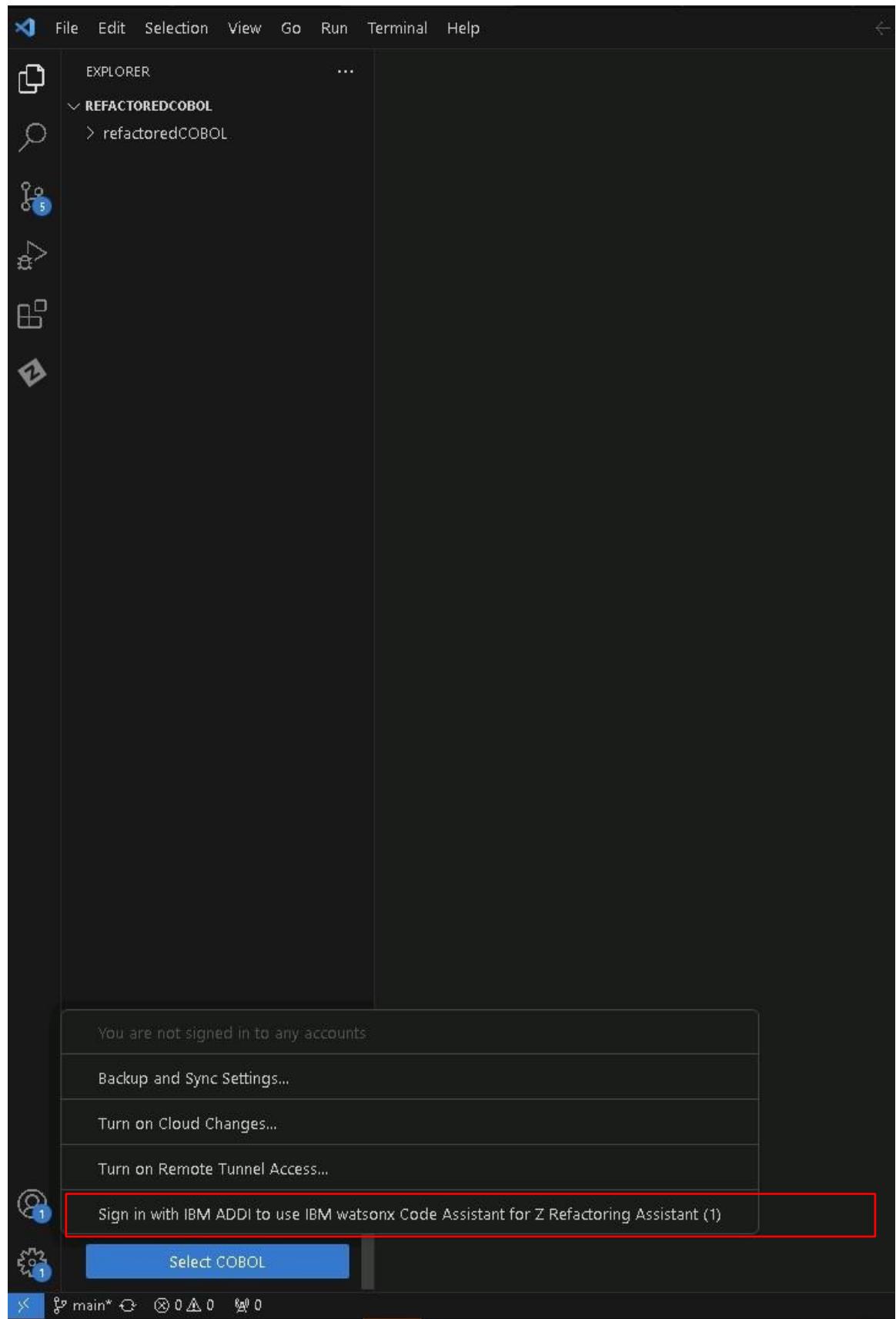


You may see changes that need to be staged and pushed to your Git repo. If so, add a commit message and then select “Commit & Push” to push your copybooks to your *refactoredCOBOL* Git repo.

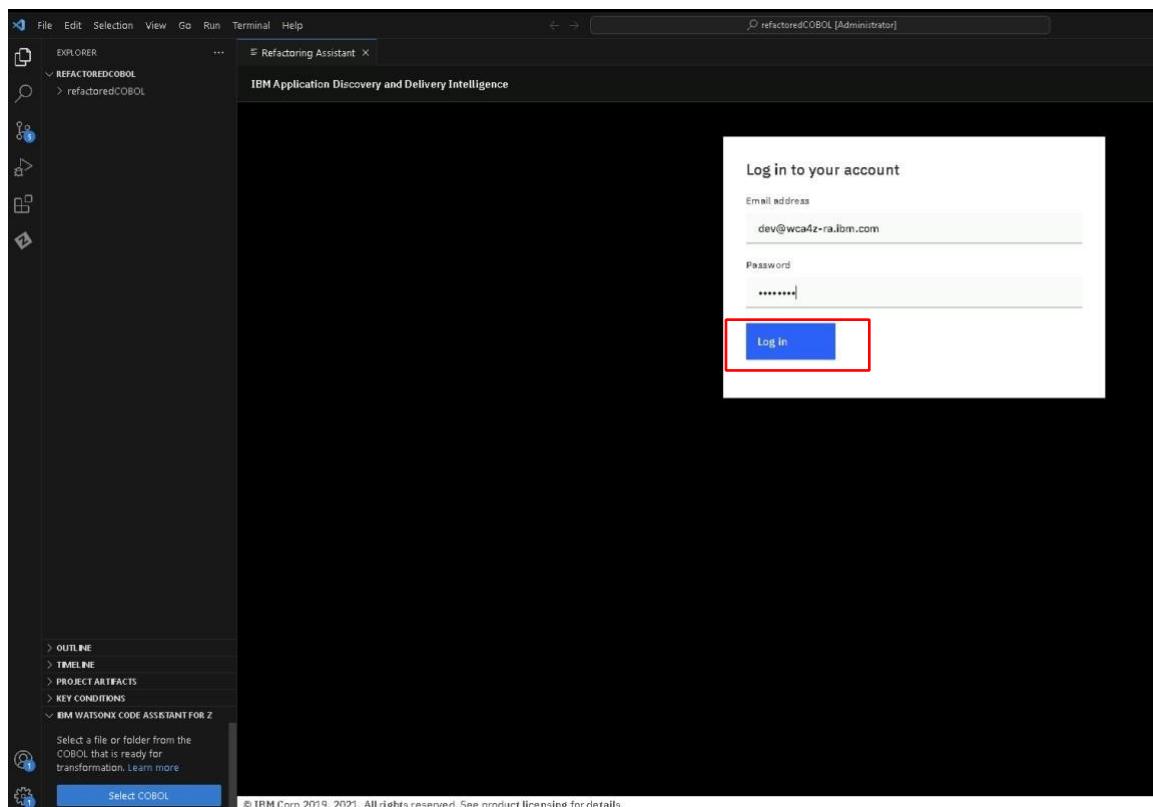


### 3.2 Step 2 – Sign in to Use Refactoring Assistant

You will use the new Refactoring Assistant interface via VS Code instead of the web browser. Launch VS Code and select “Accounts > Sign in with IBM ADDI to use IBM Watsonx Code Assistant for Z Refactoring Assistant”, as indicated in the diagram below.



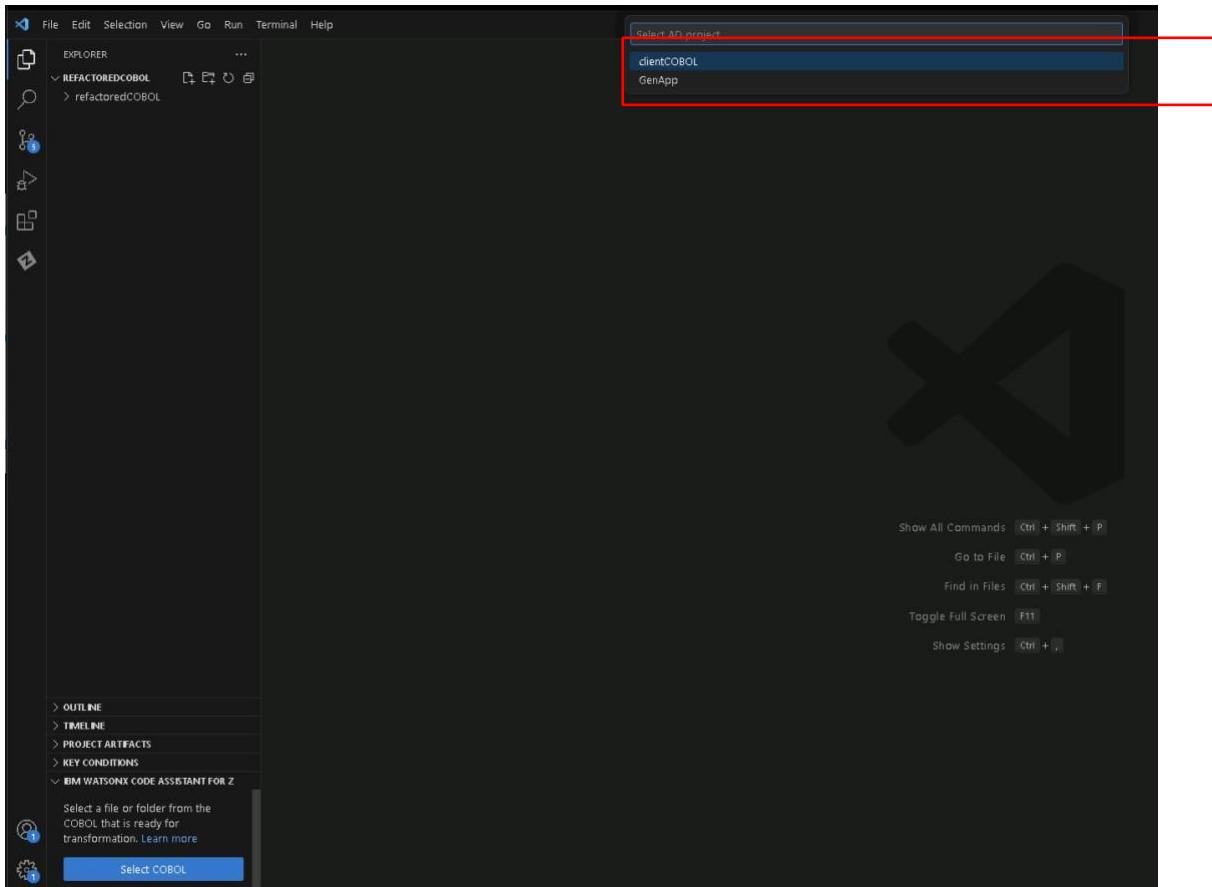
A new Refactoring Assistant window will open. Type in the credentials: [dev@wca4zra.ibm.com](mailto:dev@wca4zra.ibm.com) for *userid* and password for *password*. Click “*Log In*”.



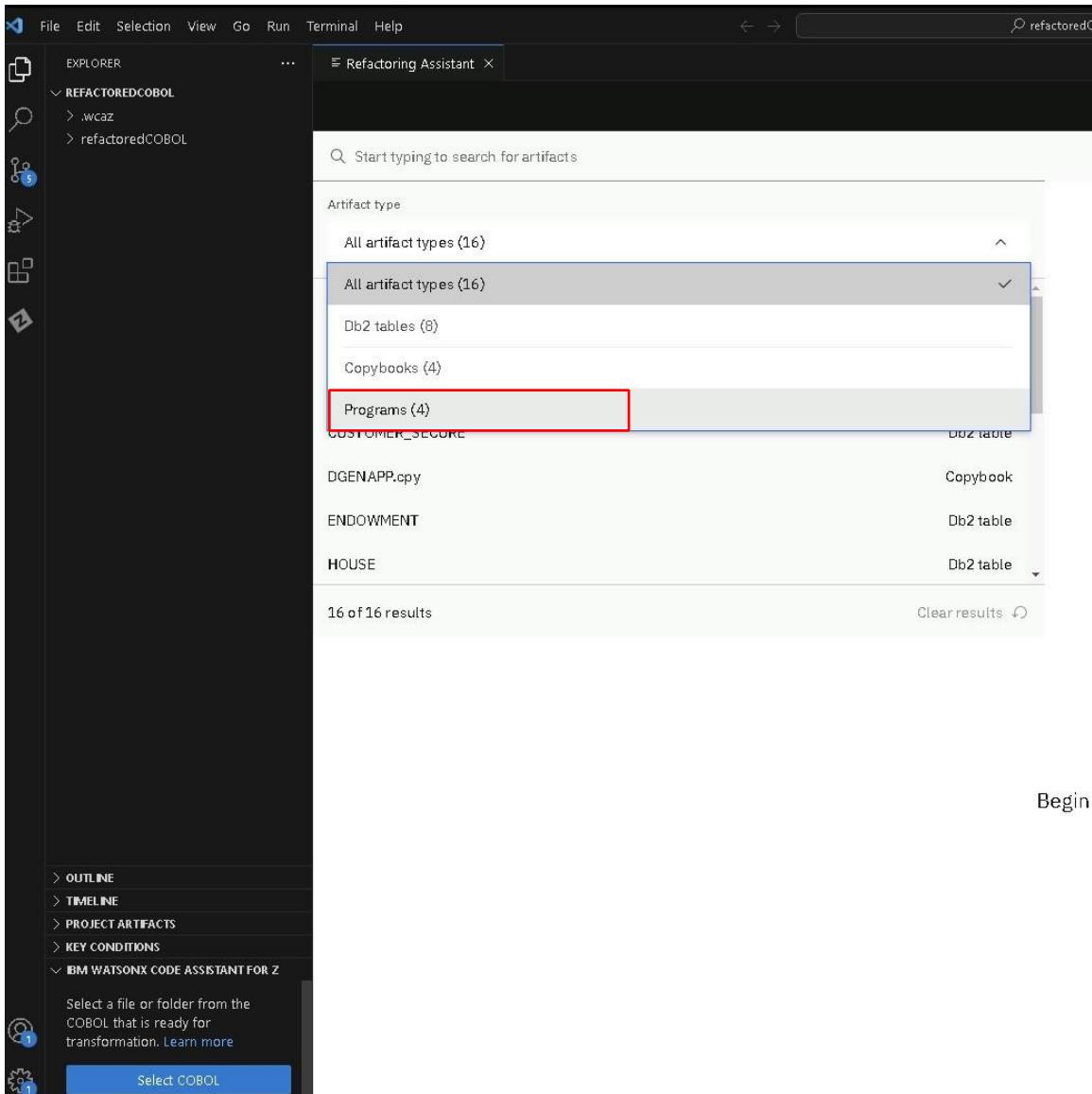
### 3.3 Step 3 – Refactor COBOL Code

At the top of a window you will see a list of ADDI projects, including GenApp and *clientCOBOL*.

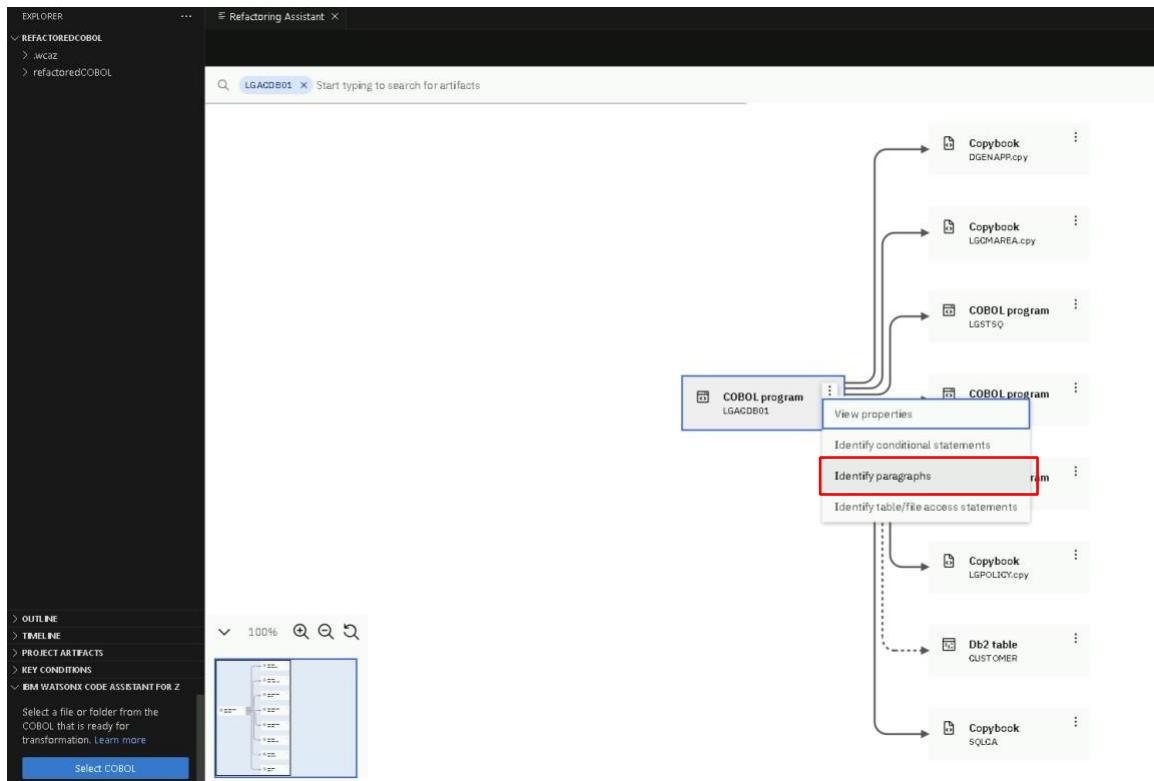
Select *GenApp* to refactor the COBOL code.



Select “*Programs*” as the artifact type and then the name of the program that you will refactor. In this guide, you select the *LGACDB01* program. The program inserts a customer into the *CUSTOMER* table, among other things.



Click on the three dots next to the program that you want to refactor and select “*Identify Paragraphs*”.



Identified paragraphs will appear in the menu on the left under “Key Conditions”. Select the paragraph *INSERT-CUSTOMER* that you would like to work on. The selected paragraph will appear on the right side.

### 3.4

The screenshot shows the IBM Watson Studio IDE interface. The main window displays a COBOL program named `LGACDB01.cbl`. The code includes several paragraphs: `MAINLINE SECTION.`, `INSERT-CUSTOMER.`, `WRITE-ERROR-MESSAGE.`, `Obtain-CUSTOMER-Number.`, and `MAINLINE-EXIT.`. The `INSERT-CUSTOMER.` paragraph is highlighted with a red box. In the bottom-left corner, there is a section titled "KEY CONDITIONS" with a "Filter: Paragraphs" dropdown. Below it, a list of conditions is shown, with the first item, `MAINLINE SECTION. (Line 145)`, also highlighted with a red box. On the left side of the interface, there is a sidebar with various icons and sections like "EXPLORER", "OUTLINE", "TIMELINE", and "PROJECT ARTIFACTS". The "PROJECT ARTIFACTS" section shows a file named `LGACDB01`. At the bottom, there is an "IBM WATSONX CODE ASSISTANT FOR Z" section with a "Select COBOL" button.

Right-click the paragraph named `INSERT-CUSTOMER` and select menu item “Slice on condition to new service”.

The screenshot shows the IBM Watson Code Assistant for Z/OS interface. The main window displays a COBOL program named `LGACDB01.cbl`. The code is as follows:

```
236      INSERT-CUSTOMER.  
237      *-----  
238      * Insert row into Customer table based on customer number  
239      *-----  
240      MOVE 'INSERT-CUSTOMER' TO EM-SQLREQ  
241      *-----  
242      IF LGAC-NCS = 'ON'  
243          EXEC SQL  
244              INSERT INTO CUSTOMER  
245                  ( CUSTOMERNUMBER,  
246                      FIRSTNAME,  
247                      LASTNAME,  
248                      DATEOFBIRTH,  
249                      HOUSENAME,  
250                      HOUSENUMBER,  
251                      POSTCODE,  
252                      PHONEMOBILE,  
253                      PHONEHOME,  
254                      EMAILADDRESS )  
255              VALUES ( :DB2-CUSTOMERNUM-INT,  
256                      :CA-FIRST-NAME,  
257                      :CA-LAST-NAME,  
258                      :CA-DOB,  
259                      :CA-HOUSE-NAME,  
260                      :CA-HOUSE-NUM,  
261                      :CA-POSTCODE,  
262                      :CA-PHONE-MOBILE,  
263                      :CA-PHONE-HOME,  
264                      :CA-EMAIL-ADDRESS )  
265          END-EXEC  
266          IF SQLCODE NOT EQUAL 0  
267              MOVE '00' TO CA-RETURN-CODE  
268              PERFORM WRITE-ERROR-MESSAGE  
269              EXEC CICS RETURN END-EXEC  
270          END-IF  
271      ELSE  
272          EXEC SQL  
273              INSERT INTO CUSTOMER  
274                  ( CUSTOMERNUMBER,  
275                      FIRSTNAME,  
276                      LASTNAME,  
277                      DATEOFBIRTH,  
278                      HOUSENAME,  
279                      HOUSENUMBER,  
280                      POSTCODE,  
281                      PHONEMOBILE,  
282                      PHONEHOME,  
283                      EMAILADDRESS )  
284              VALUES ( DEFAULT,  
285                      :CA-FIRST-NAME,  
286                      :CA-LAST-NAME,  
287                      :CA-DOB )
```

The code editor highlights several sections with red boxes and dropdown menus:

- A red box surrounds the `MAINLINE SECTION.` section, with a tooltip "Slice on condition to new service".
- A red box surrounds the `INSERT-CUSTOMER.` section, with a tooltip "Slice on condition to existing service".

The left sidebar includes sections for `EXPLORER`, `REFACTOREDCOBOL`, `OUTLINE`, `TIMELINE`, `PROJECT ARTIFACTS` (with `LGACDB01` selected), and `KEY CONDITIONS`. The `KEY CONDITIONS` section lists:

- MAINLINE SECTION. (Line 145)
- INSERT-CUSTOMER. (Line 274)
- WRITE-ERROR-MESSAGE. (Line 268)
- Obtain-CUSTOMER-Number. (Line 279)
- MAINLINE-EXIT. (Line 215)

The bottom left corner features the **IBM WATSON CODE ASSISTANT FOR Z** logo and a button labeled **Select COBOL**.

You will be prompted to type the name for a new service. Give it the name `LGACDB01`, and press the *Enter* key, as indicated in the diagram below.

The screenshot shows the IBM Watsonx Code Assistant for Z interface. On the left, the Explorer pane displays project artifacts, including a new folder named 'LGACDB01'. The main editor window shows COBOL code for inserting a customer into a database. A red box highlights the input field 'Enter service name (Press 'Enter' to confirm or 'Escape' to cancel)' which contains 'LGACDB01'. The code includes sections for inserting into the CUSTOMER table and handling SQL errors.

```
236      *-----*
237      * Insert row into Customer table based on customer number *
238      *-----*
239      MOVE ' INSERT-CUSTOMER' TO EM-SQLREQ
240
241      IF LGAC-NCS = 'ON'
242      EXEC-SQL
243      INSERT INTO CUSTOMER
244      ( CUSTOMERNUMBER,
245      FIRSTNAME,
246      LASTNAME,
247      DATEOFBIRTH,
248      HOUSENAME,
249      HOUSENUMBER,
250      POSTCODE,
251      PHONEMOBILE,
252      PHONEHOME,
253      EMAILADDRESS )
254      VALUES ( :DB2-CUSTOMERNUM-INT,
255      :CA-FIRST-NAME,
256      :CA-LAST-NAME,
257      :CA-DOB,
258      :CA-HOUSE-NAME,
259      :CA-HOUSE-NUM,
260      :CA-POSTCODE,
261      :CA-PHONE-MOBILE,
262      :CA-PHONE-HOME,
263      :CA-EMAIL-ADDRESS )
264
265      END-EXEC
266      IF SQLCODE NOT EQUAL 0
267      MOVE '08' TO CA-RETURN-CODE
268      PERFORM WRITE-ERROR-MESSAGE
269      EXEC CICS RETURN END-EXEC
270
271      END-IF
272      ELSE
273      EXEC-SQL
274      INSERT INTO CUSTOMER
275      ( CUSTOMERNUMBER,
276      FIRSTNAME,
277      LASTNAME,
278      DATEOFBIRTH,
279      HOUSENAME,
280      HOUSENUMBER,
281      POSTCODE,
282      PHONEMOBILE,
283      PHONEHOME,
284      EMAILADDRESS )
285      VALUES ( DEFAULT,
286      :CA-FIRST-NAME,
287      :CA-LAST-NAME,
      :CA-HOME )
```

In the file explorer on the left side, you will see a new subfolder for a new COBOL service, *LGACDB01.cbls*. The name matches the service name you gave in the previous step.

The screenshot shows the Rational Developer for COBOL interface with two main panes. The left pane displays the project structure under 'REFACTOREDCOBOL' with files like 'LGACDB01' and 'refactoredCOBOL'. A red box highlights the 'LGACDB01.cbls' file. The right pane shows the 'Refactoring Assistant' tool with a code editor containing COBOL and SQL code. The code is for inserting customer data into a database. The right pane also shows a preview of the generated service code, which is a Java class named 'INSERT-CUSTOMER' with methods for inserting customers into a database.

```

File Edit Selection View Go Run Terminal Help
REFACTOREDCOBOL
  > .waz
  > LGACDB01
  > LGACDB01.cbls
  > refactoredCOBOL

EXPLORER
LGACDB01.cbls

REFACTORING ASSISTANT
LGACDB01.cbls <--> LGACDB01.cbls

MOVE * INSERT CUSTOMER* TO EM-SQREQ
  IF LGAC-NCS = 'ON'
    EXEC SQL
      INSERT INTO CUSTOMER
        ( CUSTOMERNUMBER,
          FIRSTNAME,
          LASTNAME,
          DATEOFBIRTH,
          HOUSENAME,
          HOUSENUMBER,
          POSTCODE,
          PHONEMOBILE,
          PHONEHOME,
          EMAILADDRESS )
      VALUES ( :DB2-CUSTOMERNUM-INT,
                :CA-FIRST-NAME,
                :CA-LAST-NAME,
                :CA-DOB,
                :CA-HOUSE-NAME,
                :CA-HOUSE-NUM,
                :CA-POSTCODE,
                :CA-PHONE-MOBILE,
                :CA-PHONE-HOME,
                :CA-EMAIL-ADDRESS )
    END-EXEC
    IF SQLCODE NOT EQUAL 0
      MOVE '99' TO CA-RETURN-CODE
      PERFORM WRITE-ERROR-MESSAGE
      EXEC CICS RETURN END-EXEC
    END-IF
  ELSE
    EXEC SQL
      INSERT INTO CUSTOMER
        ( CUSTOMERNUMBER,
          FIRSTNAME,
          LASTNAME,
          DATEOFBIRTH,
          HOUSENAME,
          HOUSENUMBER,
          POSTCODE,
          PHONEMOBILE,
          PHONEHOME,
          EMAILADDRESS )
      VALUES ( :DEFAULT,
                :CA-FIRST-NAME,
                :CA-LAST-NAME,
                :CA-DOB,
                :CA-HOUSE-NAME,
                :CA-HOUSE-NUM,
                :CA-POSTCODE,
                :CA-PHONE-MOBILE,
                :CA-PHONE-HOME,
                :CA-EMAIL-ADDRESS )
    END-EXEC
    IF SQLCODE NOT EQUAL 0
      MOVE '90' TO CA-RETURN-CODE
      PERFORM WRITE-ERROR-MESSAGE
      EXEC CICS RETURN END-EXEC
    END-IF
  ELSE
    EXEC SQL
      INSERT INTO CUSTOMER
        ( CUSTOMERNUMBER,
          FIRSTNAME,
          LASTNAME,
          DATEOFBIRTH,
          HOUSENAME,
          HOUSENUMBER,
          POSTCODE,
          PHONEMOBILE,
          PHONEHOME,
          EMAILADDRESS )
      VALUES ( :DEFAULT,
                :CA-FIRST-NAME,
                :CA-LAST-NAME,
                :CA-DOB,
                :CA-HOUSE-NAME,
                :CA-HOUSE-NUM,
                :CA-POSTCODE,
                :CA-PHONE-MOBILE,
                :CA-PHONE-HOME,
                :CA-EMAIL-ADDRESS )
    END-EXEC
    IF SQLCODE NOT EQUAL 0
      MOVE '90' TO CA-RETURN-CODE
      PERFORM WRITE-ERROR-MESSAGE
      EXEC CICS RETURN END-EXEC
    END-IF
  END-IF
END-EXEC
  
```

Select the *LGACDB01.cbls* file on the left, right click and select “*Refactoring Assistant > Generate service code*”.

The screenshot shows the IBM Watson Studio interface with the Refactoring Assistant tool open. The 'Generate Service Code' button is highlighted with a red box. The code editor displays COBOL and Java code blocks.

```

REFACTOREDCOBOL
LGACDB01
LGACDB01
Code block from LGACDB01 (lines 276-308) graph
TOMER.
    * into Customer table based on customer number
    * INSERT CUSTOMER TO EM-SQLEQ
    IF LOAC-NCS = 'ON'
        EXEC SQL
        INSERT INTO CUSTOMER
        ( CUSTOMERNUMBER,
        FIRSTNAME,
        LASTNAME,
        DATEOFBIRTH,
        HOUSENAME,
        HOUSENUMBER,
        POSTCODE,
        PHONEMOBILE,
        PHONEHOME,
        EMAILADDRESS )
        VALUES ( :IB2-CUSTOMERNUM-INT,
        :CA-FIRST-NAME,
        :CA-LAST-NAME,
        :CA-DOB,
        :CA-HOUSE-NAME,
        :CA-HOUSE-NUM,
        :CA-POSTCODE,
        :CA-PHONE-MOBILE,
        :CA-PHONE-HOME,
        :CA-EMAIL-ADDRESS )
    END-EXEC
    IF SQLCODE NOT EQUAL 0
        MOVE '00' TO CA-RETURN-CODE
        PERFORM WRITE-ERROR-MESSAGE
        EXEC CICS RETURN END-EXEC
    END-IF
    ELSE
        EXEC SQL
        INSERT INTO CUSTOMER
        ( CUSTOMERNUMBER,
        FIRSTNAME,
        LASTNAME,
        DATEOFBIRTH,
        HOUSENAME,
        HOUSENUMBER,
        POSTCODE,
        PHONEMOBILE,
        PHONEHOME,
        EMAILADDRESS )
        VALUES ( DEFAULT,
        :CA-FIRST-NAME,
        :CA-LAST-NAME,
        :CA-DOB,
        :CA-HOUSE-NAME,
        :CA-HOUSE-NUM,
        :CA-POSTCODE,
        :CA-PHONE-MOBILE,
        :CA-PHONE-HOME,
        :CA-EMAIL-ADDRESS )
    END-EXEC
    IF SQLCODE NOT EQUAL 0
        MOVE '00' TO CA-RETURN-CODE
        PERFORM WRITE-ERROR-MESSAGE
        EXEC CICS RETURN END-EXEC
    END-IF
    ELSE
        EXEC SQL
        INSERT INTO CUSTOMER
        ( CUSTOMERNUMBER,
        FIRSTNAME,
        LASTNAME,
        DATEOFBIRTH,
        HOUSENAME,
        HOUSENUMBER,
        POSTCODE,
        PHONEMOBILE,
        PHONEHOME,
        EMAILADDRESS )
        VALUES ( DEFAULT,
        :CA-FIRST-NAME,
        :CA-LAST-NAME,
        :CA-DOB,
        :CA-HOUSE-NAME,
        :CA-HOUSE-NUM,
        :CA-POSTCODE,
        :CA-PHONE-MOBILE,
        :CA-PHONE-HOME,
        :CA-EMAIL-ADDRESS )
    END-EXEC
    IF SQLCODE NOT EQUAL 0
        MOVE '00' TO CA-RETURN-CODE
        PERFORM WRITE-ERROR-MESSAGE
        EXEC CICS RETURN END-EXEC
    END-IF
END-IF

```

You will be prompted for a service name. Enter *RAEXPORT* and then press the *Enter* key.

The screenshot shows the IBM WatsonX Code Assistant for Z interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar shows 'RAEXPORT' with a red box around it, and 'Enter service program name (Press 'Enter' to confirm or 'Escape' to cancel)'. The left sidebar has sections for Explorer, Outline, Timeline, Project Artifacts, Key Conditions, and IBM WatsonX Code Assistant for Z. Under Project Artifacts, 'LGACDB01' is selected. The main workspace displays COBOL code for an 'INSERT-CUSTOMER' block. The code includes SQL INSERT statements and logic for handling different SQL codes. A tooltip in the bottom right corner of the workspace says 'Select COBOL'.

```
INSERT-CUSTOMER.  
* Insert row into Customer table based on customer number  
*  
MOVE ! INSERT CUSTOMER! TO EM-SQLREQ  
*  
IF LGAC-NCS = 'ON'  
  EXEC SQL  
    INSERT INTO CUSTOMER  
    ( CUSTOMERNUMBER,  
      FIRSTNAME,  
      LASTNAME,  
      DATEOFBIRTH,  
      HOUSENAME,  
      HOUSENUMBER,  
      POSTCODE,  
      PHONEMOBILE,  
      PHONEHOME,  
      EMAILADDRESS )  
    VALUES ( :B2-CUSTOMERNUM-INT,  
             :CA-FIRST-NAME,  
             :CA-LAST-NAME,  
             :CA-DOB,  
             :CA-HOUSE-NAME,  
             :CA-HOUSE-NUM,  
             :CA-POSTCODE,  
             :CA-PHONE-MOBILE,  
             :CA-PHONE-HOME,  
             :CA-EMAIL-ADDRESS )  
  END-EXEC  
  IF SQLCODE NOT EQUAL 0  
    MOVE '90' TO CA-RETURN-CODE  
    PERFORM WRITE-ERROR-MESSAGE  
    EXEC CICS RETURN END-EXEC  
  END-IF  
ELSE  
  EXEC SQL  
    INSERT INTO CUSTOMER  
    ( CUSTOMERNUMBER,  
      FIRSTNAME,  
      LASTNAME,  
      DATEOFBIRTH,  
      HOUSENAME,  
      HOUSENUMBER,  
      POSTCODE,  
      PHONEMOBILE,  
      PHONEHOME,  
      EMAILADDRESS )  
    VALUES ( DEFAULT,  
             :CA-FIRST-NAME,  
             :CA-LAST-NAME,  
             :CA-DOB,  
             :CA-HOUSE-NAME,  
             :CA-HOUSE-NUM,  
             :CA-POSTCODE,  
             :CA-PHONE-MOBILE,
```

You will see the refactored COBOL code *RAEXPORT.cbl* in the folder *LGACDB01/cobol*, as indicated in the diagram below.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, under the 'REFACTOREDCOBOL' project, there is a folder named 'copybooks'. This folder contains sub-folders 'cobol' and 'RAEXPORT.cbl', and files 'LGACDB01.cbs' and 'zapp.yaml'. A red box highlights this folder.
- Editor:** The main editor area displays a COBOL program titled 'RAEXPORT.cbl'. The code is as follows:

```
236      *-----*
237      * Insert row into Customer table based on customer number *
238      *-----*
239      MOVE 'INSERT-CUSTOMER' TO EM-SQLREQ
240
241      IF LGAC-NCS = 'ON'
242          EXEC SQL
243              INSERT INTO CUSTOMER
244                  ( CUSTOMERNUMBER,
245                      FIRSTNAME,
246                      LASTNAME,
247                      DATEOFBIRTH,
248                      HOUSENAME,
249                      HOUSENUMBER,
250                      POSTCODE,
251                      PHONEMOBILE,
252                      PHONEHOME,
253                      EMAILADDRESS )
254              VALUES ( :DB2-CUSTOMERNUM-INT,
255                      :CA-FIRST-NAME,
256                      :CA-LAST-NAME,
257                      :CA-DOB,
258                      :CA-HOUSE-NAME,
259                      :CA-HOUSE-NUM,
260                      :CA-POSTCODE,
261                      :CA-PHONE-MOBILE ,
262                      :CA-PHONE-HOME ,
263                      :CA-EMAIL-ADDRESS )
264
265          END-EXEC
266          IF SQLCODE NOT EQUAL 0
267              MOVE '90' TO CA-RETURN-CODE
268              PERFORM WRITE-ERROR-MESSAGE
269              EXEC CICS RETURN END-EXEC
270          END-IF
271
272          ELSE
273              EXEC SQL
274                  INSERT INTO CUSTOMER
275                      ( CUSTOMERNUMBER,
276                          FIRSTNAME,
277                          LASTNAME,
278                          DATEOFBIRTH,
279                          HOUSENAME,
                          HOUSENUMBER )
```

The code is annotated with several comments and symbols, such as asterisks (\*), hash marks (#), and various MOVE, EXEC SQL, and IF statements.

### IMPORTANT!!! DO NOT SKIP THIS STEP.

To correctly execute all the next steps and generate Java code, you will need to drag and drop refactored COBOL file *RAEXPORT.cbl* to the folder *refactoredCOBOL/cobol*, all within VS Code file explorer. The refactored COBOL program must reside in the folder *refactoredCOBOL/cobol*.

The screenshot shows the Visual Studio Code interface with the following details:

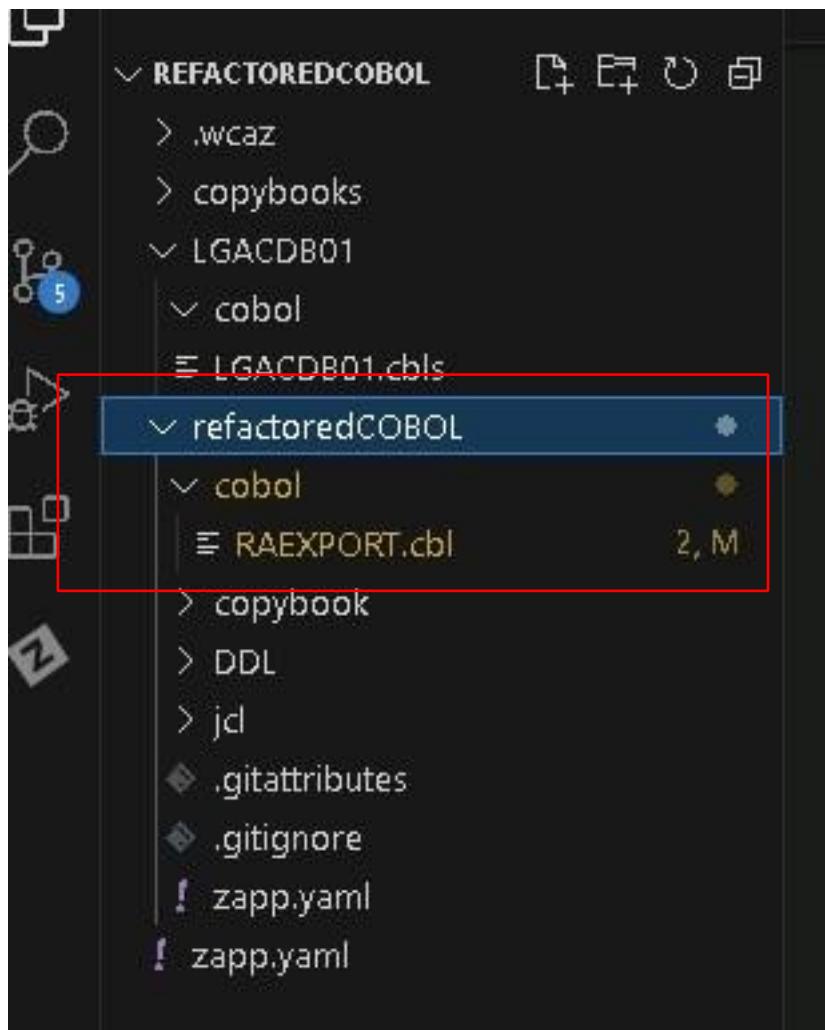
- Explorer View:** Shows the project structure under "REFACTOREDCOBOL". The "RAEXPORT.cbl" file is selected.
- Editor View:** Displays a COBOL program code. The code includes an "INSERT-CUSTOMER" procedure and logic for handling SQLCODE and errors.
- Bottom Right Modal:** A confirmation dialog from Visual Studio Code asks, "Are you sure you want to move 'RAEXPORT.cbl' into 'cobol'?". It has "Move" and "Cancel" buttons, and a checkbox for "Do not ask me again".

```

236      1 INSERT-CUSTOMER.
237      * Insert row into Customer-table based on customer number
238
239      *=====
240      MOVE ' INSERT CUSTOMER' TO EM-SOLREQ
241
242      IF LGAC-NCS = 'ON'
243          EXEC SQL
244              INSERT INTO CUSTOMER
245                  ( CUSTOMERNUMBER,
246                      FIRSTNAME,
247                      LASTNAME,
248                      DATEOFBIRTH,
249                      HOUSENAME,
250                      HOUSENUMBER,
251                      POSTCODE,
252                      PHONEMOBILE,
253                      PHONEHOME,
254                      EMAILADDRESS )
255              VALUES (:DB2-CUSTOMERNUM-INT,
256                      :CA-FIRST-NAME,
257                      :CA-LAST-NAME,
258                      :CA-DOB,
259                      :CA-HOUSE-NAME,
260                      :CA-HOUSE-NUM,
261                      :CA-POSTCODE,
262                      :CA-PHONE-MOBILE,
263                      :CA-PHONE-HOME,
264                      :CA-EMAIL-ADDRESS )
265
266          END-EXEC
267          IF SQLCODE NOT EQUAL 0
268              MOVE '90' TO CA-RETURN-CODE
269              PERFORM WRITE-ERROR-MESSAGE
270              EXEC CICS RETURN END-EXEC
271          END-IF
272      ELSE
273          EXEC SQL
274              INSERT INTO CUSTOMER
275                  ( CUSTOMERNUMBER,
276                      FIRSTNAME,
277                      LASTNAME,
278                      DATEOFBIRTH,
279                      HOUSENAME,
                      HOUSENUMBER )

```

Move the refactored COBOL program to *refactoredCOBOL/cobol* folder in VS Code. You may see changes that need to be staged and pushed to your Git repo. If so, add a commit message and then “Commit & Push” your copybooks to your *refactoredCOBOL* Git repo. You should be all set to generate the Java code.



END OF REFACTOR SECTION

# 4 03 Transform

---

All the steps below are performed within the ADDI Server image.

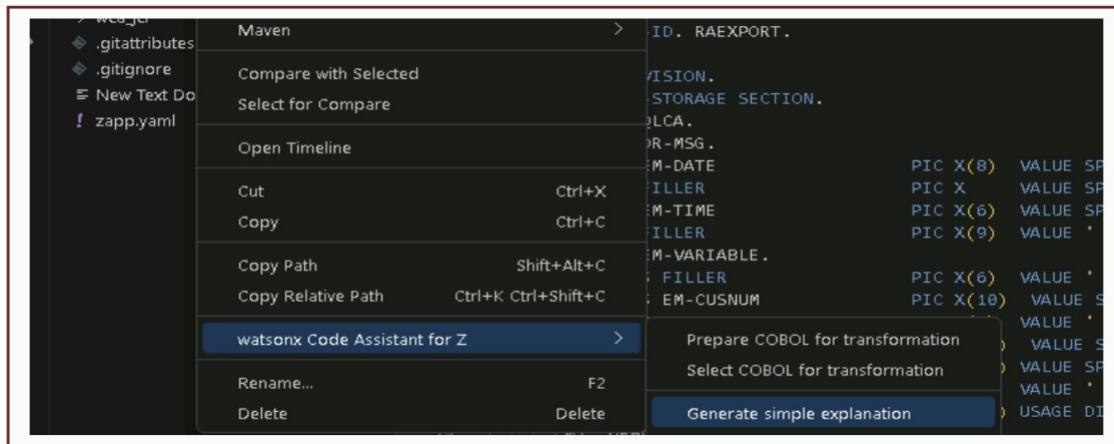
Before you start to transform the refactored COBOL to Java:

1. Ensure your refactored COBOL code is in your *refactoredCOBOL* repo
2. Git Push your changes to the Git repo

## Step 1 - Executing Code Explanation:

In your VSCode, right click on your refactored COBOL program -> watsonx code Assistant for Z ->

Generate simple explanation



## 4.1 Step 2 – Build Meta Data

Run the *automation.bat* located in:

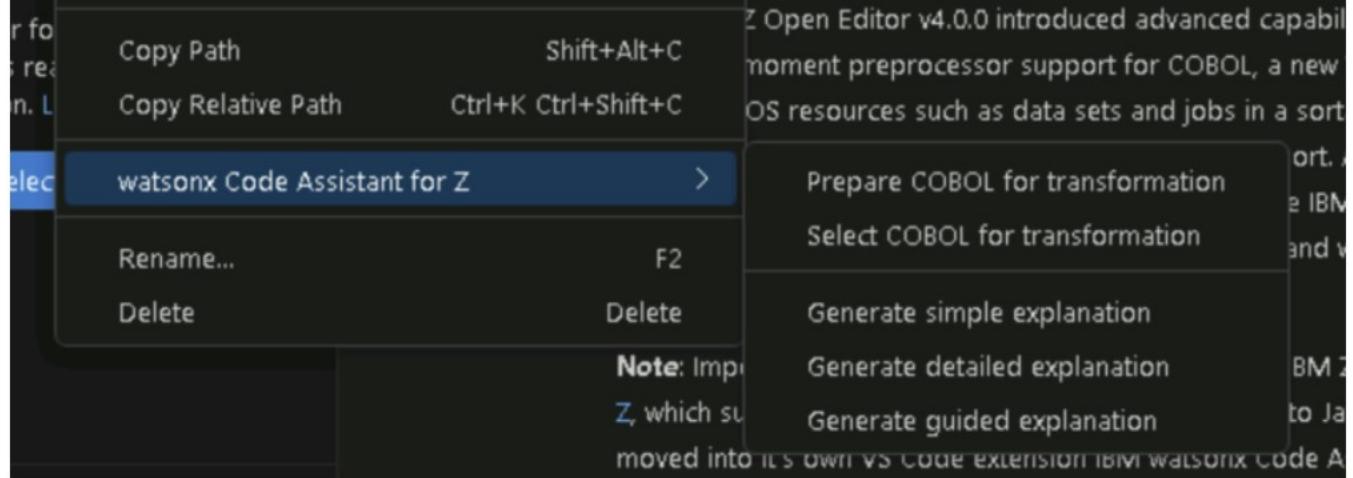
"C:\Program Files\IBM Application Discovery and Delivery Intelligence\WCA4Z Data Generator" via a Windows Command Prompt (cmd) in your ADDI Server image.

```
cmd> cd "C:\Program Files\IBM Application Discovery and Delivery Intelligence\WCA4Z Data Generator"  
cmd> automation.bat
```

This step populates the DB2 Meta Data, providing the context the LLM Service uses to generate Java classes and methods.

## 4.2 Step 3 – Select COBOL Program

Right-click on the refactored COBOL program *RAEXPORT.cbl* you want to transform to Java, select “*watsonx Code Assistant for Z > Select COBOL for transformation*”, as indicated in the diagram below.



This adds the selected COBOL program `RAEXPORT.cbl` to “*IBM WATSONX CODE ASSISTANT FOR Z > cobol*” folder located on the bottom of the *Explorer* pane on the left side, as indicated in the diagram below.

## 4.3 Step 4 - Generate Java Classes

Click the “Generate Java classes” icon (as indicated by the red arrow in the diagram below) to generate Java classes.

The screenshot shows the IBM Watsonx Code Assistant for Z interface. On the left, the Explorer sidebar displays a project structure under 'REFACTOREDCOBOL' with files like '.wcaz', 'cobol', '.gitkeep', and 'RAEXPORT.cbl'. The 'TIMELINE' and 'PROJECT ARTIFACTS' sections are also visible. A red arrow points to the 'Browse for directory' button next to the 'RAEXPORT.cbl' file entry in the list.

The main window shows the COBOL source code for 'RAEXPORT.cbl':

```
1 ****
2 * Created: Sun, 2 Jun 2024 23:24:34 GMT
3 * Generated by: IBM Watsonx Code Assistant for Z Refactoring
4 * Assistant
5 * Workbook name: LGACDB01
6 * Workbook id: dd334541-b1ca-457c-9c80-edbd43a4bb14
7 * Project: $clientCOBOL_894592fa-e5e2-40fa-a195-4bfff9965c532
8 ****
9 IDENTIFICATION DIVISION.
10 PROGRAM-ID. RAEXPORT.
11
12 DATA DIVISION.
13 WORKING-STORAGE SECTION.
14 COPY SQLCA.
15 01 DB2-OUT-INTEGERS.
16   03 DB2-CUSTOMERNUM-INT PIC S9(9) COMP.
17
18   77 LGAC-NCS          PIC X(2) VALUE 'ON'.
19   01 WS-ABSTIME        PIC S9(8) COMP VALUE +0.
20   01 WS-TIME           PIC X(8)  VALUE SPACES.
21   01 WS-DATE           PIC X(18) VALUE SPACES.
22
23   01 DFHCOMMAREA-1.
24     COPY LGCMAREA.
25   01 ERROR-MSG.
26
27   03 EM-DATE           PIC X(8)  VALUE SPACES.
28   03 FILLER             PIC X    VALUE SPACES.
29   03 EM-TIME           PIC X(6)  VALUE SPACES.
30   03 FILLER             PIC X(9)  VALUE ' LGACDB01'.
31   03 EM-VARIABLE.
32     05 FILLER           PIC X(6)  VALUE ' CNUM='.
33     05 EM-CUSNUM         PIC X(18) VALUE SPACES.
34     05 FILLER           PTC X(6)  VALUE ' PNJIM='.
35
36
37
```

Click on “*Browse for directory*” to select the folder for generated Java classes, as indicated in the diagram below.

The screenshot shows the IBM WatsonX Code Assistant for Z Refactoring interface. On the left is the Explorer sidebar with sections like REFACTOREDCOBOL, cobol, and PROJECT ARTIFACTS. The main area displays a COBOL source code editor with the following content:

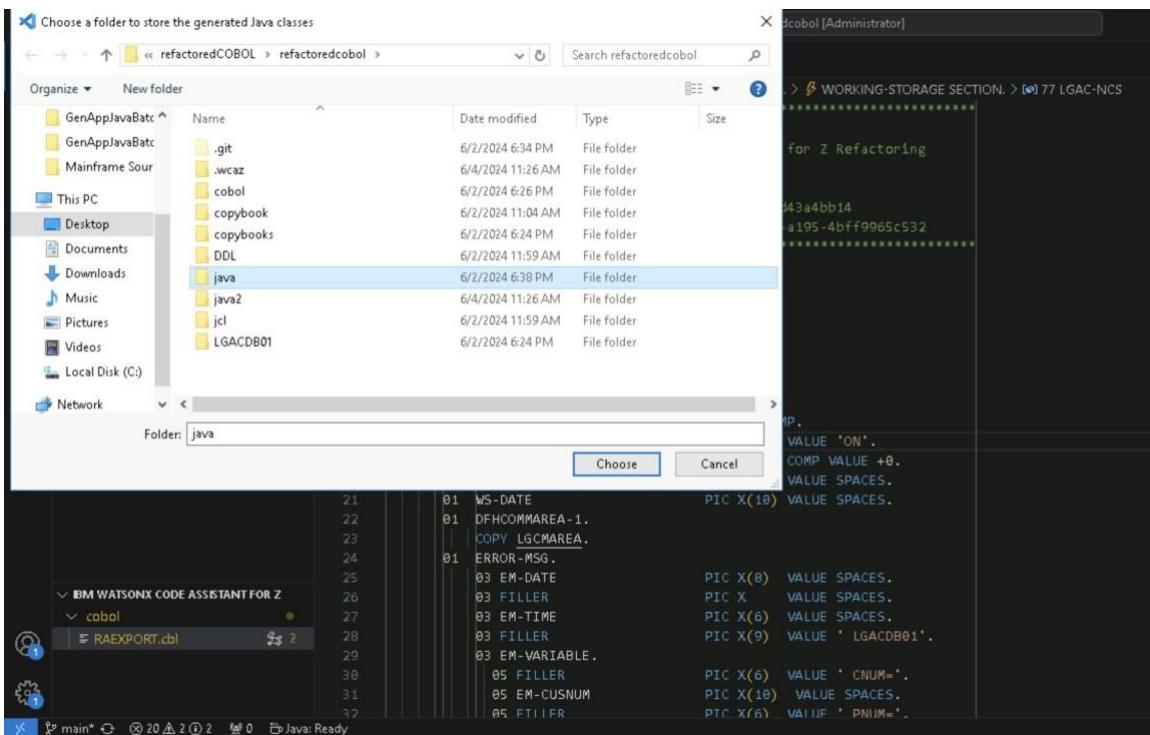
```

1  *RAEXPORT
2  IDENTIFICATION DIVISION.
3  PROGRAM-ID. RAEXPORT.
4
5  DATA DIVISION.
6  WORKING-STORAGE SECTION.
7  COPY SQLCA.
8  01 DB2-OUT-INTGERS.
9    03 DB2-CUSTOMERNUM-INT PIC S9(9) COMP.
10   03 LGAC-NCS          PIC X(2) VALUE 'ON'.
11   01 WS-ABSTIME        PIC S9(8) COMP VALUE +0.
12   01 WS-TIME           PIC X(8)  VALUE SPACES.
13   01 WS-DATE           PIC X(18) VALUE SPACES.
14   01 DFHCOMMAREA-1.
15     COPY LGCMAREA.
16   01 ERROR-MSG.
17     03 EM-DATE          PIC X(8)  VALUE SPACES.
18     03 FILLER            PIC X   VALUE SPACES.
19     03 EM-TIME          PIC X(6)  VALUE SPACES.
20     03 FILLER            PIC X(9)  VALUE 'LGACDB01'.
21     03 EM-VARIABLE.
22       05 FILLER          PIC X(6)  VALUE 'CNUM='.
23       05 EM-CUSNUM        PIC X(18) VALUE SPACES.
24       05 FILLER          PIC X(6)  VALUE 'PNJIM='.

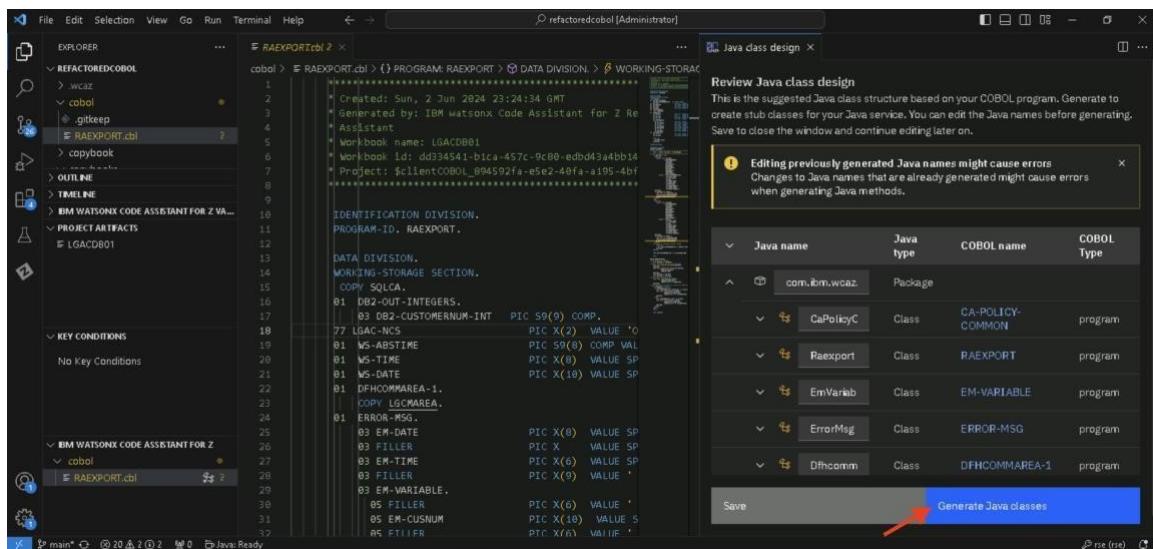
```

A context menu is open at the top right of the code editor, with the "Browse for directory" option highlighted.

Select the folder *java* from the pop-up window. You can create a new folder and select it, such as *java2*.



The right side of the diagram below shows the *Java class design*. Here, you can review the classes and their attributes and make changes if desired. Notice these classes are derived from the COBOL copybooks and DDLs. Click the “Generate Java classes”, as indicated by the red arrow.



Java classes are generated in the folder *java* (or whatever you selected before). Expand the folder on the *Explorer* pane to see the classes generated. Review the classes and understand the relationships between them.

```

package com.ibm.wcaz.implementation;

import com.ibm.json.fields.CobolDatatypeFactory;
import com.ibm.json.fields.External10cMma1AsIntField;
import com.ibm.json.fields.StringField;
import java.io.UnsupportedEncodingException;
import java.util.Arrays;

public class CaMotor extends CaPolicyRequest {
    private String caMMake = "";
    private String caModel = "";
    private String caMValue;
    private String caRegnumber = "";
    private String caColour = "";
    private int caMCC;
    private String caManufactured = "";
    private int caMPremium;
    private int caMAccidents;
    private String caMMiller = "";

    public CaMotor() {}

    public CaMotor(String caRequestId, int caReturnCode, long caCustomerNum, long caPolicyNum, CaPolicyCommon caPolicyCommon) {
        super(caRequestId, caReturnCode, caCustomerNum, caPolicyNum, caPolicyCommon);
        this.caMMake = caMMake;
        this.caModel = caModel;
        this.caMValue = caMValue;
        this.caRegnumber = caRegnumber;
        this.caColour = caColour;
        this.caMCC = caMCC;
        this.caManufactured = caManufactured;
        this.caMPremium = caMPremium;
    }
}

```

13 Java classes generated in c:\Users\Administrator\Desktop\WCA4Z-Data\refactoredCOBOL\refactoredcobol\java  
Source: IBM Z Open Editor (Extension)

## 4.4 Step 5 - Generate Java Method

Select and review the Java class that matches the COBOL program name *Raexport.java*.

Then, find and select the method *insertCustomer* that you want to generate Java code.

```

public static Raexport fromBytes(String bytes) {
    try {
        return fromBytes(bytes.getBytes(factory.getStringEncoding()));
    } catch (UnsupportedEncodingException e) {
        throw new RuntimeException(e);
    }
}

View COBOL source
public static void insertCustomer() {}

Run[Debug]
public static void main(String[] args) {
    insertCustomer();
}

public String toString() {
    StringBuilder s = new StringBuilder();
    s.append("Raexport[" + this.toString());
    return s.toString();
}

public boolean equals(Raexport that) {
    return true;
}

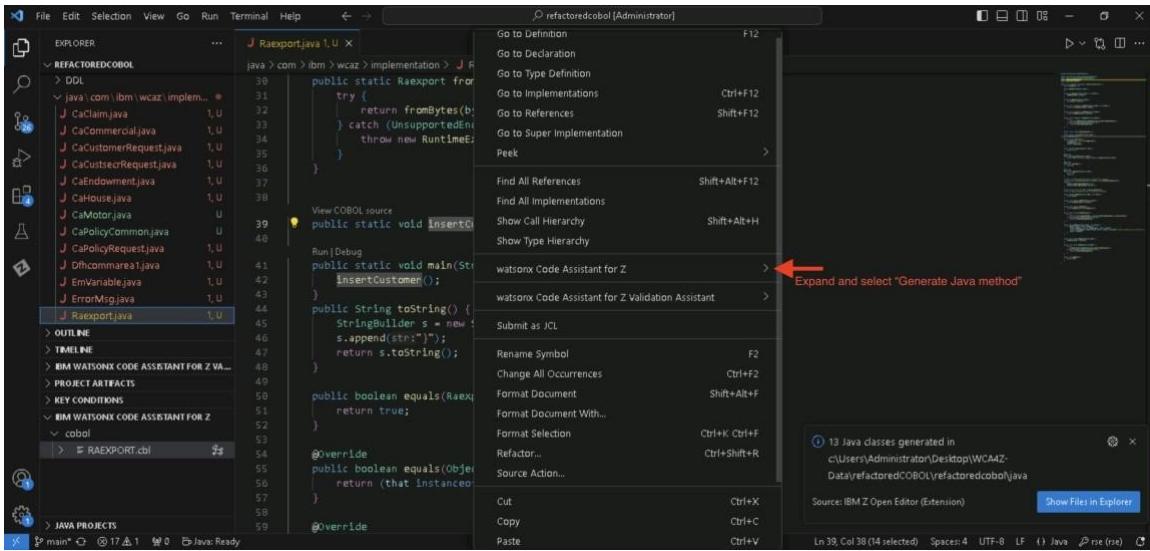
@Override
public boolean equals(Object that) {
    return (that instanceof Raexport) && this.equals((Raexport)that);
}

@Override

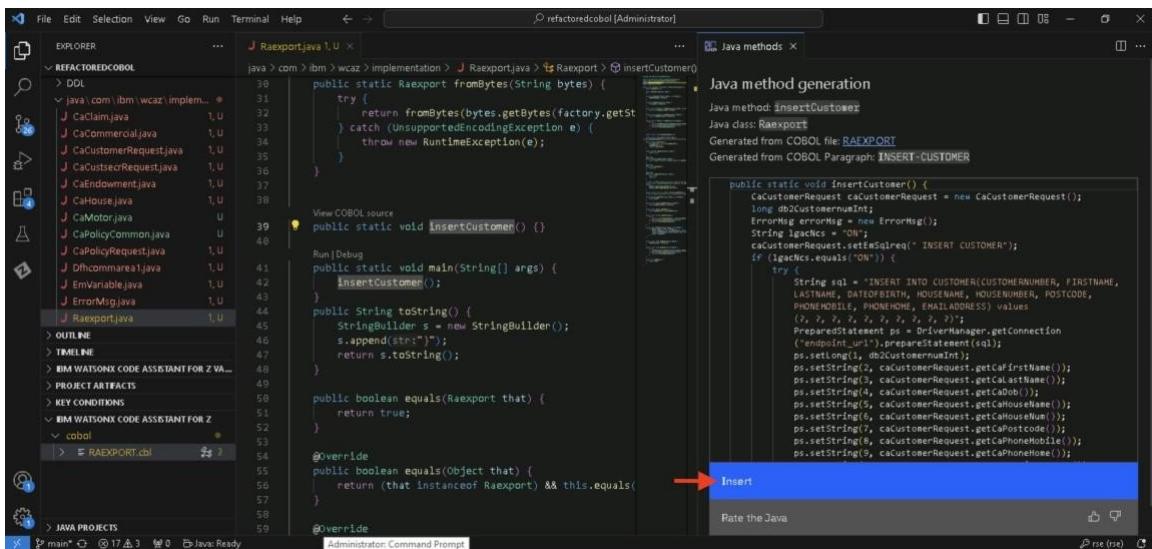
```

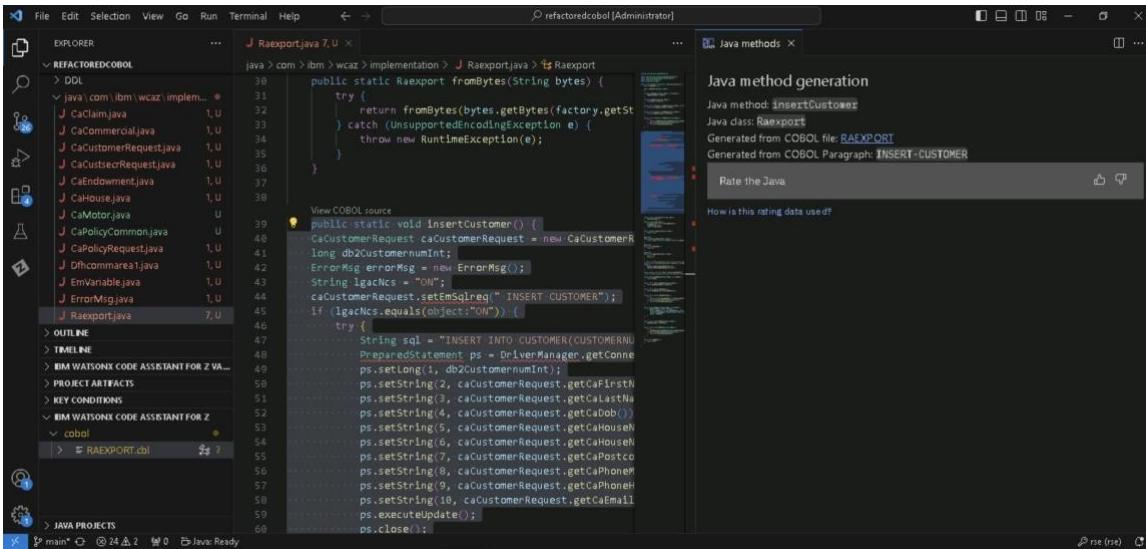
13 Java classes generated in c:\Users\Administrator\Desktop\WCA4Z-Data\refactoredCOBOL\refactoredcobol\java  
Source: IBM Z Open Editor (Extension)

Right-click on the method, select “watsonx Code Assistant for Z > Generate Java method”, as indicated in the diagram below.



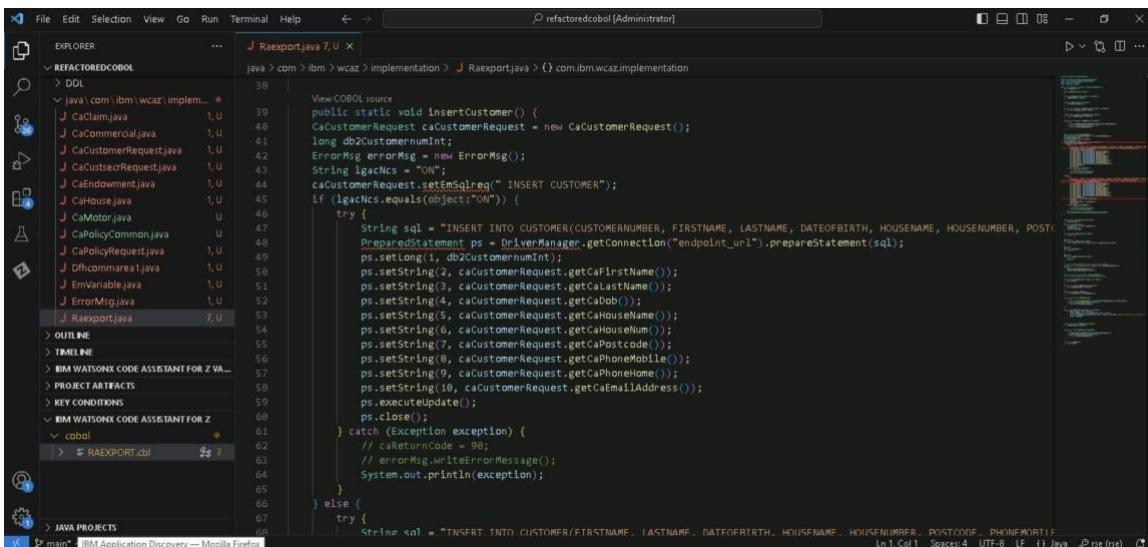
The right-side pane shows the *Java method generation*, as indicated in the diagram below. Here, you can review the code and click “*Insert*” to insert the code into the Java method. This code is generated based on the code in the COBOL program.





## 4.5 Step 6 – Review and Improve Java Code

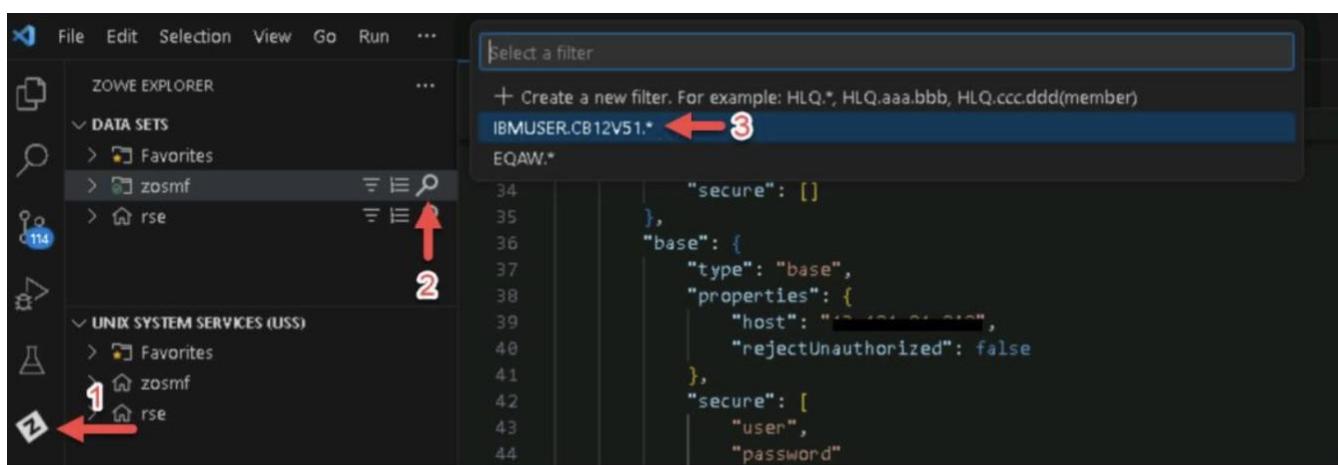
Take your time reviewing the code and comparing it with the COBOL code. You may have to fix some of the code to eliminate the errors and refine it to use the specific code library and configurations specific to your application, such as the JDBC driver and URL to the underlying datastore. Notice that the tool is an assistant to the Java developers — it's not meant to replace them completely.



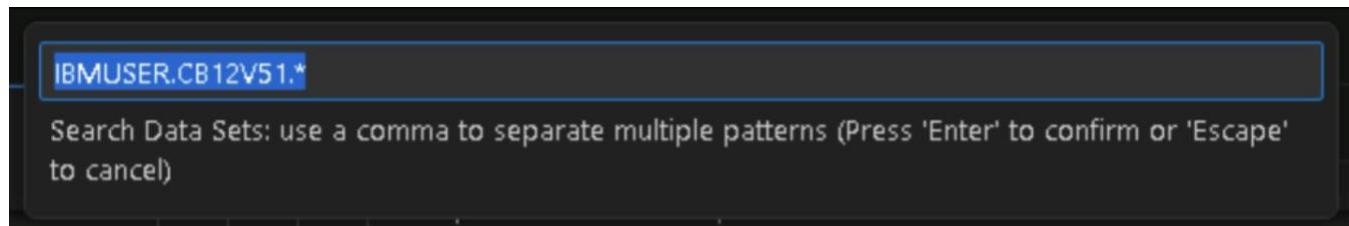
WCA4Z also supports code validation by auto-generating test cases to determine whether the generated Java code is equivalent to the COBOL code.

## 5 04 Validation

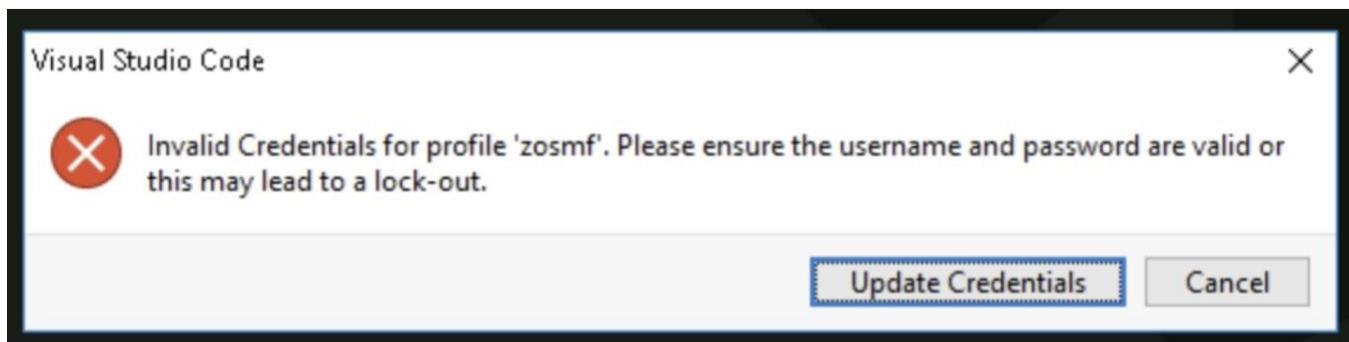
Within VS Code on your ADDI Windows instance, navigate to the ‘ZOWE Explorer’ tab on the left-hand menu In VS Code, connect to z/OS by clicking on the magnifying glass next to ‘zosmf’ and select the filter IBMUSER.CB12V51.\* in the Zowe Explorer. If the filter does not exist, then create a new filter.



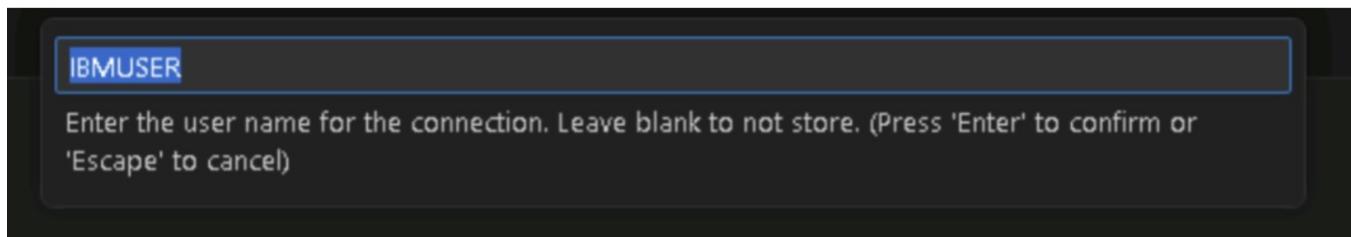
Click enter when you see this at the top of the screen:



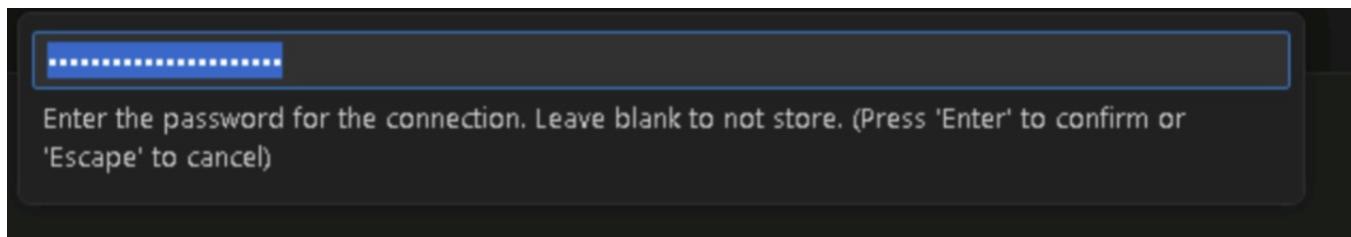
A message may then appear related to ‘Invalid Credentials’ for the zosmf profile. Click “Update Credentials”:



When prompted for the user name, leave the default “IBMUSER” and click enter:



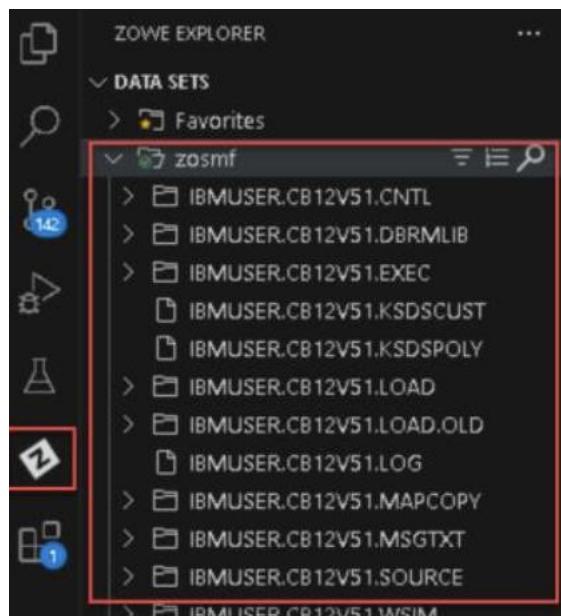
Then enter in the password and click enter-



And lastly, click on the magnifying glass again next to ‘zosmf’ with the filter and hit enter.

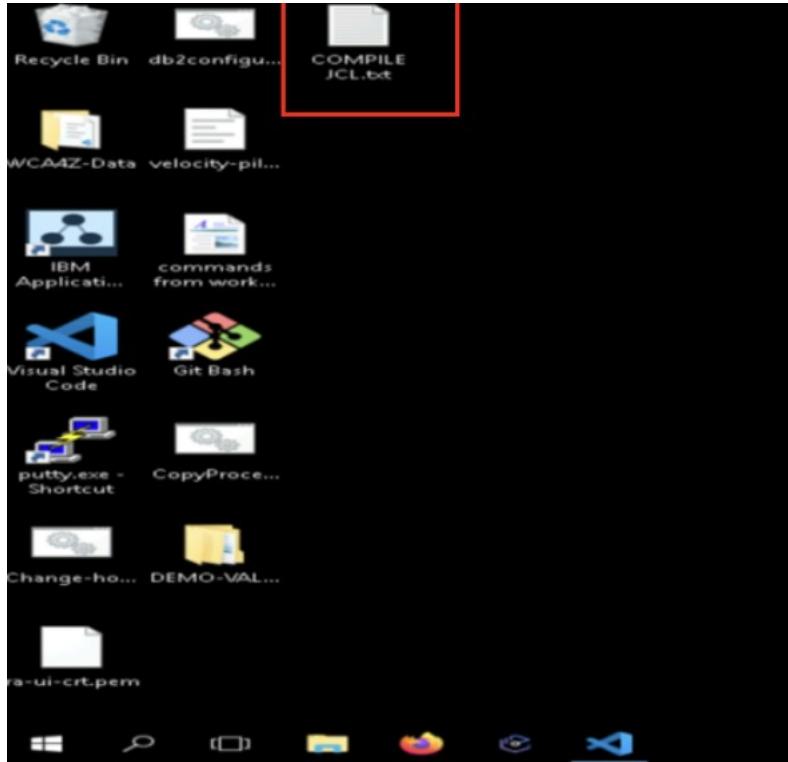
You should see a list of datasets show up here now, if not, please ensure you correctly

entered the passphrase you changed and that you have entered the correct public IP address in the step above  
OR reselect the filer to IBMUSER.CB12V51.\*

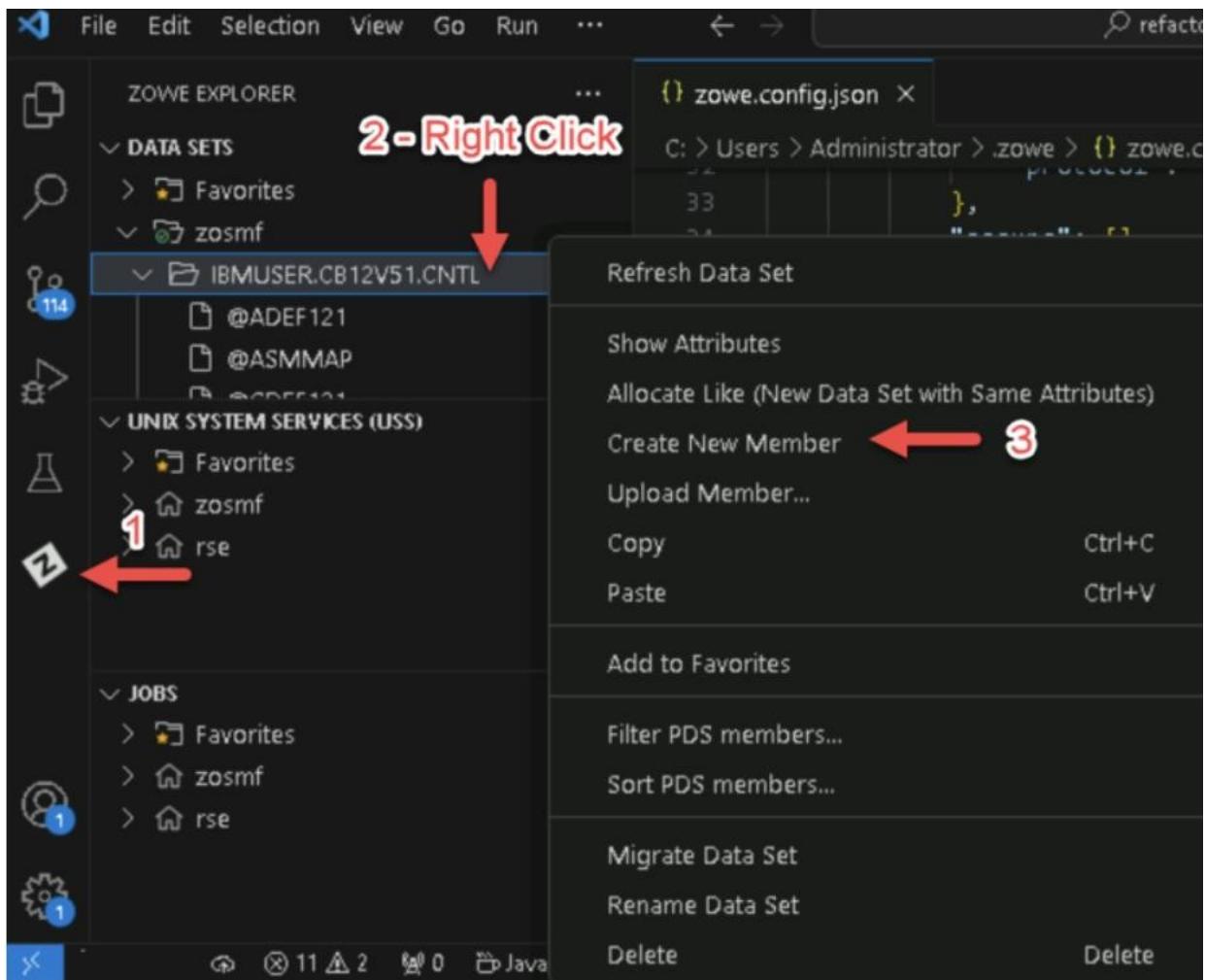


Create a compilation member in IBMUSER.CB12V51.CNTL dataset

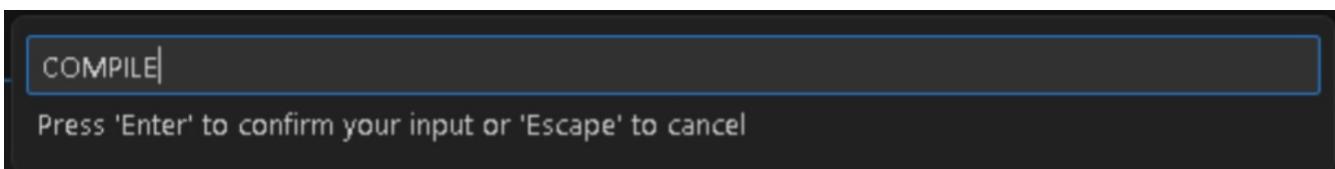
1. Locate the ‘COMPILE JCL.txt’ file on the Desktop in your Windows Image and copy its contents.



2. Within VS Code, navigate to ZOWE Explorer, right-click on the 'IBMUSER.CB12V51.CNTL' dataset, and then select 'Create New Member':



3. Type 'COMPILE' as the name of the new member and hit enter:



4. Past the copied JCL contents from the desktop into the editor for the COMPILE member and save it:

```

25 //STEPLIB DD DSN=IGY.V6R4M0.SIGYCOMP,DISP=SHR
26 // DD DSN=CICSTS.V6R1M0.CICS.SDFHLOAD,DISP=SHR
27 // DD DSN=DB2.V13R1M0.SDSNEXIT,DISP=SHR
28 // DD DSN=DB2.V13R1M0.SDSNLOAD,DISP=SHR
29 //SYSLIB DD DSN=CICSTS.V6R1M0.CICS.SDFHCOB,DISP=SHR
30 // DD DSN=CICSTS.V6R1M0.CICS.SDFHMAC,DISP=SHR
31 // DD DSN=CICSTS.V6R1M0.CICS.SDFHSAMP,DISP=SHR
32 // DD DSN=DB2.V13R1M0.SDSNSAMP,DISP=SHR
33 // DD DSN=DB2.V13R1M0.SDSNMACS,DISP=SHR
34 // DD DSN=IBMUSER.CB12V51.MAPCOPY,DISP=SHR
35 // DD DSN=IBMUSER.CB12V51.SOURCE,DISP=SHR
36 //SYSIN DD DISP=SHR,DSN=IBMUSER.CB12V51.SOURCE(&MEM)
37 //DBRMLIB DD DSN=IBMUSER.CB12V51.DBRLIB(&MEM),DISP=SHR
38 //SYSLIN DD DSN=&&LOADSET,DISP=(NEW,PASS),UNIT=SYSDA,

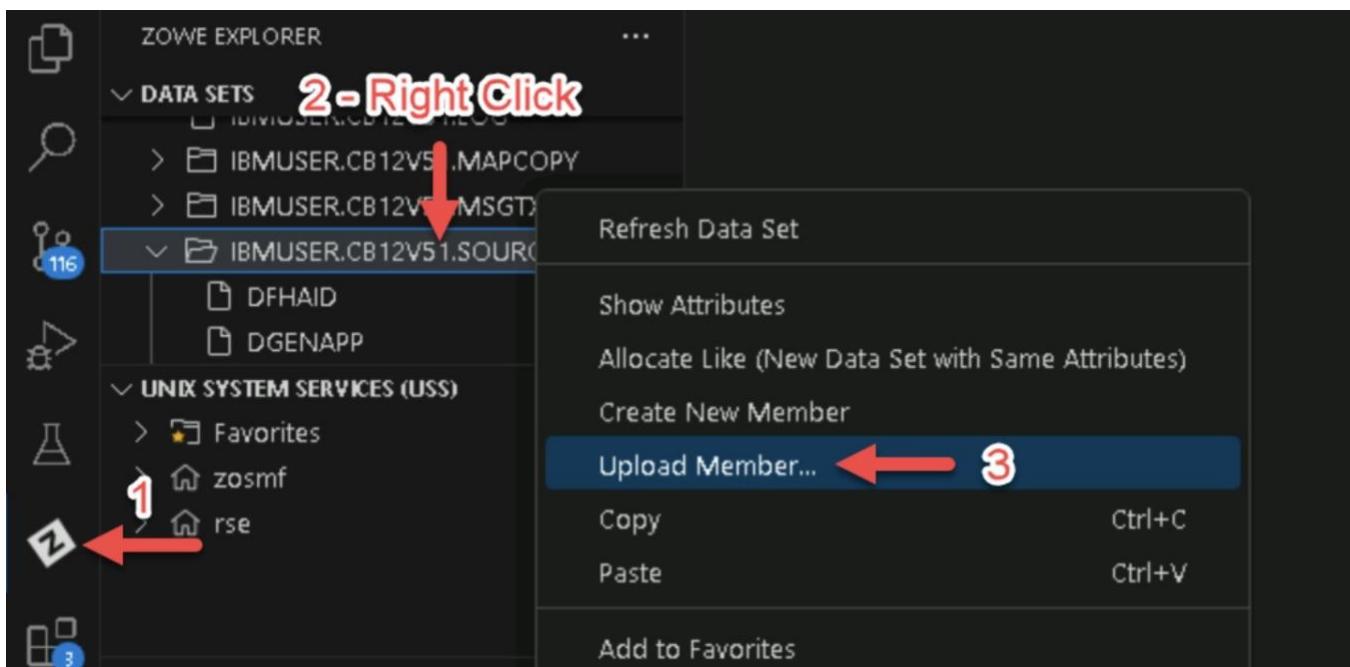
```

PASTE THE COPIED JCL HERE

PLEASE NOTE: This JCL will be used to compile and link-edit your refactored curated COBOL.

Compile your refactored curated code

1. Locate the IBMUSER.CB12V51.SOURCE dataset and upload your refactored COBOL to it:



2. Now go to your IBMUSER.CB12V51.CNTL(COMPILE) member and update line 103.

Change MEM= to the refactored program name that you uploaded into

IBMUSER.CB12V51.SOURCE and click save (see below):

```

ZOWE EXPLORER
DATA SETS
  COBOL
  COBOLBR
  COMP
  COMPILE
  CPSMDE2

IBMUSER.CB12V51.CNTL(COMPILE).jd
C: > Users > Administrator > .vscode > extensions > zowe.vscode-extension-for-zowe-2.15.1 > resources > temp > _D_ > zosmf > IBMUSER.CB12V51.SOURCE

96 //** DD DISP=SHR,DSN=IBMUSER.CB12V51.SOURCE(LINKPARM)
97 //SYSLIN DD DISP=(OLD,DELETE),DSN=&&COPYLINK
98 // DD DISP=(OLD,DELETE),DSN=&&LOADSET
99 // DD DISP=SHR,DSN=IBMUSER.CB12V51.SOURCE(LINKPARM)
100 // PEND
101 /**
102 /**
103 //PROGRAM EXEC DB2PROC, MEM=REFCOBOL
104

```

3. After saving, submit the COMPILE JCL member by right-clicking anywhere in the file and select “Submit as JCL”:

```

IBMUSER.CB12V51.CNTL(COMPILE).jd
Administrator > .vscode > extensions > zowe.vscode-extension-for-zowe-2.15.1 > resources > temp > _D_ > zosmf > IBMUSER.CB12V51.SOURCE

96 //** DD DISP=SHR,DSN=IBMUSER.CB12V51.SOURCE(LINKPARM)
97 //SYSLIN DD DISP=(OLD,DELETE),DSN=&&COPYLINK
98 // DD DISP=(OLD,DELETE),DSN=&&LOADSET
99 // DD DISP=SHR,DSN=IBMUSER.CB12V51.SOURCE(LINKPARM)
100 // PEND
101 /**
102 /**
103 //PROGRAM EXEC DB2PROC, MEM=REFCOBOL
104

Right Click and Select
Submit as JCL
Change All Occurrences Ctrl+F2
Refactor... Ctrl+Shift+R
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Command Palette... Ctrl+Shift+P

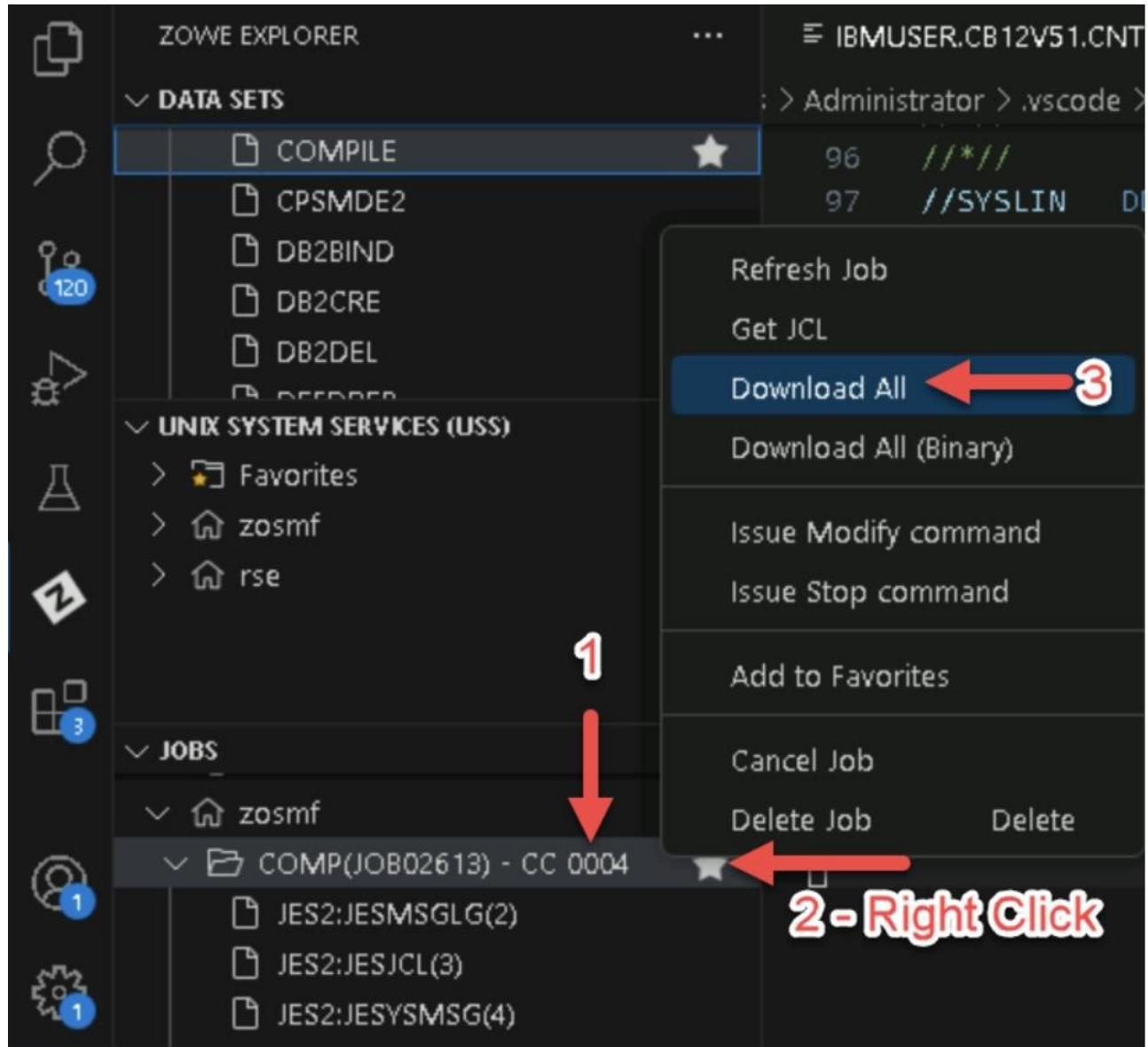
```

4. After submitting, a message should appear at the bottom right of the screen indicating that the Job was submitted. Click on the Job number:

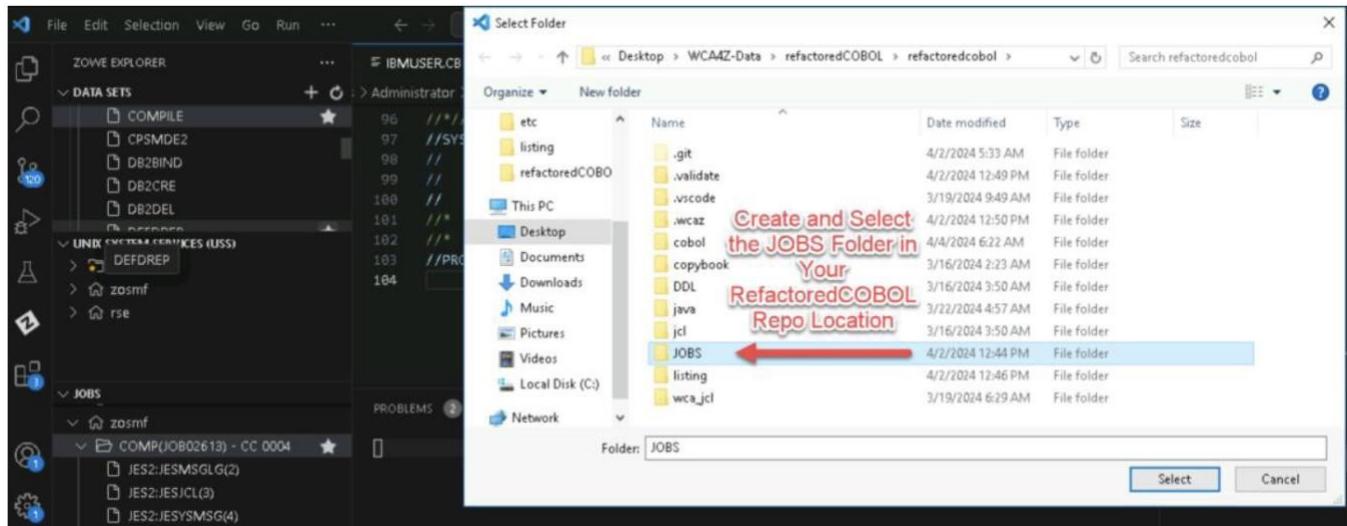


5. This should take you to the job output where you are expecting to see a return code 0 or 4 (see below as an example). If you don't see 4 or 0, then you need to review the output and fix it with your clients COBOL developer / resource.

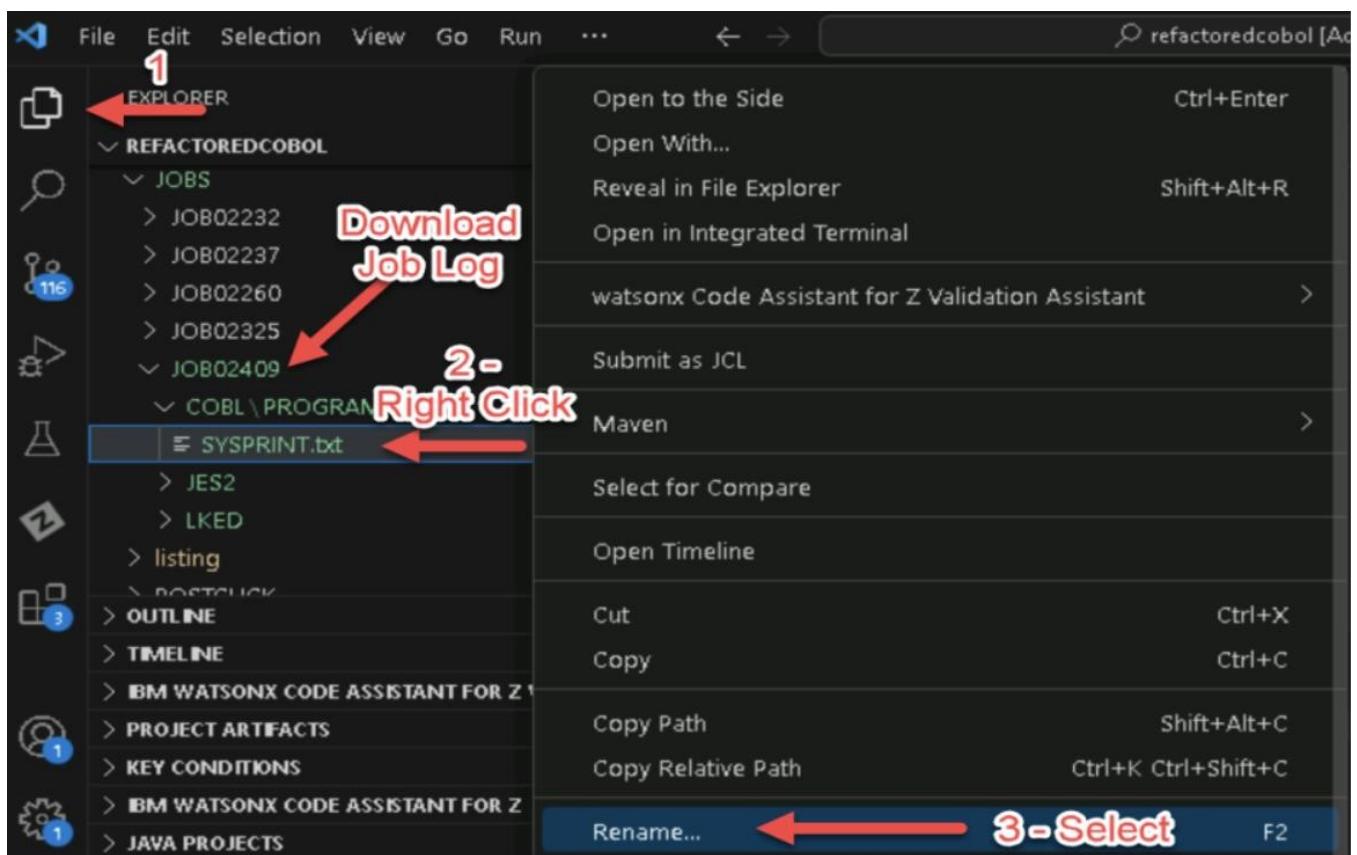
Now right-click on the COMP job output and select “Download All”:



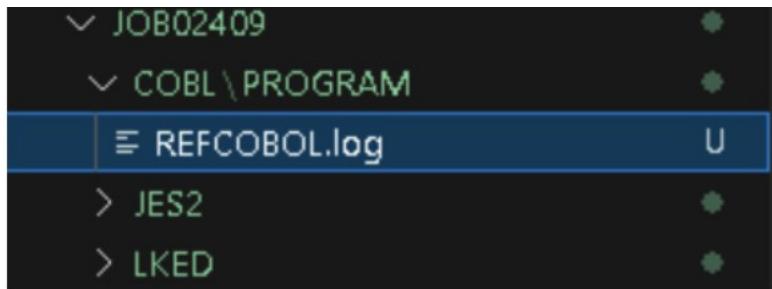
6. Then create and select the 'JOBS' folder in C:\Users\Administrator\Desktop\WCA4Z-Data\refactoredCOBOL\refactoredcobel



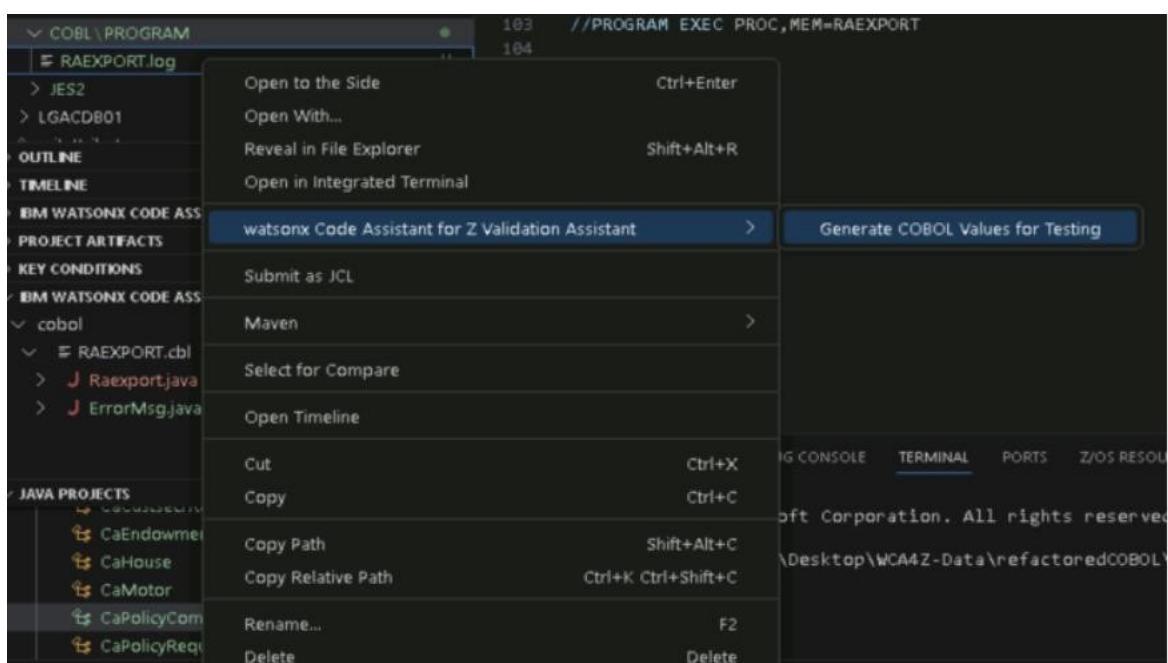
7. Within VS Code, click on the Explorer tab to locate the download job log in your JOBS folder in Project Explorer view in VS Code and rename the SYSPRINT.txt file to <your refactored program name>.log



This is an example, you should rename it to your <your refactored program name>.log



Now right click on the <your refactored program name>.log and select 'Generate COBOL Values for Testing':



You will then be prompted for the location of the load library at the top of the VS Code window. Enter "IBMUSER.CB12V51.LOAD" and hit ENTER:

```
IBMUSER.CB12V51.LOAD
Enter the location of the load library Ex. HLQ.PROJECT.LOAD (Press 'Enter' to confirm or 'Escape' to cancel)

95 // /**/      DD DISP=(OLD,DELETE),DSN=&&LOADSET
96 // /**/      DD DISP=SHR,DSN=IBMUSER.CB12V51.SOURCE(LINKPARM)
97 //SYSLIN     DD DISP=(OLD,DELETE),DSN=&&COPYLINK
98 //          DD DISP=(OLD,DELETE),DSN=&&LOADSET
99 //          DD DISP=SHR,DSN=IBMUSER.CB12V51.SOURCE(LINKPARM)
100 //          PEND
```

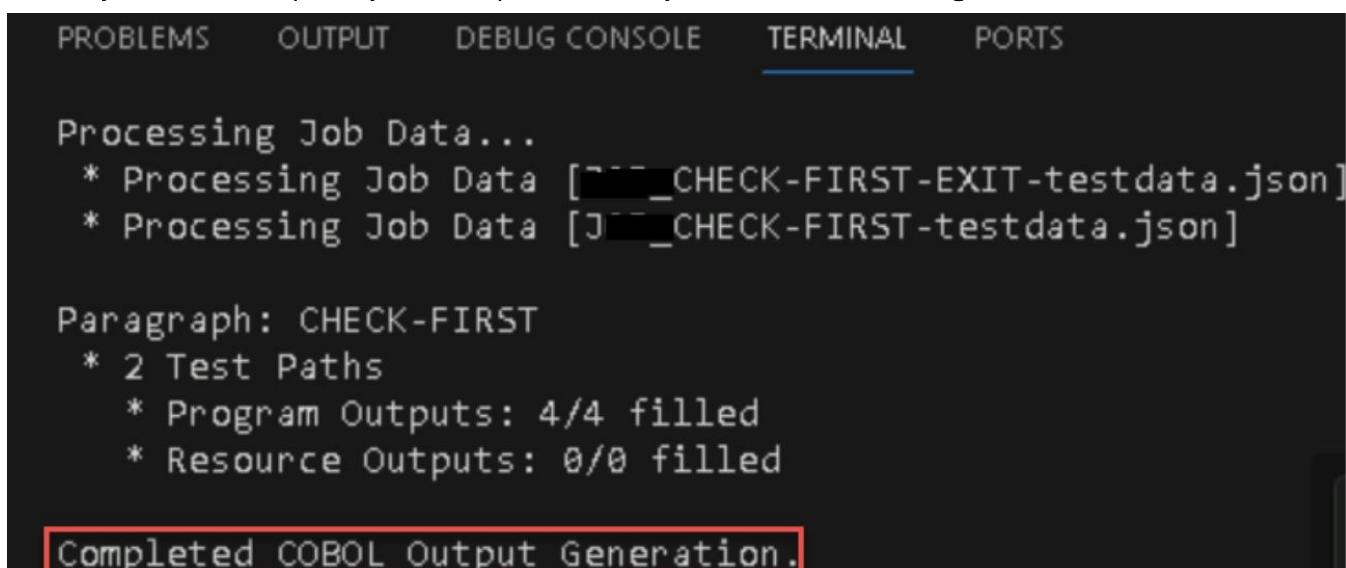
If you see this error below in the output: Failed to Generate Cobol Outputs: Error:

Failed Running Generating COBOL Outputs:

CodeExpectedError: cannot open file:///c%3A/Users/Administrator/Desktop/WCA4Z-Data/refactoredCOBOL/refactoredcobol/.validate/java/debugger/output/jcls/<program name>\_<paragraph name>-testdata\_test\_1\_out.txt

Then please close VS Code and delete this folder before reopening VS Code and retrying 'Generate COBOL Outputs' above: C:\Users\Administrator\Desktop\WCA4Z-Data\refactoredCOBOL\refactoredcobol\.validate

Ensure you see this output, if you don't, please intently look at the OUTPUT log



The screenshot shows the VS Code interface with the 'TERMINAL' tab selected. The terminal window displays the following text:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Processing Job Data...
* Processing Job Data [J--_CHECK-FIRST-EXIT-testdata.json]
* Processing Job Data [J--_CHECK-FIRST-testdata.json]

Paragraph: CHECK-FIRST
* 2 Test Paths
* Program Outputs: 4/4 filled
* Resource Outputs: 0/0 filled

Completed COBOL Output Generation.
```

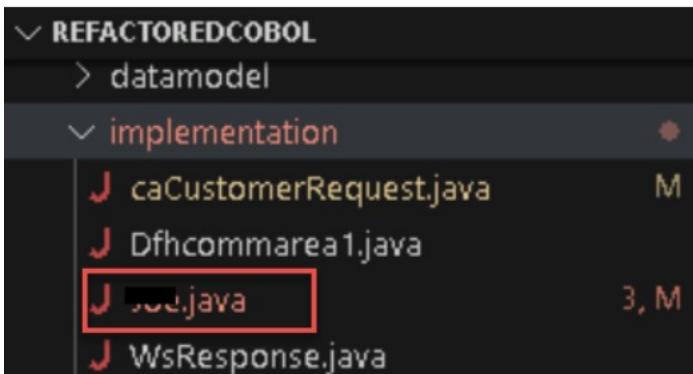
The last line, "Completed COBOL Output Generation.", is highlighted with a red box.

## Executing the Equivalence Test

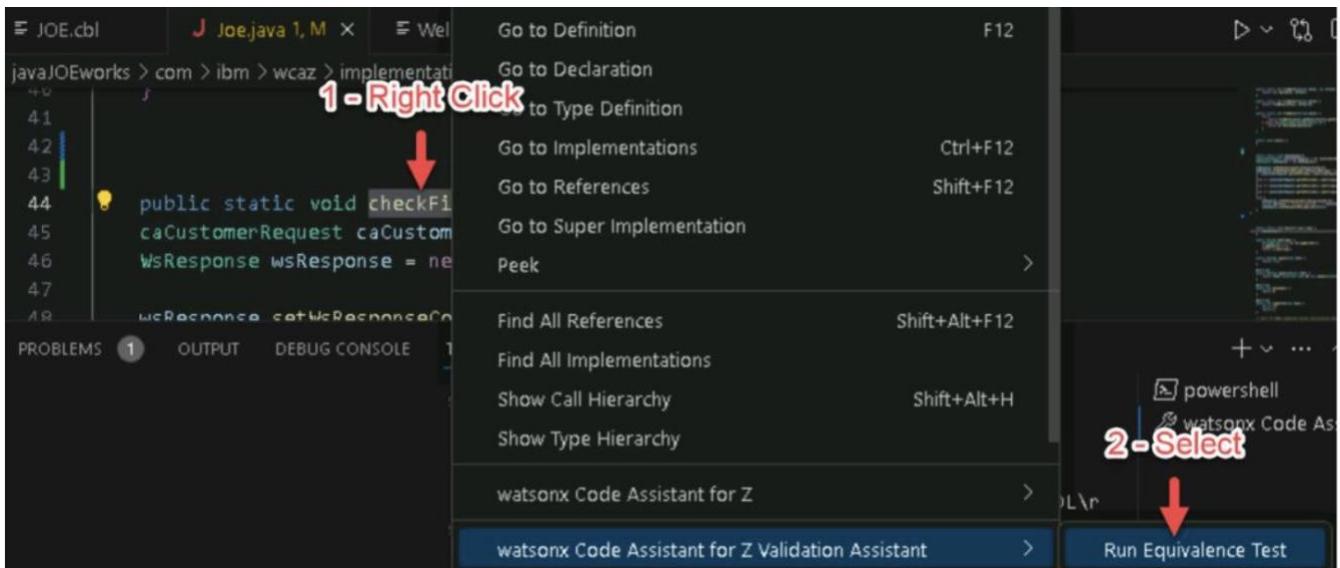
Make sure the transformed Java is compile-able using VS Code

The screenshot shows the Rational Developer for COBOL interface. In the center, there is a code editor window displaying Java code. A red arrow points from the text "Fix all the syntax errors in the problems tab" to the "PROBLEMS" tab at the bottom left of the editor. Another red arrow points from the text "Look for red lines and hover mouse over to resolve them" to a red box highlighting a syntax error in the code. The code editor has a tooltip "Duplicate method checkFirst() in type Joe Java(67109219)" pointing to a specific line. The "PROBLEMS" tab shows several errors, each with a red icon and a message like "The type Genadb1Customer is already defined Java(16777539)". The bottom status bar shows "Ln 44, Col 20" and other settings.

When your Java is highlighted in ‘red’ it likely means your Java has syntax issues and will likely not compile during the Validation phase.



Once your Java has no syntax issues, highlight the relevant Java method which has the generated Java code, and then run the ‘equivalence test’



Wait for either a successful message like below:

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

cdol/java: works/com/ibm/wcaz/implementation/Joe.java>

Testing Semantic Equivalence:
Retrieving Method Validation Information...
 * Getting COBOL outputs from: c:\Users\Administrator\Desktop\WCA4Z-Data\refactoredCOBOL\refactoredcobol\validate\..\debugger\output\values\__CHECK-FIRST-testdata-values.json
Preparing Java Files...
Creating JUnit Comparison Tests...
Compiling...
Running Tests...
Equivalency Tests Complete.

```

Or if there is a failed execution of the generated J Unit tests, then look at the Terminal output log and attempt to address the issues. ITS VITAL TO ENSURE YOU HAVE A JAVA DEVELOPER WITH YOU FROM YOUR CLIENT TO HELP DIAGNOSE AND VALIDATE THE ISSUE.

Locate the J Unit test here to see what was generated and then executed against your

Java here: C:\Users\Administrator\Desktop\WCA4Z- Data\refactoredCOBOL\refactoredcobol\validate\<project name>\generated\_code\test\com\ibm\wcaz\implementation