

RBC x IBM

Bootcamp for watsonx Code Assistant for Z

Revolutionizing Mainframe Application Modernization with AI



Table of Contents

1	Introduction	4
1.1	About this hands-on lab	4
2	Log In to the Environment.....	4
3	Launch Z Understand Eclipse client (formerly ADDI) IBM Developer for z/OS - IDz)	5
4	Visualize the entire Application Landscape.....	8
5	Deep Dive into Specific Applications	11
6	Code explanation – Understand Eclipse client / ADDI and Refactor Assistant (RA)	16
7	Use Refactor Assistant (RA)	19
7.1.1	Complete the following steps	22
8	Transformation – Converting Cobol to Java.....	31
8.1	Reach out to the Lab instructor.....	39

Notices and disclaimers

© 2025 International Business Machines Corporation. No part of this document may be reproduced or transmitted in any form without written permission from IBM.

U.S. Government Users Restricted Rights – use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information.

This document is distributed “as is” without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity. IBM products and services are warranted per the terms and conditions of the agreements under which they are provided. The performance data and client

examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

IBM products are manufactured from new parts or new and used parts.

In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to ensure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Notices and disclaimers (Continued)

Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

1 Introduction

Use this section to provide a brief introduction to the product, scenario, etc.

IBM Watson Code Assistant for Z (WCA4Z) helps enterprises modernize mainframe applications by analyzing COBOL workloads, refactoring them into services, and transforming selected portions into Java. This lab provides an end-to-end walkthrough of logging in to the environment, exploring applications in ADDI, using Code Explanation, applying Refactor Assistant in VS Code, and completing COBOL-to-Java transformations.

1.1 About this hands-on lab

Use this section to provide an overview of what the user will learn from this hands-on lab.

The objective of this lab is to give participants experience with WCA4Z and related tooling. By the end of this lab, participants will have analyzed COBOL applications using ADDI, refactored code into services, and transformed a COBOL program into Java classes.

2 Log In to the Environment

Please do this right away so we can make sure everyone has access to the environment.

- ⇒ Login to:
<https://techzone.ibm.com/my/workshops/student/69138bfb7c990d772ba9c235>
- ⇒ If you don't have an IBM ID, you will have to create one.

You should automatically be assigned a workstation. You will be prompted for a password.

- ⇒ The password is **provided by your instructor**
rbc-wca4z

In one of your environments click on the remote desktop URL.

- ⇒ Click the main link: <https://vdi.cloud.techzone.ibm.com/guacamole>
- ⇒ If you have all permissions, you should see a VM popup within your browser. I recommend using Chrome if you have it, as it allows for easier copy and pasting within the VM.
- ⇒ You're all ready to go!

3 Launch Z Understand Eclipse client (formerly ADDI) IBM Developer for z/OS - IDz

ADDI (Application Discovery and Delivery Intelligence) is a graphical tool used to analyze and understand your application landscape. For this lab, we've already loaded the necessary source code.

⇒ Open IBM Developer for z/OS (IDz)

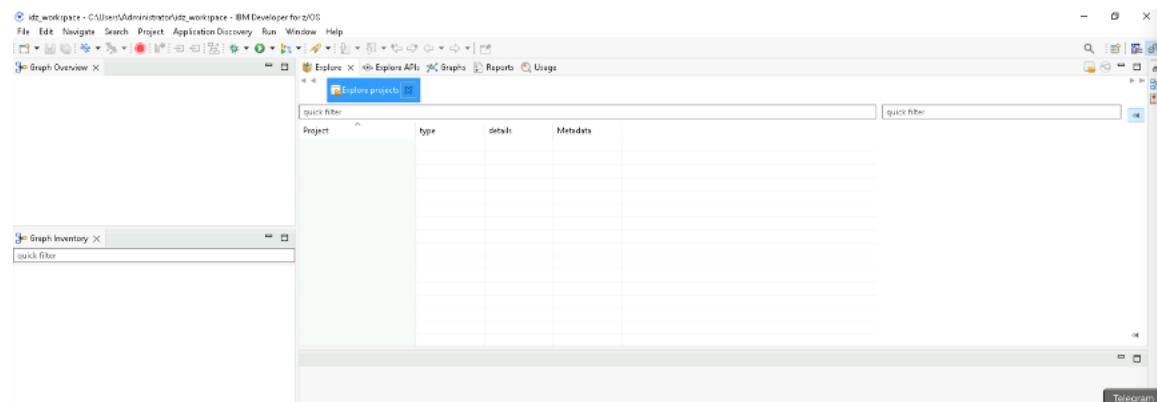


⇒ The default workspace should load automatically. If prompted for a workspace location, enter:

C:\Users\Administrator\idz_workspace

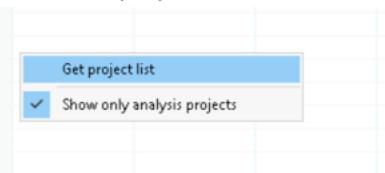
Navigating the Application in ADDI

⇒ Open the Explore view in ADDI.



⇒ If this is your first time launching IDz (or if a new build was completed), refresh the project list:

- Right-click anywhere in the project list panel.
- Select “Get project list.”



- ⇒ Once refreshed, you'll see available projects. For this workshop, we'll be working with the GenApp project.

The screenshot shows the Rational Application Developer interface in the 'Explore' perspective. The top navigation bar includes tabs for 'Explore', 'Explore APIs', 'Graphs', 'Reports', and 'Usage'. Below the navigation bar, there is a toolbar with a 'Explore projects' button. A 'quick filter' input field is present. The main area displays a table with two rows of project information:

Project	type	details	Metadata
GenApp	z/OS	MVS; (ADS, A...)	
rasandbox	z/OS	MVS; (ADS, A...)	

At the bottom right of the table area, there is a button labeled 'Get project list' and a checked checkbox labeled 'Show only analysis projects'.

- ⇒ Double-click on **GenApp** to explore it.

Explore X Explore APIs Graphs Reports Usage

Explore projects Explore GenApp

Resource type: Cobol Program Name filter:

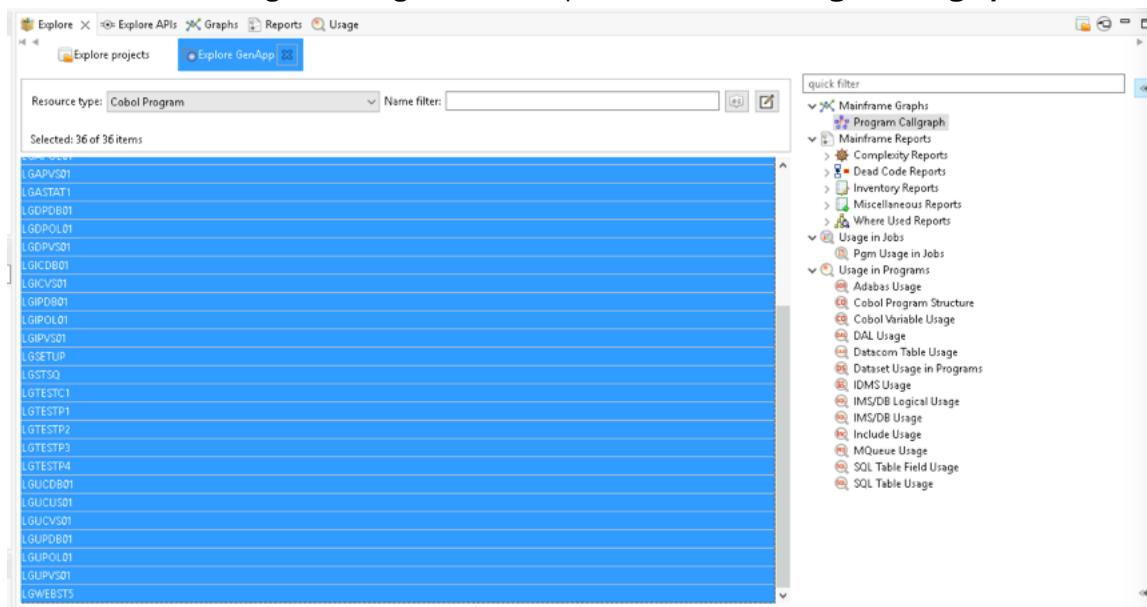
36 items

- AAAAAAA
- COBOLXMP
- CSNBDEC
- CSNBENC
- CSNBGN
- LGACDB01
- LGACDB02
- LGACUS01
- LGACVS01
- LGAPBR01
- LGAPDB01
- LGAPOL01
- LGAPVS01
- LGASTAT1
- LGDPPDB01
- LGDPPOL01
- LGDPVS01
- LGICDB01
- LGICVS01
- LGIPDB01
- LGIPOL01
- LGIPVS01
- LGSETUP
- LGSTSQ

4 Visualize the entire Application Landscape

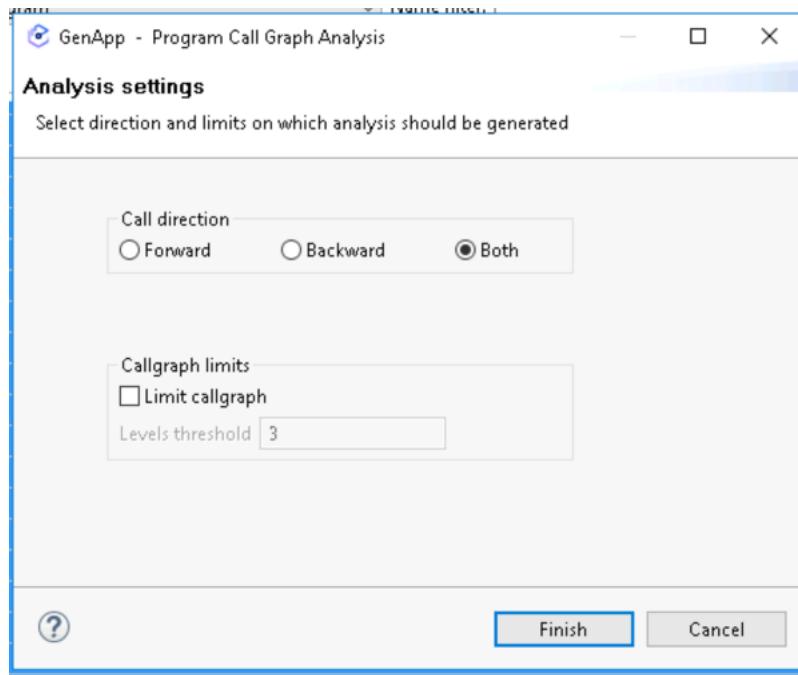
To view the entire application and understand the interconnections:

- ⇒ Select all programs within GenApp:
 - Click the first program.
 - Hold Shift and click the last program in the list.
- ⇒ With all selected, go to the right-hand side panel and click on **Program Callgraph**.

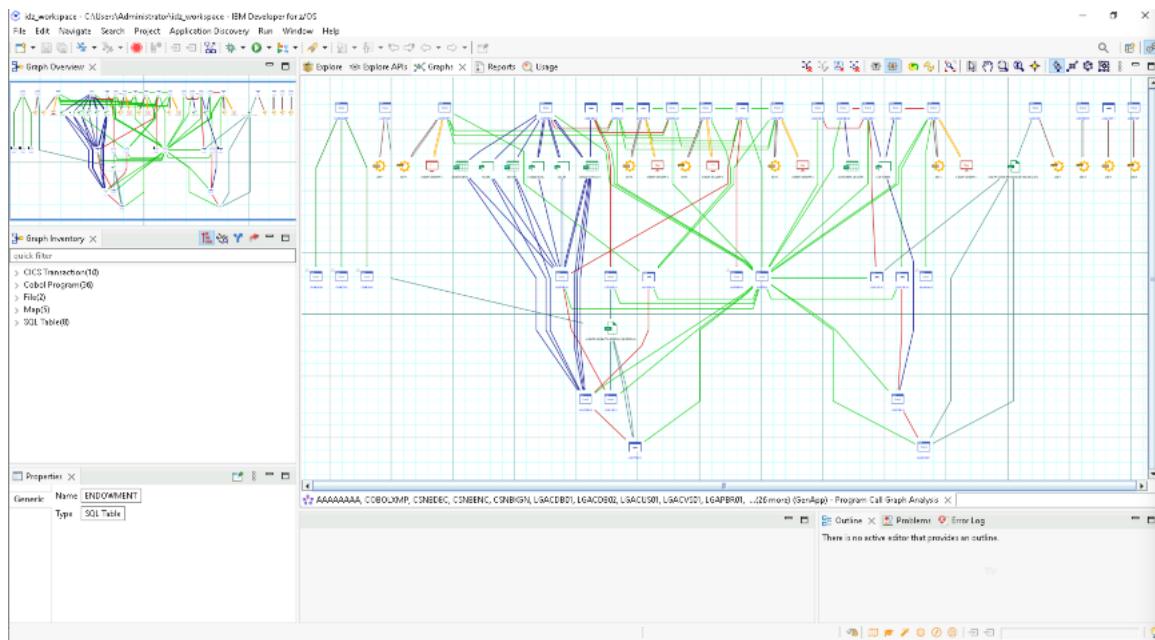


- ⇒ In the Analysis Settings window:
 - Set Call Direction to Both.
 - Uncheck the “Limit callgraph” option. See the illustration below

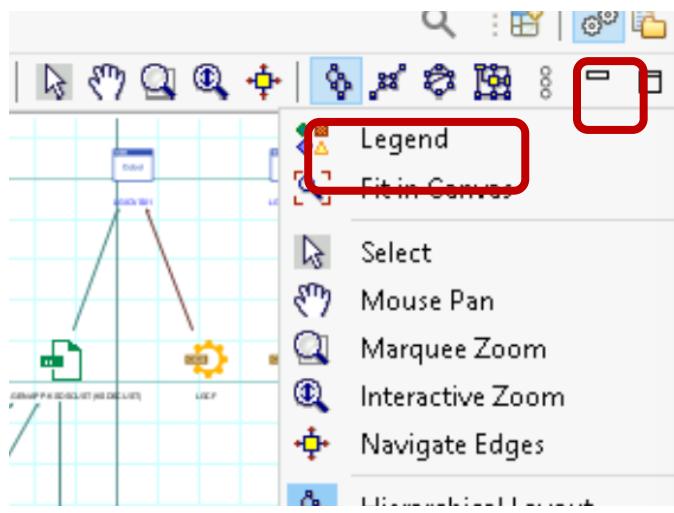
- Click Finish.



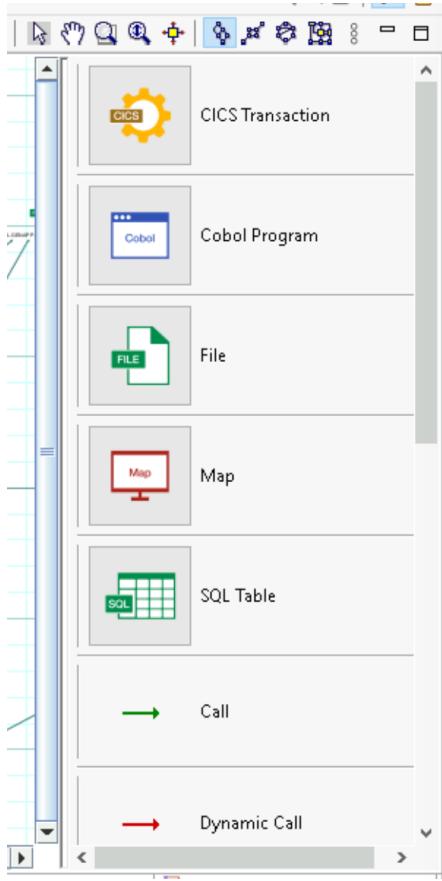
⇒ View the results:



⇒ To understand the graph a little better, there is a legend, on the top right there are 3 vertical dots. Click it and you'll see the option for the legend to show what each arrow and icon represent.



⇒ You'll see something like this:



5 Deep Dive into Specific Applications

Now we're going to look a little deeper into a specific program.

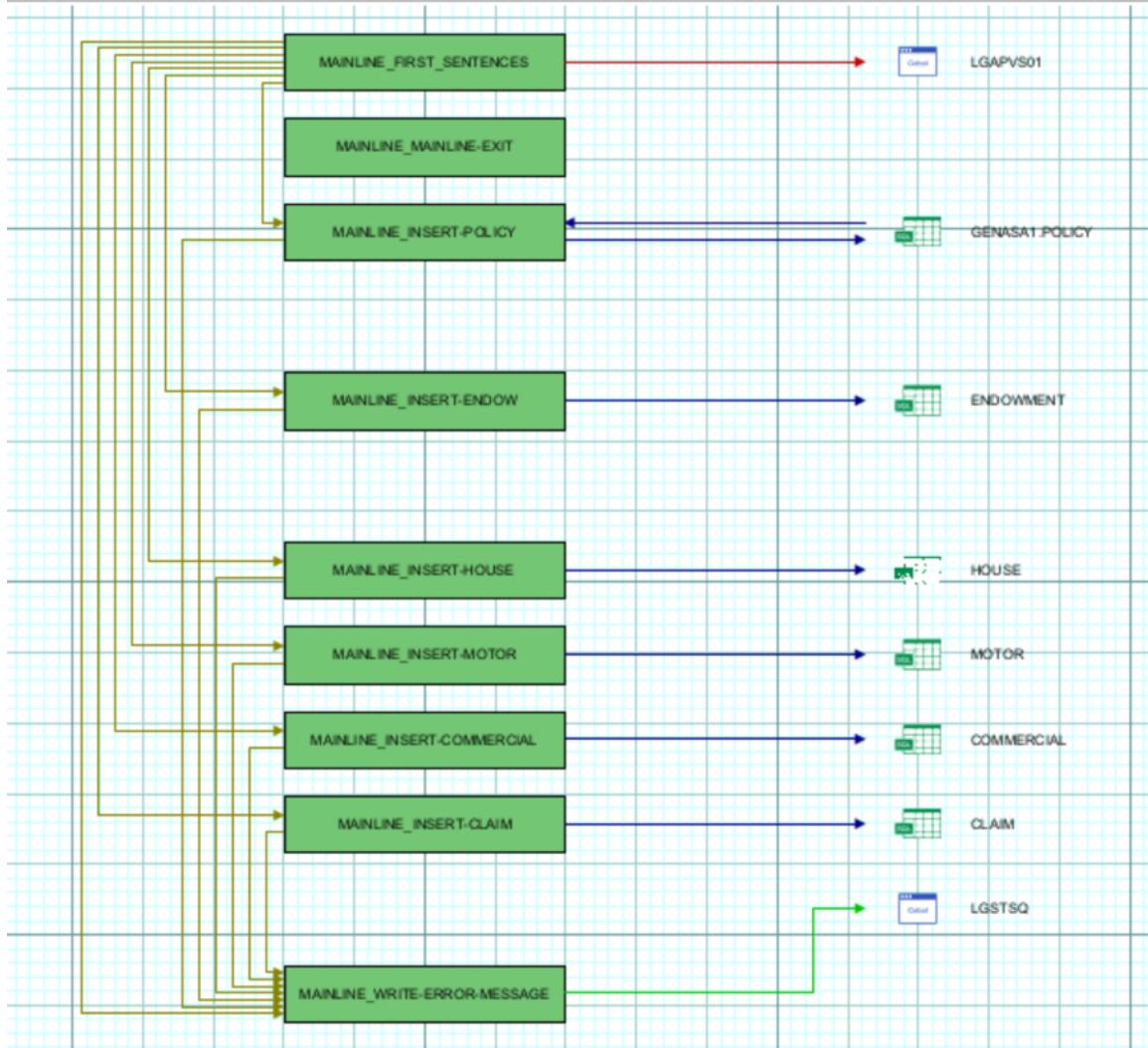
⇒ Navigate back to the **Explore tab** to analyze a specific program in more detail.

We're going to look specifically at **LGAPDB01**, however you're welcome to try any of these steps with any of the programs.

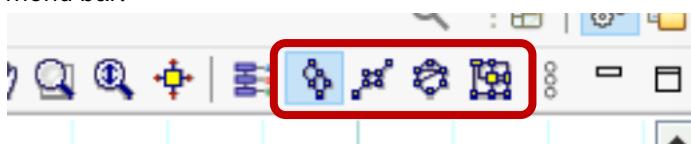
⇒ select the application, then on the right-hand side you can select **Program Flow**

The screenshot shows the 'Explore' interface with the 'Explore GenApp' tab selected. In the main pane, a list of Cobol Programs is displayed, with 'LGAPDB01' selected. On the right, a 'quick filter' sidebar is open, showing various reporting options under categories like Annotations, Mainframe Reports, and Usage. The 'Program Flow' option is visible under the Mainframe Reports section.

- ⇒ A new chart will show up and you can view the flow of the application and any outside calls, again click the three dots on the top right and select Legend to view what each of the lines and icons mean.

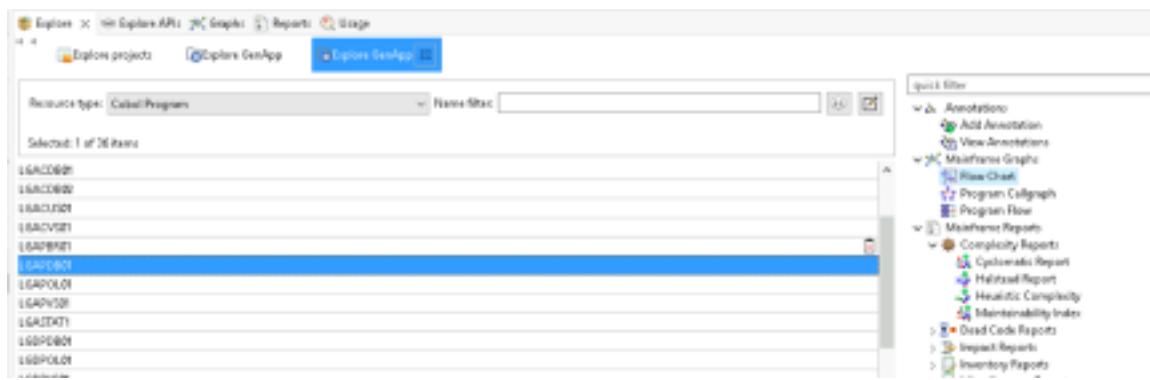


- ⇒ Click on any box and it will open the source code to the start of the specific paragraph.
 ⇒ Play around with different views of the graph by clicking different layout options in the top right of the menu bar.

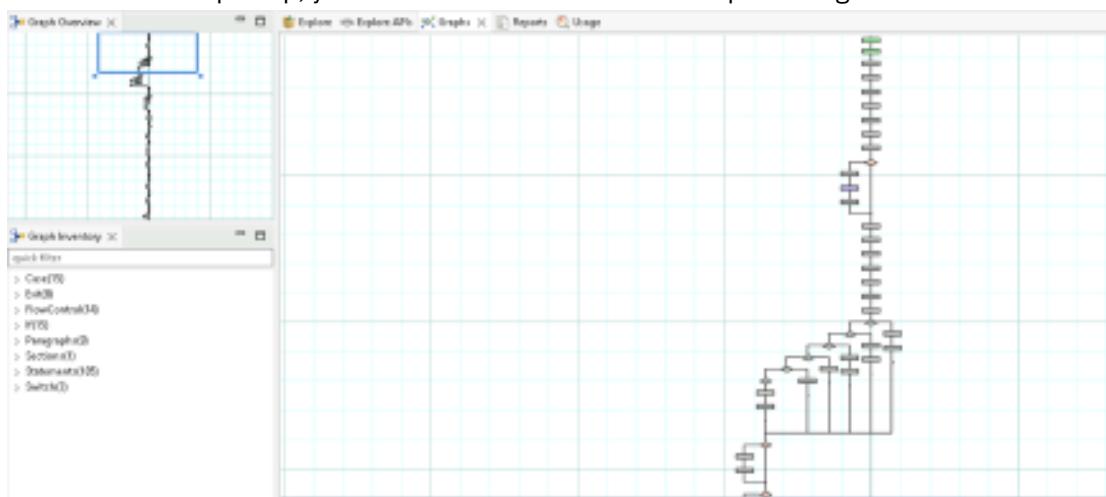


This can be particularly useful when trying to determine which applications are best for refactoring.

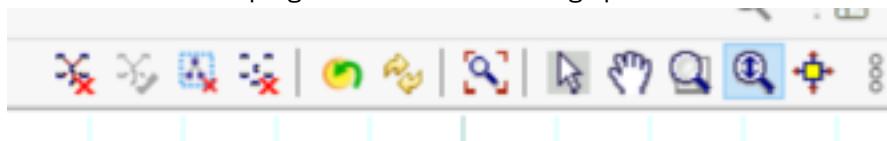
- ⇒ Next, we can look at the Flow chart of the application for a more granular view within the specific paragraphs.



⇒ A new chart will open up, you'll have to scroll in to view the specific logic



⇒ Notice the Graph overview on the top left so you can see where you are within the application and notice the toolbar on the top right for different zooming options

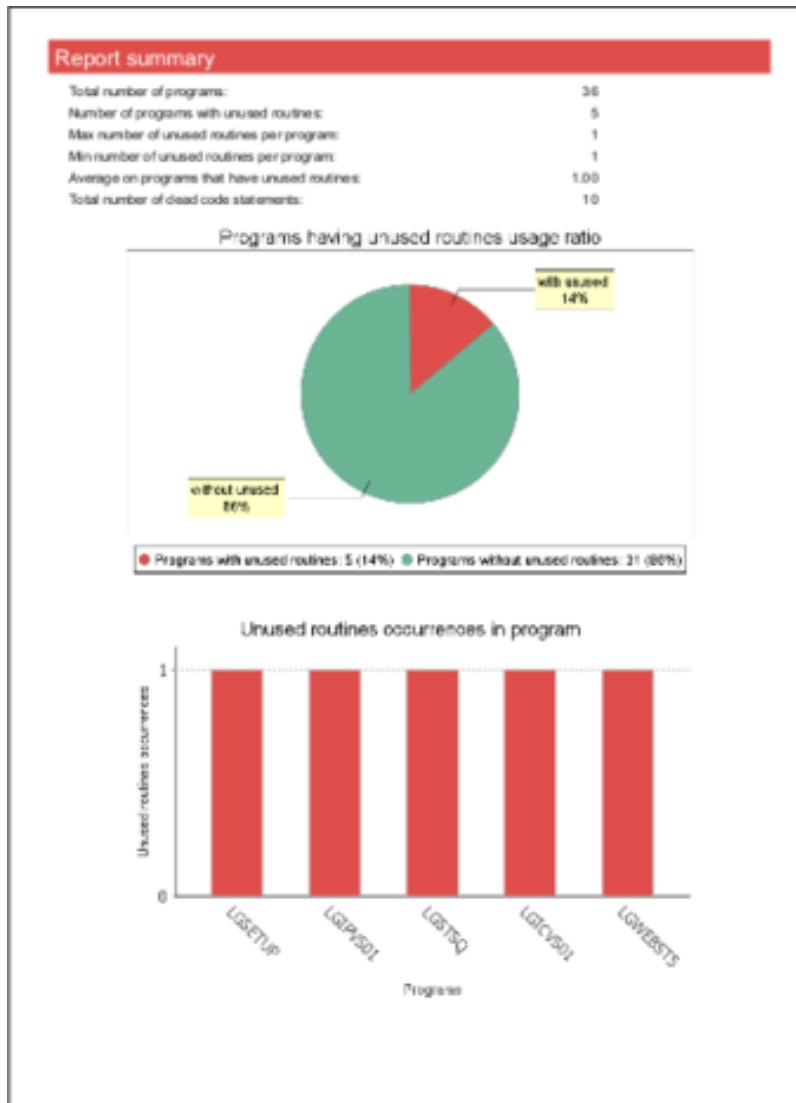


More Reports and views:

- ⇒ We're going to look at different reports the tool can produce. We're going to first look at the *Dead Code Report*.
- ⇒ Select 1 or more of the applications in the Explore window and then select **Dead Code Report > Unused Routines Within Programs**

The screenshot shows the 'Explore' interface with the 'Mainframe Graphs' tab selected. In the center, there is a list of Cobol Programs: LGIPOL01, LGIPVS01, LGSETUP, LGTSQ, LGTESTC1, LGTESTP1, LGTESTP2, LGTESTP3, LGTESTP4, LGUCDB01, LGUCUS01, LGUCVS01, LGUPDB01, LGUPOL01, LGUPVS01, and LGWEBSTS. To the right, a 'quick filter' sidebar is open, showing a hierarchical tree of report categories under 'Mainframe Graphs'.

⇒ Here we can see which programs have unused routines and what they are.



⇒ Continue this for other reports, we recommend looking at:

- Program Flow
- Flow Chart

- Complexity Report
- Dead Code Report

Tip: These tools help identify optimization opportunities, understand logic flow, and assess maintainability.

6 Code explanation – Understand Eclipse client / ADDI and Refactor Assistant (RA)

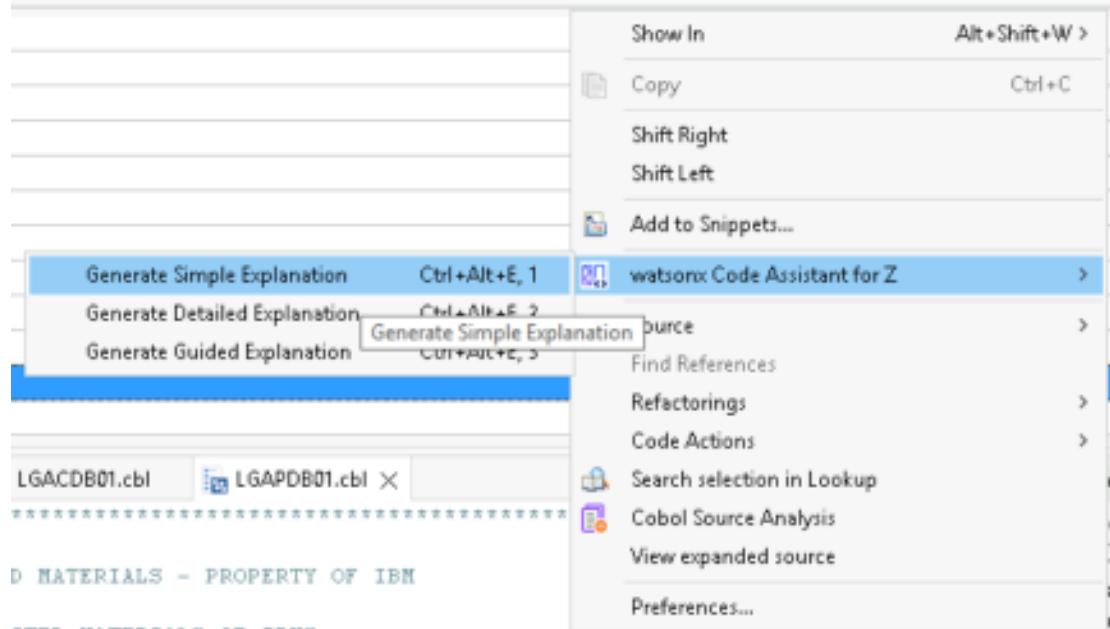
We're going to look at the Code Explanation feature. The code explanation feature is available in both IDz as well as VSCode, we're going to look at IDz first because we've already got it open.

- ⇒ First, we need to open the source code. Let's continue to look at **LGAPDB01** from the explore menu and double click on the program. You should see the source code open in a tab below.

The screenshot shows the Eclipse IDE interface. At the top, there is a toolbar with icons for 'Explore', 'Explore APIs', 'Graphs', 'Reports', and 'Usage'. Below the toolbar, there is a navigation bar with tabs: 'Explore projects', 'Explore GenApp', 'Explore GenApp', and 'Explore GenApp' (the last one is selected). A dropdown menu labeled 'Resource type' is set to 'Cobol Program'. A 'Name filter' input field is also present. Below the navigation bar, a message says 'Selected: 1 of 36 items'. A list of program names is shown, with 'LGAPDB01' highlighted by a blue selection bar. The bottom half of the screen shows the source code editor with three tabs: 'LGDPDB01.cbl', 'LGACDB01.cbl', and 'LGAPDB01.cbl' (which is the active tab). The code in the editor is a COBOL program with various comments and sections like 'LICENSED MATERIALS - PROPERTY OF IBM' and 'ADD Policy'.

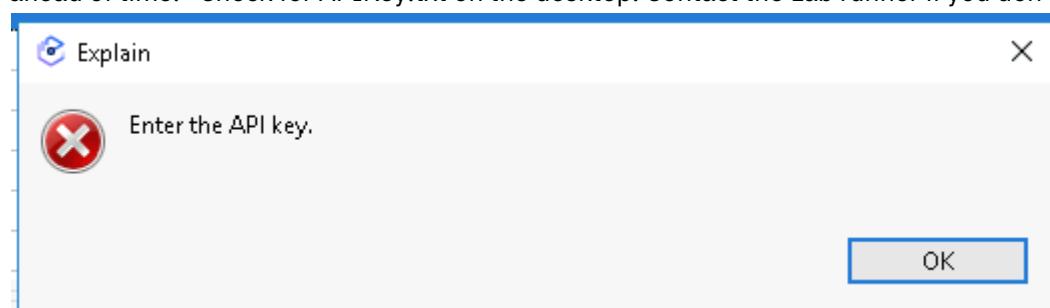
- ⇒ Right click anywhere on the source code and select:
watsonx Code Assistant for Z > Generate Simple Explanation

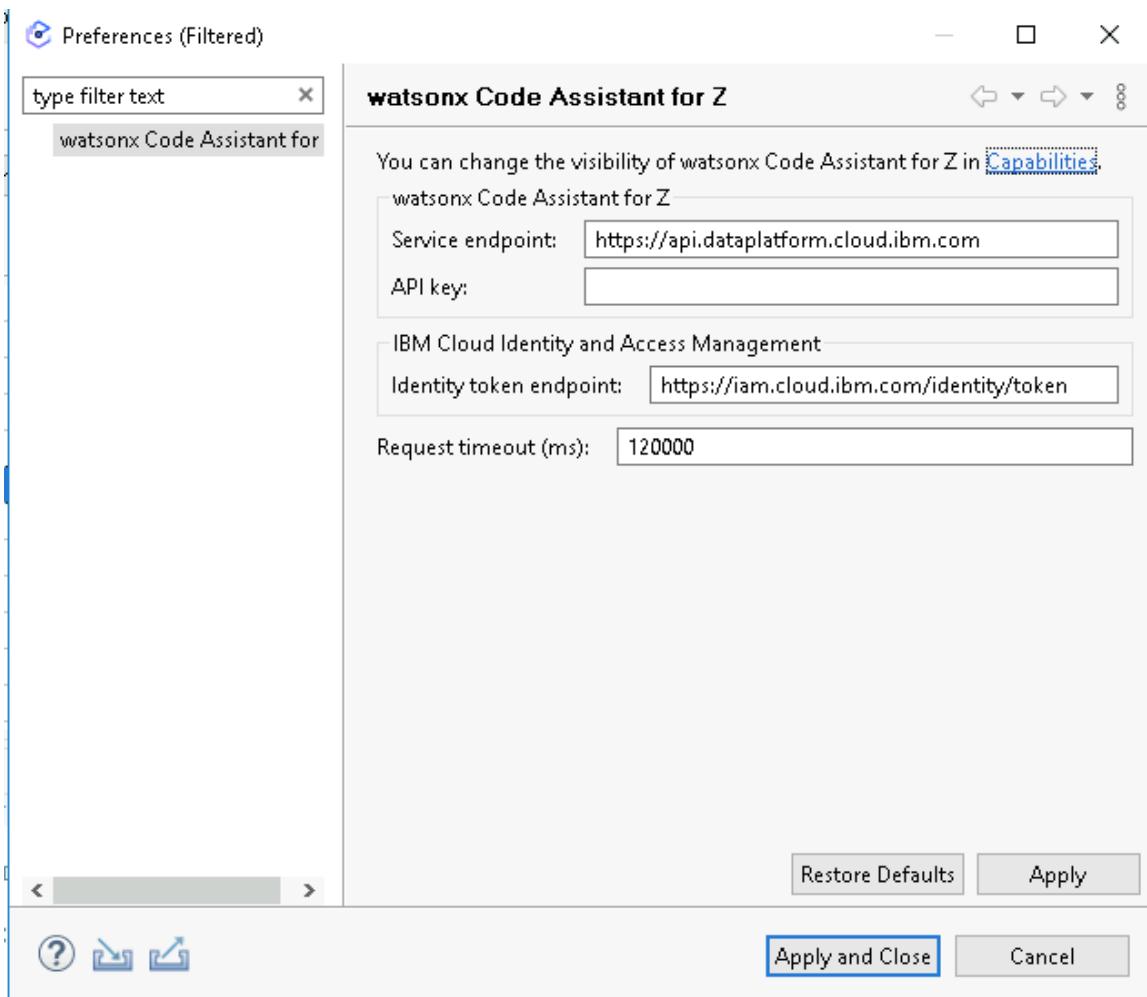
You might be asked to input API Key, input the API key generated in IBM Cloud.



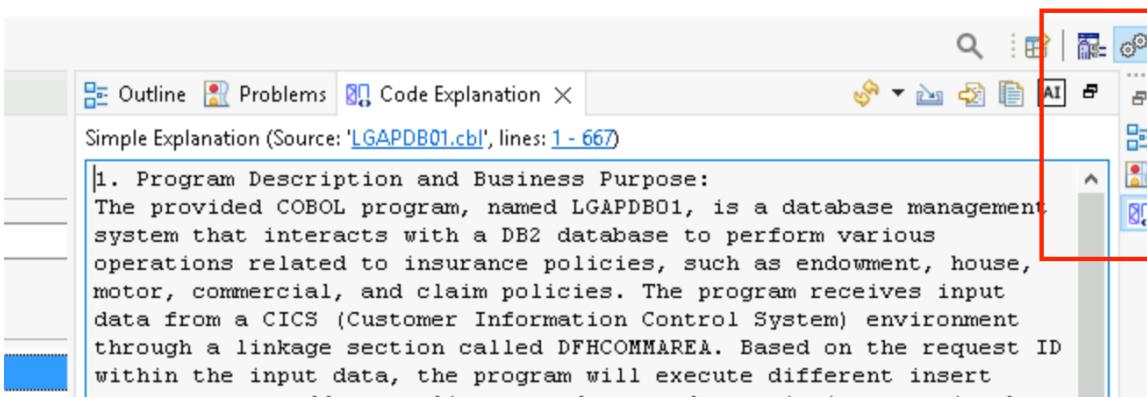
This will send this code to IBM Cloud and will generate a human readable explanation of the code.

- ⇒ You might get an error saying you need to input your API key, this API key should have been sent to you ahead of time. Check for APIKey.txt on the desktop. Contact the Lab runner if you don't have one.





- ⇒ Once you've inputted the API key, try to generate the explanation again
- ⇒ If you don't see the Code Explanation pop up, click the last icon in the right-hand side tab for Code Explanation



⇒ Review this code, and play around with Simple, Detailed and Guided explanations.

Simple Explanation (Source: 'LGAPDB01.chl', lines: 1 - 667)

1. Program Description and Business Purpose:
The provided COBOL program, named LGAPDB01, is a database management system that interacts with a DB2 database to perform various operations related to insurance policies, such as endowment, house, motor, commercial, and claim policies. The program receives input data from a CICS (Customer Information Control System) environment through a linkage section called DFHCOMMAREA. Based on the request ID within the input data, the program will execute different insert statements to add new policy records or update existing ones in the respective tables (POLICY, ENDOWMENT, HOUSE, MOTOR, and COMMERCIAL). The program also handles error messages and returns appropriate return codes based on the success or failure of the database operations.

2. Free-text Summary of Code Logic:
The program begins by initializing working storage variables and setting up the necessary pointers for the input and output data structures. It then checks if a valid commarea (communication area) has been received and evaluates the request ID to determine the type of policy to be processed (endowment, house, motor, commercial, or claim). Depending on the request ID, the program sets the required length of the commarea and performs input validation.

For each policy type, there is a separate paragraph (INSERT-<policytype>) that handles the insertion of a new policy record into the corresponding table. These paragraphs move relevant input data to the appropriate DB2 host variables and execute an INSERT SQL statement to add the new record. After the insert operation, the program retrieves the last changed timestamp for the newly inserted policy number using the IDENTIFY_VAL_LOCAL() function.

In case of any errors during the insert operation (e.g., unique constraint violations), the program sets an error return code and writes an error message using the LGTSQ program. The error message includes the date, time, request ID, and other relevant information about the failed operation. Finally, the program links to another program (LGAPVS01) and returns control to the calling environment using the CICS RETURN command.

In summary, this COBOL program acts as a mediator between the CICS environment and a DB2 database, handling different types of insurance policies and managing their insertion while providing error handling and feedback mechanisms.

⇒ There are some limitations on the size of the code you can send, but that limit is ever increasing.

Try highlight a specific paragraph and only analyze that portion of the code, or if the application isn't too large, you can get an explanation of the entire application.

7 Use Refactor Assistant (RA)

Refactor Assistant is accessed via Visual Studio Code.

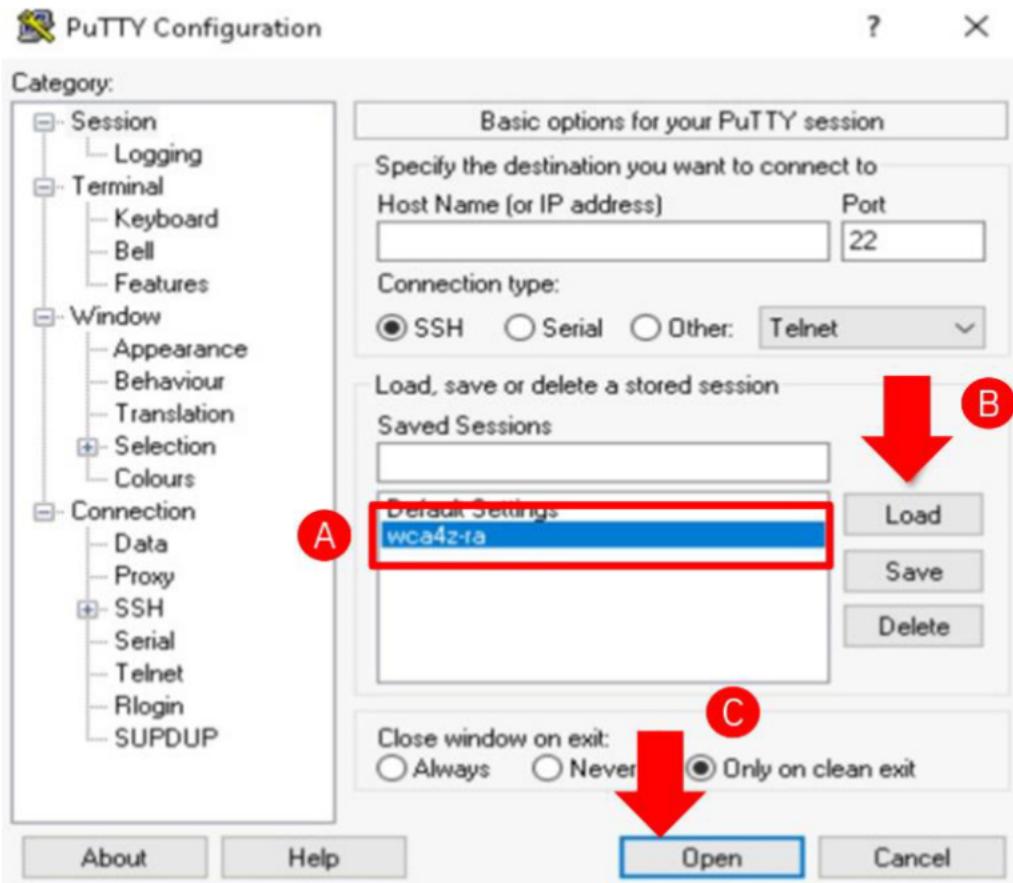
Pre-work Completed:

- The WCA4Z plugin is already installed, we've also pre-done most of the configuration to connect the servers together. To start RA follow these steps

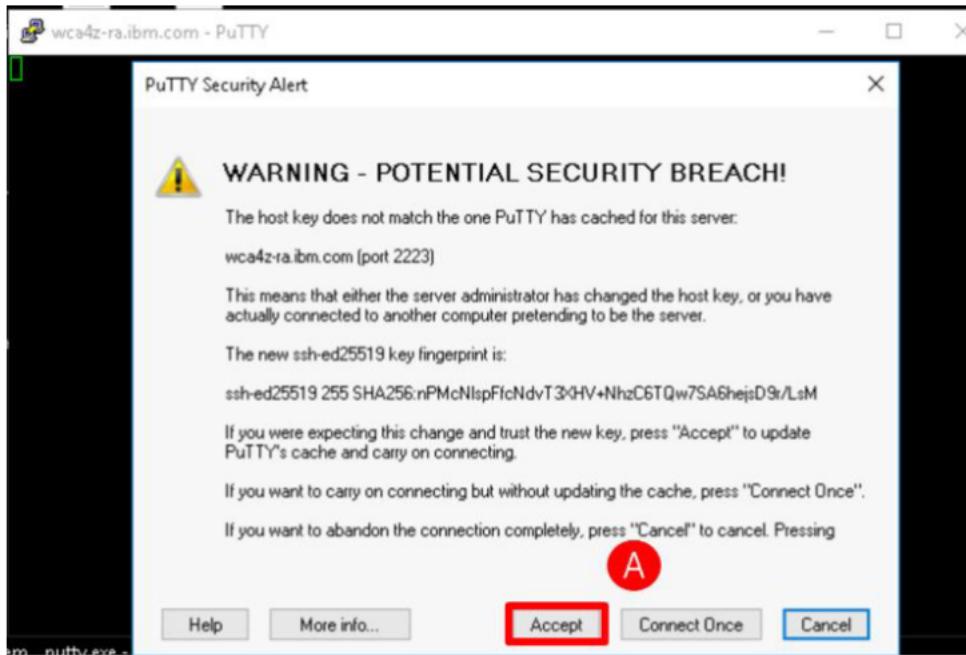
⇒ On the Windows Desktop of the ADDI Server Tech Zone image, open putty.exe (A).



⇒ Within PuTTY, click on the wca4z-ra profile, click Load and then Open.



⇒ If you see this warning, click Accept



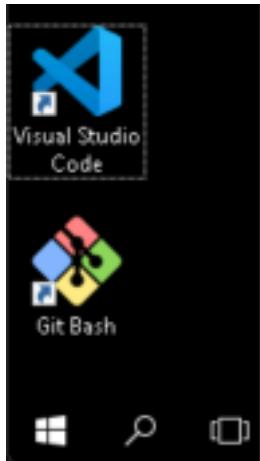
⇒ When the script finishes running you should be the messages up and running.

```
Starting OrientDB...
/home/demo/.config/cni/net.d/sd_network.conflict
ab1f4eb2a94f5bf2995cd78f2cf31a3f68a1ebbeb4c5d3d541767c590c770bdf
Bundling orientdb.crt and orientdb.key into pkcs12 keystore
Restarting OrientDB with SSL configuration
ab1f4eb2a94f5bf2995cd78f2cf31a3f68a1ebbeb4c5d3d541767c590c770bdf
9bb2b95702a39831b4b6f4ea6596d168dd69accdfc5962c5c5e945e1c24ab2e9
Starting IBM Watsonx Code Assistant for Z Refactoring Assistant...
Bundling ad-core-server.crt and ad-core-server.key into pkcs12 keystore
Certificate was added to keystore
Bundling refactoring-assistant.crt and refactoring-assistant.key into pkcs12 key
store
Certificate was added to keystore
00f2d1121990b6fc1d8f6ba86f32a550fc66e6c6bf0826179438d2aa83774d72
e9d6312fb9b4f3745eed3c1c204470414918cb3ff731686d98410665815cf533
a5784b326b72e542b64dea705d9e612d96309abc1655e55f5552e6d93d788fc3
IBM Watsonx Code Assistant for Z Refactoring Assistant started
```

⇒ The pre-work for connecting RA is done, you can proceed to use RA.

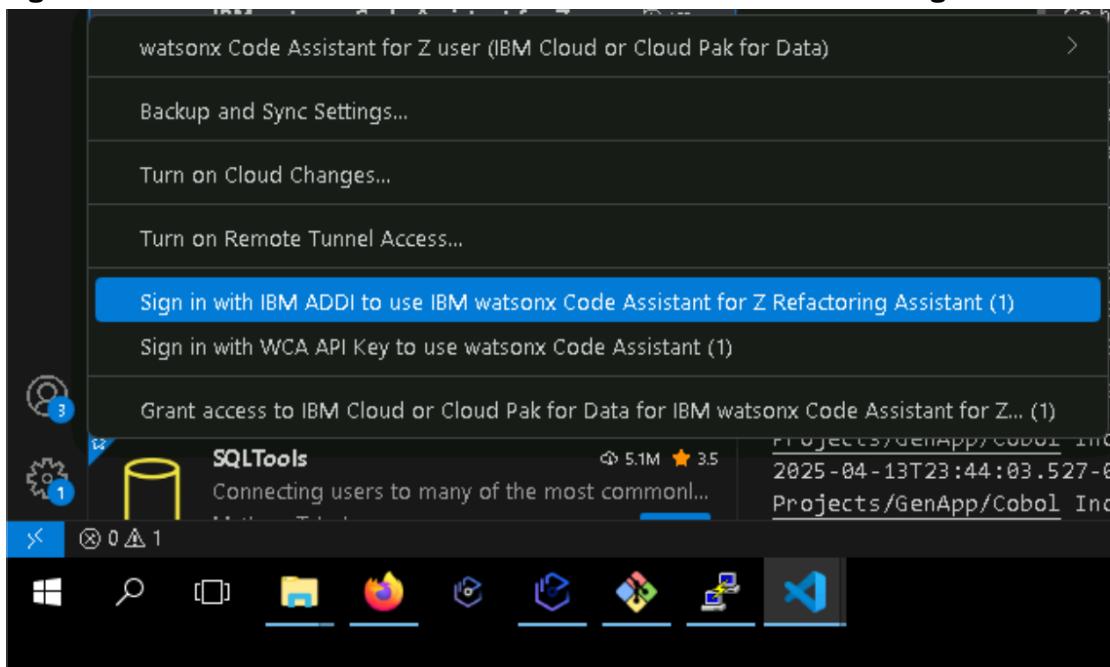
7.1.1 Complete the following steps

- ⇒ Double-click the VS Code icon on the desktop.



- ⇒ In the bottom-left corner, then click:

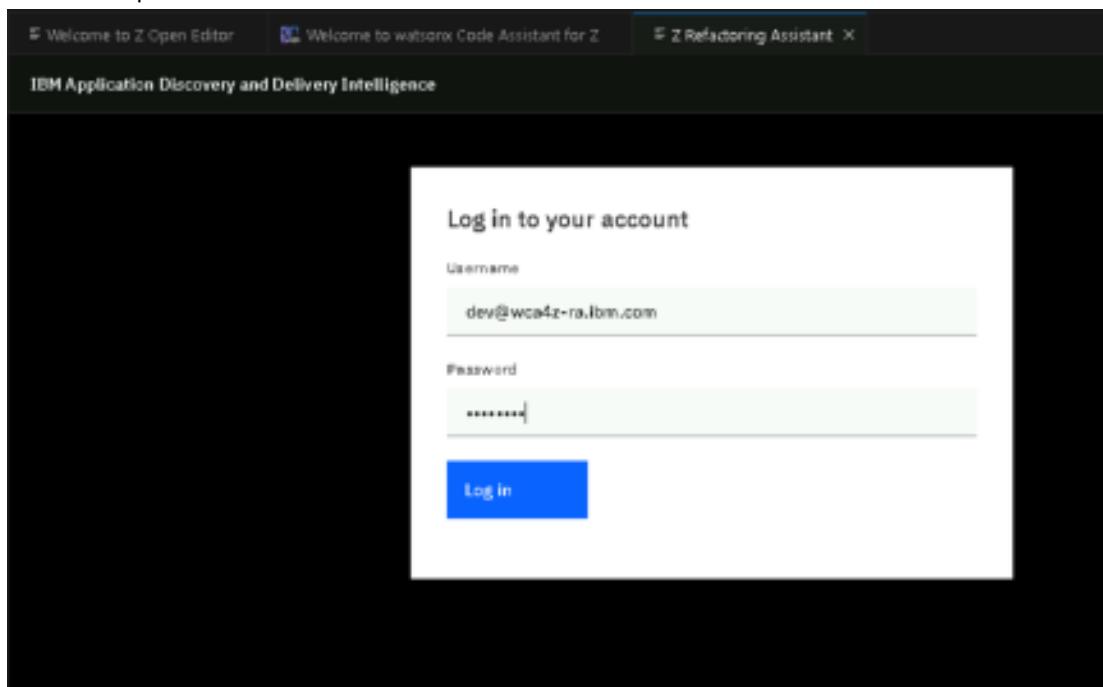
Sign in with IBM ADDI to use IBM watsonx Code Assistant for Z Refactoring Assistant.



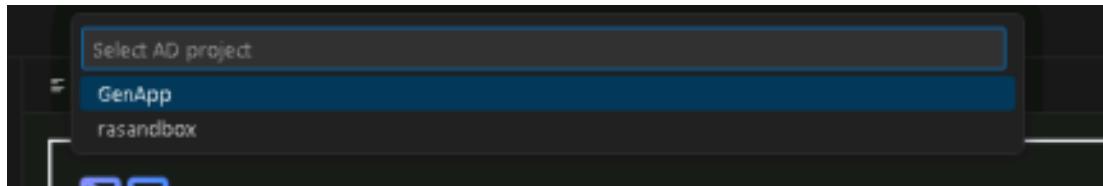
- ⇒ When prompted, use the following credentials:

Username: dev@wca4z-ra.ibm.com

Password: password



- ⇒ From the top menu you will see a drop down to select the project to work with. Select GenApp (This may take 30 seconds to load)



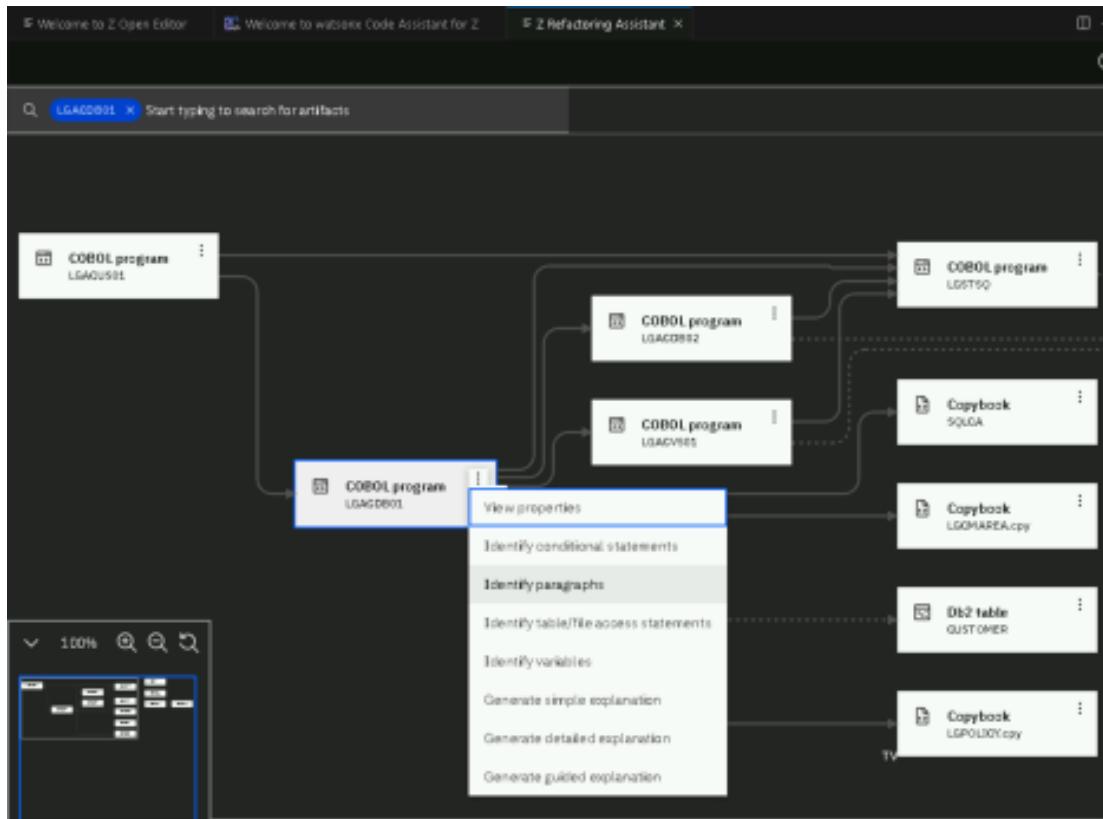
⇒ You should see the screen below, Filter by *Programs* and select **LGACDB01**

The screenshot shows a search interface with a search bar at the top containing the placeholder text "Start typing to search for artifacts". Below the search bar is a dropdown menu labeled "Artifact type" with the option "Programs (36)" selected. The main area displays a list of 36 results, each consisting of a name and a category. The names listed are: AAAAAAAA, COBOLXMP, CSNBDEC, CSNBENC, CSNBKGN, LGACDB01, and LGACDB02. All results are categorized as "Program". At the bottom of the list, it says "36 of 36 results" and there is a "Clear results" button.

Name	Type
AAAAAAA	Program
COBOLXMP	Program
CSNBDEC	Program
CSNBENC	Program
CSNBKGN	Program
LGACDB01	Program
LGACDB02	Program

⇒ Find the COBOL program LGACDB01, click on the 3 vertical dots and select:

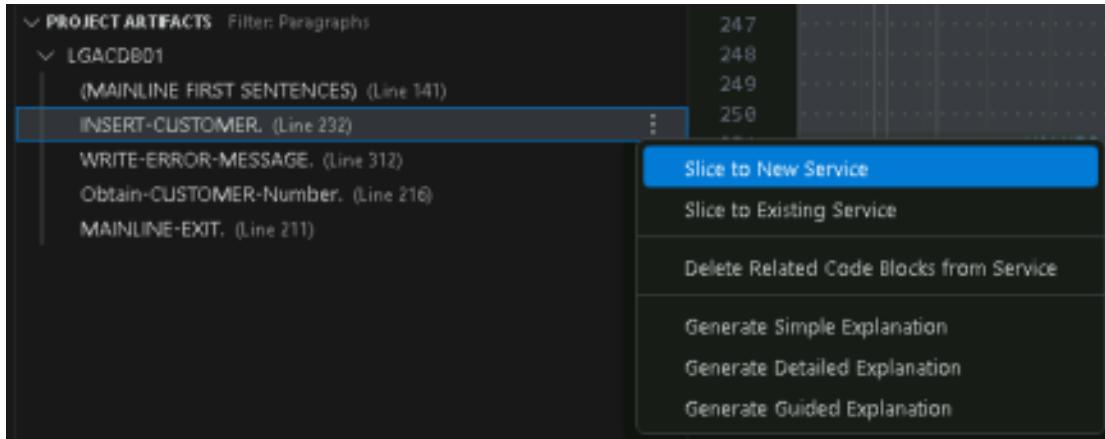
Identify paragraphs.



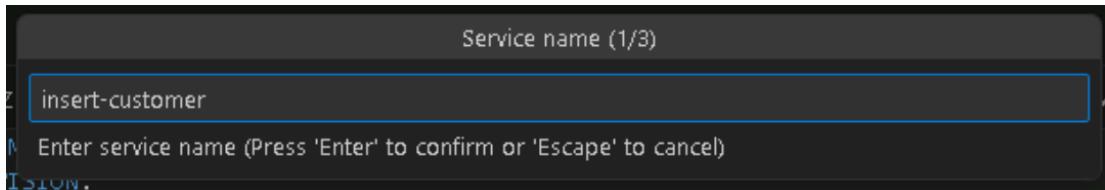
⇒ On the left side you should see a list of all the Paragraphs for the application.

The screenshot shows the "PROJECT ARTIFACTS" view with a filter set to "Paragraphs". The tree view on the left shows a node for "LGACDB01". Underneath it, a list of paragraphs is displayed: "(MAINLINE FIRST SENTENCES) (Line 141)", "INSERT-CUSTOMER. (Line 232)" (which is currently selected and highlighted in blue), "WRITE-ERROR-MESSAGE. (Line 312)", "Obtain-CUSTOMER-Number. (Line 216)", and "MAINLINE-EXIT. (Line 211)".

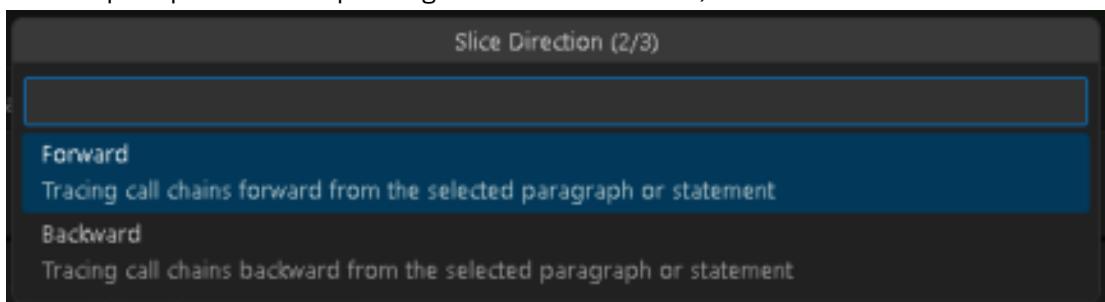
- ⇒ We're going to isolate **INSERT-CUSTOMER** into its own standalone service.
- ⇒ Select the three dots on the right side and select **Slice to New Service**



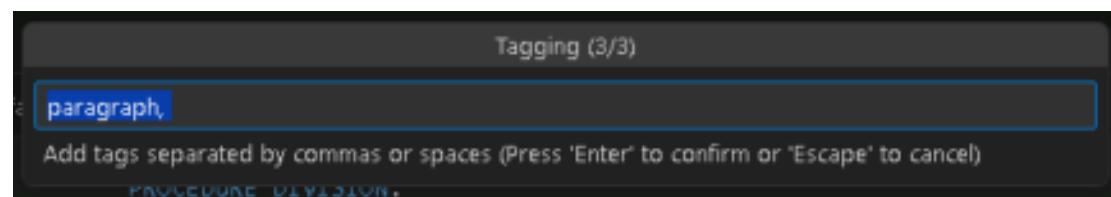
- ⇒ On the top you will see a text bar open asking to **Enter service name**, you can name it whatever you want, but we'll name it *insert-customer*



- ⇒ Another prompt will come up asking for the Slice Direction, select **Forward** and then hit Enter.

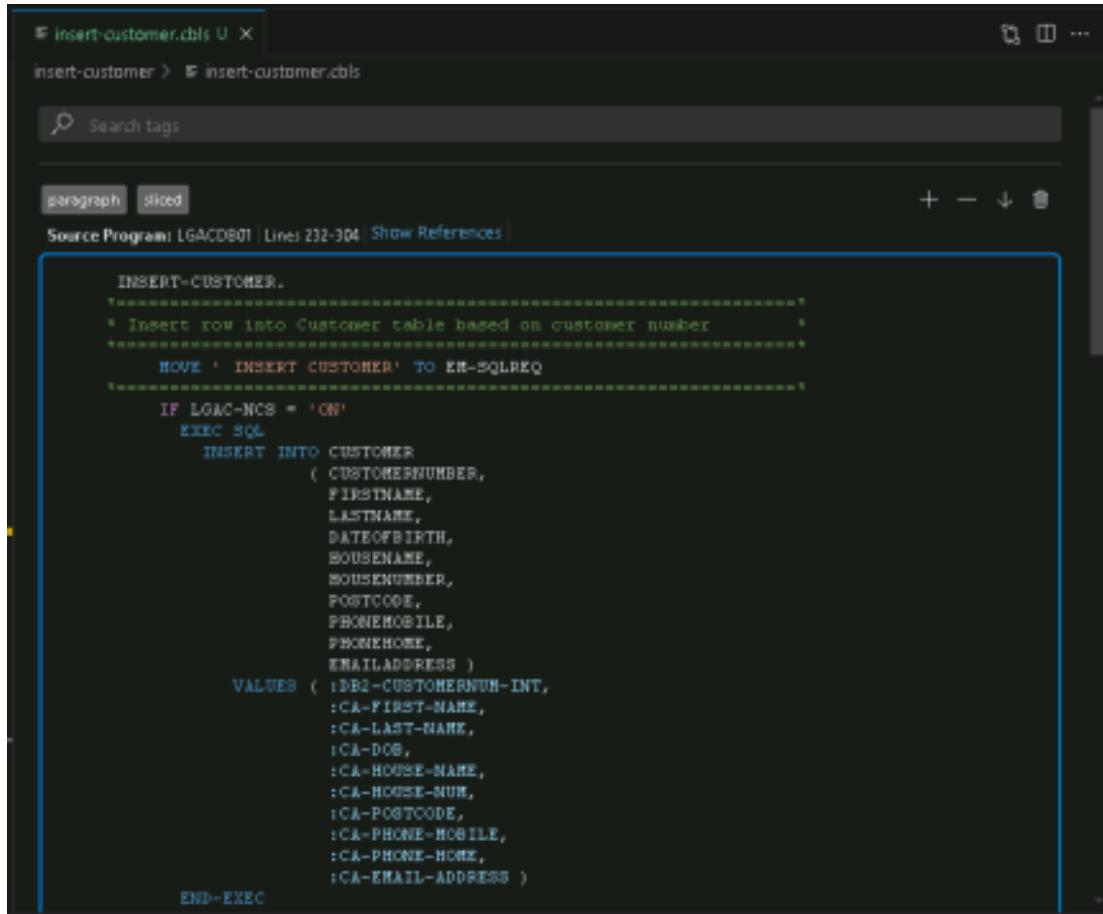


- ⇒ Lastly it will ask for Tagging, leave the default value **paragraph**, and hit Enter.



- ⇒ On the right side you'll see a new tab open called **insert-customer.cbls**.
.cbls is a cobol slice. This contains the paragraph you selected, but if you scroll down you'll notice it also included all the paragraphs in which this paragraph calls. It does this recursively. In this case it's only 2

paragraphs, but in larger applications this can be a much larger slice



The screenshot shows the Watsonx Code Assistant for Z interface. At the top, there's a file navigation bar with 'insert-customer.cbls' selected. Below it is a search bar labeled 'Search tags'. Underneath, there are two buttons: 'paragraph' (which is highlighted) and 'sliced'. A status bar at the bottom indicates 'Source Program: LGACDB01 | Lines: 232-304 | Show References'. The main area displays a COBOL program code block:

```
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 ( :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
```

⇒ On the left side with the File explorer, you will see the newly created file structure. This cobol slice on its own isn't very useful, but if we right click on the cobol slice we can create a full cobol program.

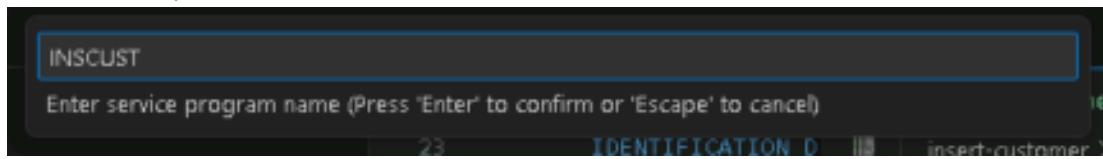
Right click on the Cobol slice and select:

Watsonx Code Assistant for Z > Generate Service Code

The screenshot shows the SAP ABAP code editor with the file 'insert-customer.cbls' open. The code is a COBOL program that inserts a row into the CUSTOMER table. It includes logic to check if the LGAC-NCS parameter is ON and to move the 'INSERT CUSTOMER' statement to the ER-SQLREQ queue.

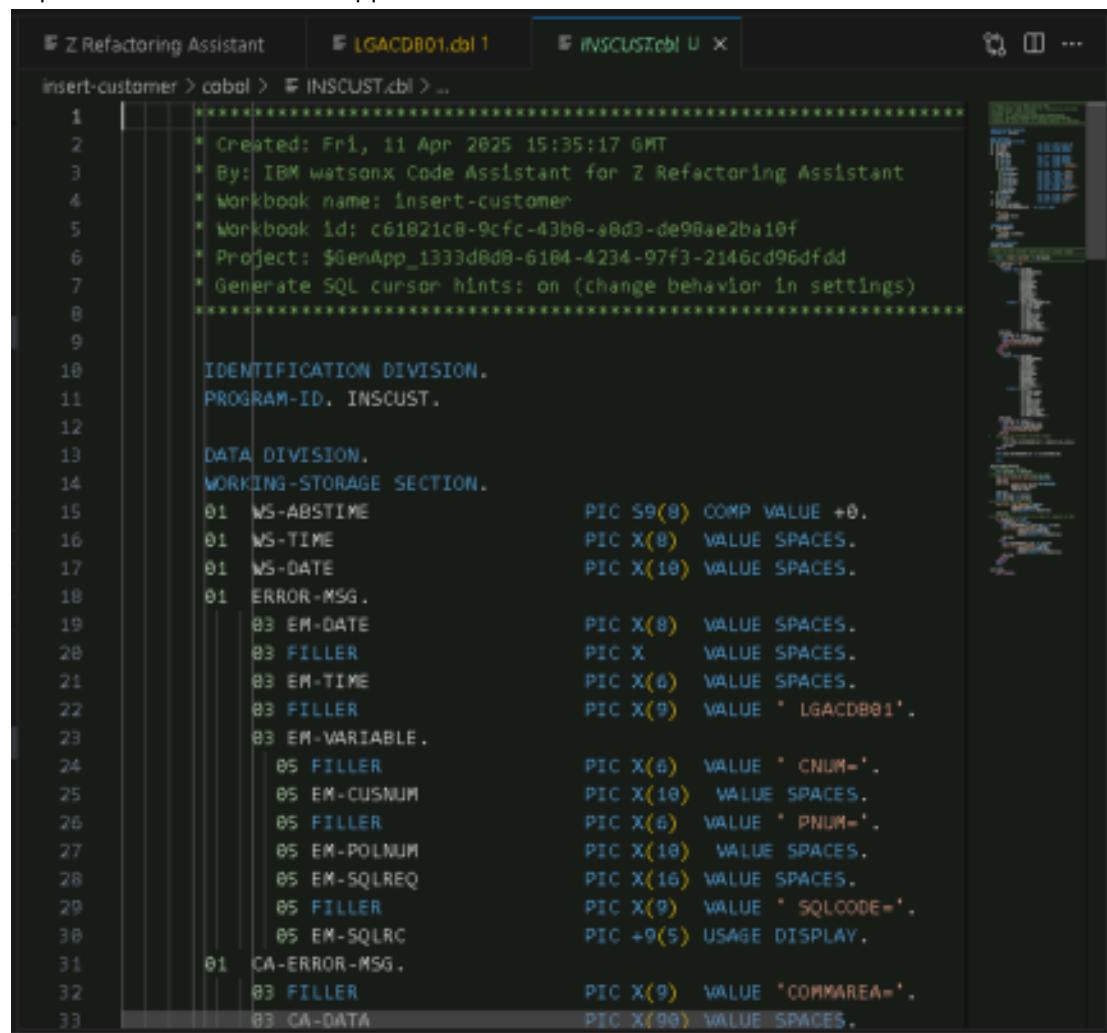
```
INSERT=CUSTOMER.  
-----  
* Insert row into Customer table based on customer number  
-----  
MOVE 'INSERT CUSTOMER' TO ER-SQLREQ  
-----  
IF LGAC-NCS = 'ON'  
  EXEC SQL  
    INSERT INTO CUSTOMER  
      ( CUSTOMERNUMBER,  
        FIRSTNAME,  
        LASTNAME,  
        DATEOFBIRTH,  
        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
```

- ⇒ You will be prompted to Enter service program name. This can only be 8 characters long. Let's use INSCUST and press Enter



This will take about 30 seconds and will generate a stand along program with Working storage and

dependencies. Review this application.



```
1 ****
2 * Created: Fri, 11 Apr 2025 15:35:17 GMT
3 * By: IBM Watsonx Code Assistant for Z Refactoring Assistant
4 * Workbook name: insert-customer
5 * Workbook id: c61821c8-9cf8-48d3-de98ae2ba10f
6 * Project: $genApp_1333d8d8-6104-4234-97f3-2146cd96dfdd
7 * Generate SQL cursor hints: on (change behavior in settings)
8 ****
9
10 IDENTIFICATION DIVISION.
11 PROGRAM-ID. INSCUST.
12
13 DATA DIVISION.
14 WORKING-STORAGE SECTION.
15 01 WS-ABSTIME          PIC S9(8) COMP VALUE +0.
16 01 WS-TIME             PIC X(8)  VALUE SPACES.
17 01 WS-DATE             PIC X(10) VALUE SPACES.
18 01 ERROR-MSG.
19     03 EM-DATE          PIC X(8)  VALUE SPACES.
20     03 FILLER            PIC X   VALUE SPACES.
21     03 EM-TIME           PIC X(6)  VALUE SPACES.
22     03 FILLER            PIC X(9)  VALUE " LGACDB01".
23     03 EM-VARIABLE.
24         05 FILLER          PIC X(6)  VALUE " CNUM=".
25         05 EM-CUSNUM        PIC X(10) VALUE SPACES.
26         05 FILLER          PIC X(6)  VALUE " PNUM=".
27         05 EM-POLNUM        PIC X(10) VALUE SPACES.
28         05 EM-SQLREQ        PIC X(16) VALUE SPACES.
29         05 FILLER          PIC X(9)  VALUE " SQLCODE=".
30         05 EM-SQLRC          PIC +9(5) USAGE DISPLAY.
31     01 CA-ERROR-MSG.
32     03 FILLER            PIC X(9)  VALUE " COMMAREA=".
33     03 CA-DATA            PIC X(90) VALUE SPACES.
```

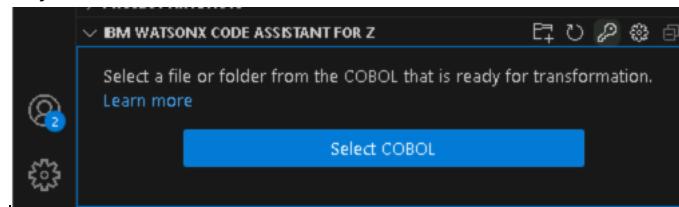
Congratulations, you've refactored your first application!

Code Explanation in VS Code

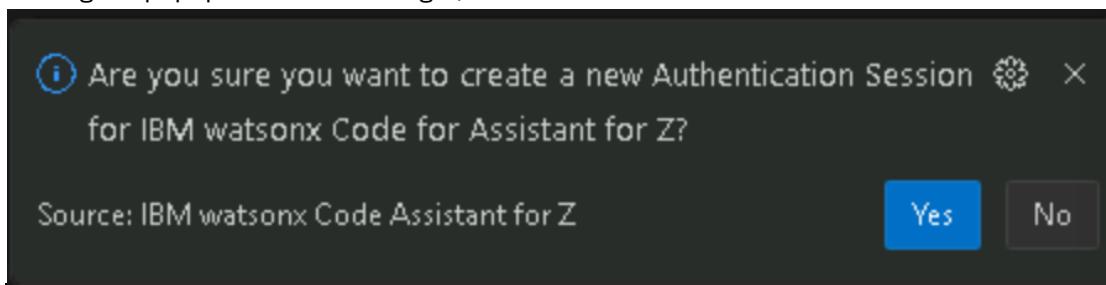
Pre-work:

We need to add the API key which the instructor should have shared with you

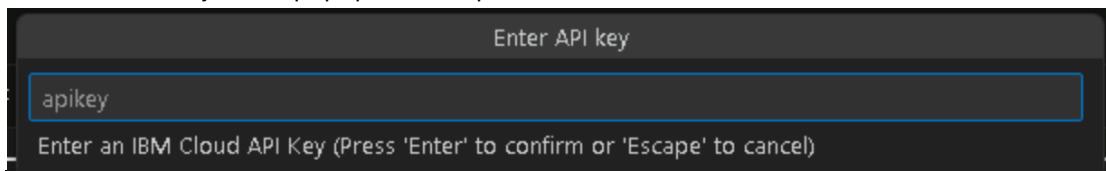
- ⇒ In the bottom left, there is a section called IBM WATSONX CODE ASSISTANT FOR Z, and there is a small key icon. Click it



⇒ You'll get a popup on the bottom right, click Yes



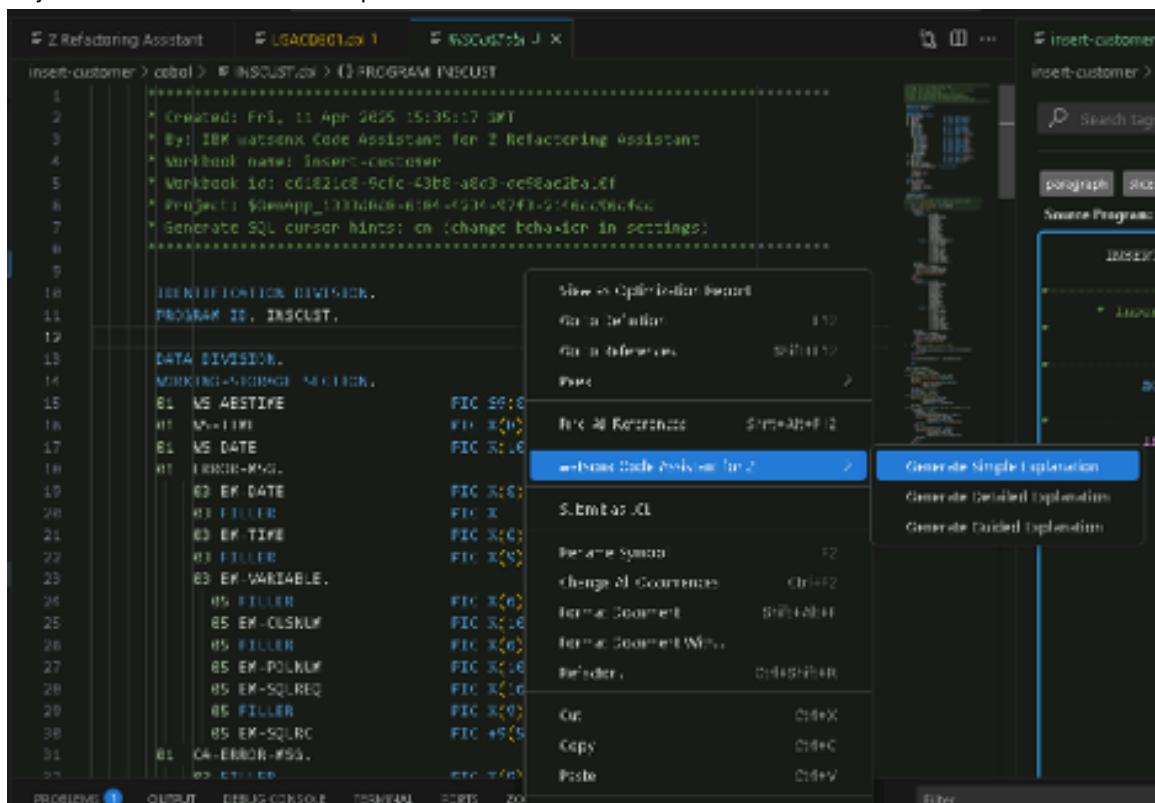
⇒ Enter the API key in the popup at the top



⇒ You're all set to use the Watsonx code explanation

While we're here, you can also use explain in VS Code too!

⇒ When you have the cobol code open, either the original, or the new refactored code, you can right click anywhere and see the same options we saw in IDz



Again, play around with different options for viewing the explanation. Try to view the Simple, Detailed and Guided Explanations. Try explaining the entire program and just a paragraph, or even just a section of code by highlighting that code and selecting the level of explanation you want to see.

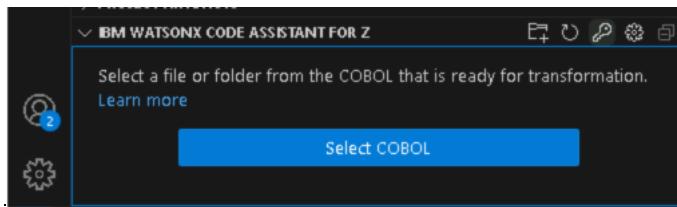
8 Transformation – Converting Cobol to Java

If you didn't complete this in the Code explanation step above, then complete it now. If this was already added, you can skip ahead.

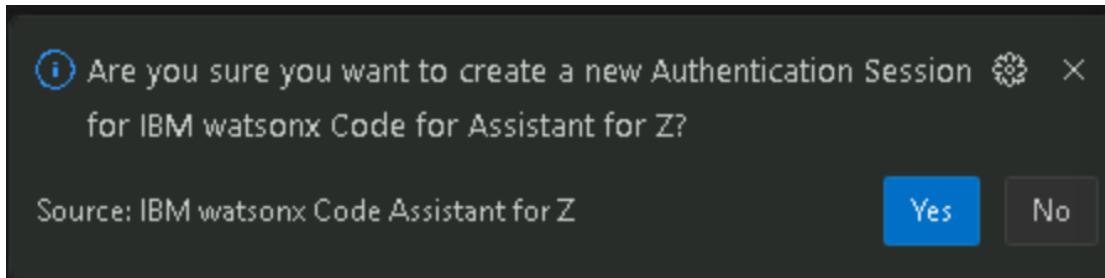
Pre-work:

~~We need to add the API key which the instructor should have shared with you~~

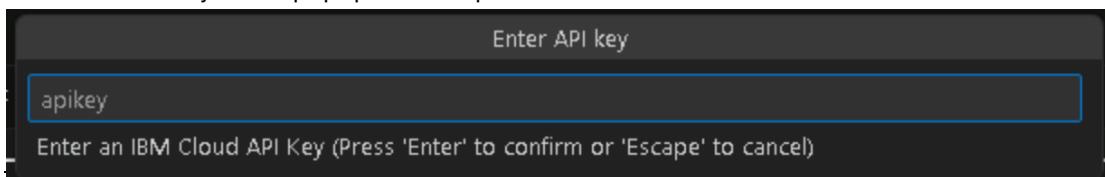
⇒ In the bottom left, there is a section called ~~IBM WATSONX CODE ASSISTANT FOR Z~~, and there is a small key icon. Click it



⇒ You'll get a popup on the bottom right, click Yes



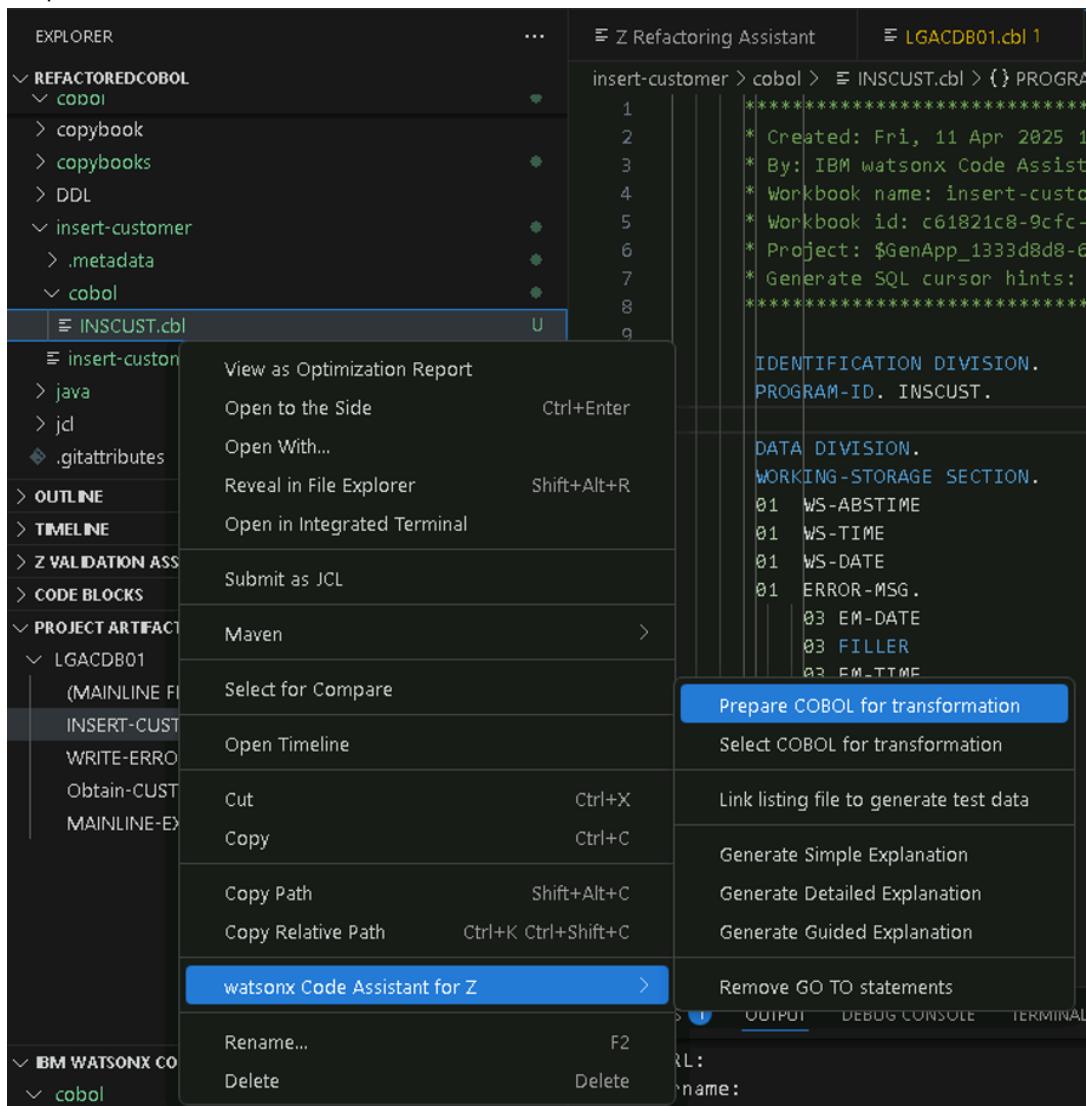
⇒ Enter the API key in the popup at the top



⇒ You're all set to use the Watsonx code explanation

Now we're going to work on transforming a portion of the code from COBOL to Java. The ideal scenario for migration is to isolate a service from the application and transform that to Java, so we're going to work with the refactored piece of code from above.

- ⇒ Right click on the newly created application INSCUST.cbl and select watsonx Code Assistant for Z > Prepare COBOL for transformation



- ⇒ This will sync the code to IBM cloud database. Once this is complete, you can right click the same application again and select:

watsonx Code Assistant for Z > Select COBOL for transformation

The screenshot shows the Watsonx Code Assistant for Z interface. A context menu is open over a COBOL source code file named `INSCUST.cbl`. The menu items include:

- View as Optimization Report
- Open to the Side Ctrl+Enter
- Open With...
- Reveal in File Explorer Shift+Alt+R
- Open in Integrated Terminal
- Submit as JCL
- Maven >
- Select for Compare
- Open Timeline
- Cut Ctrl+X
- Copy Ctrl+C
- Copy Path Shift+Alt+C
- Copy Relative Path Ctrl+K Ctrl+Shift+C
- watsonx Code Assistant for Z >
- Rename... F2
- Delete Delete

Below the main menu, there is a secondary dropdown menu with the following options:

- Prepare COBOL for transformation
- Select COBOL for transformation
- Link listing file to generate test data
- Generate Simple Explanation
- Generate Detailed Explanation
- Generate Guided Explanation
- Remove GO TO statements

The COBOL source code visible in the editor window is:

```
      35          01 DB2-OUT-INTGERS.
      36          03 DB2-CUSTOMERNUM-IN
      40          END-EXEC.

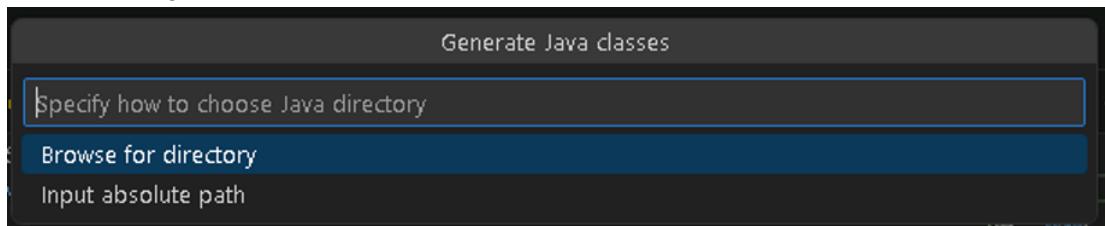
      41          LINKAGE SECTION.
      42          01 DFHCOMMAREA.
      43          EXEC SQL
      44              INCLUDE LGCMAREA
      45          END-EXEC.

      46          PROCEDURE DIVISION.
      47          INSERT-CUSTOMER.
      48          *-----*
      49          * Insert row into Customer
      50          *-----*
      51          MOVE ' INSERT CUSTOMER'
      52          IF LGAC-NCS = 'ON'
      53          EXEC SQL
      54              INSERT INTO CUSTOMER
      55                  ( CUSTID
      56                  FIRSTNAME
      57                  LASTNAME
      58                  DATEOFBIRTH
      59                  HOUSENUMBER
      60                  HOUSENAME
      61                  POSTCODE
      62          ) VALUES
      63          ( LGAC-CUSTID
      64          , LGAC-FNAME
      65          , LGAC-LNAME
      66          , LGAC-BIRTHDATE
      67          , LGAC-HOUSENO
      68          , LGAC-HOUSENAME
      69          , LGAC-POSTCODE
      70          )
      71          END-EXEC.
```

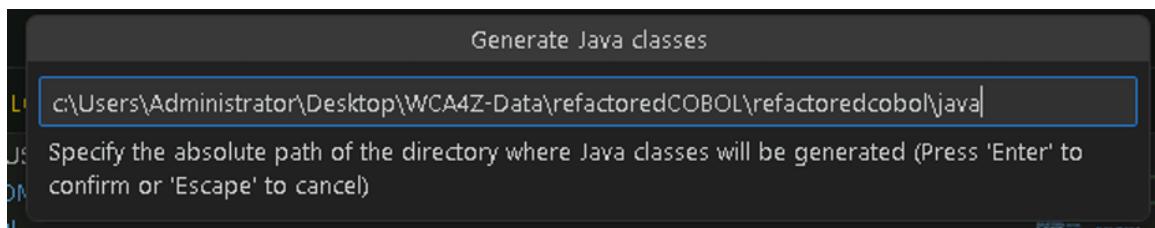
⇒ You'll see the application selected on the bottom left. Select the application and select **Generate Java classes**.

The screenshot shows the IBM WatsonX Code Assistant for Z interface. On the left, there's a sidebar with 'CODE BLOCKS' and 'PROJECT ARTIFACTS' sections. Under 'PROJECT ARTIFACTS', 'LGACDB01' is expanded, showing several COBOL programs: '(MAINLINE FIRST SENTENCES)' (Line 141), 'INSERT-CUSTOMER.' (Line 232), 'WRITE-ERROR-MESSAGE.' (Line 312), 'Obtain-CUSTOMER-Number.' (Line 216), and 'MAINLINE-EXIT.' (Line 211). Below this is the 'IBM WATSONX CODE ASSISTANT FOR Z' section, which is also expanded, showing a 'insert-customer\ cobol' folder containing 'INSCUST.cbl'. A context menu is open over 'INSCUST.cbl', with options: 'Reveal in explorer', 'Open file location', 'Remove', and 'Transform to Java'. At the bottom of the interface, there are tabs for 'PROBLEMS' (29) and 'OUTPUT', and a log area showing timestamped messages.

- ⇒ On the top of the window, you'll see the option for where to put the resulting Java Classes. Select **Input absolute path**



- ⇒ This will populate with the current directory, add **/java** to the end of the path



- ⇒ You'll see a new tab open with a list of all the classes that will be generated. Here you can change the name of the Java classes to be generated, though we recommend taking the suggested names and refactor after that fact if desired.
- ⇒ Select Incremental generation

Code transformation X ⋮

Code transformation for INSCUST.cbl

Select a generation approach for remaining transformation

Incremental generation

Gradually generate Java starting with `insertCustomer`. Generating this method creates dependent data classes and adds stub methods to the main class. Repeat this process for all stub methods.

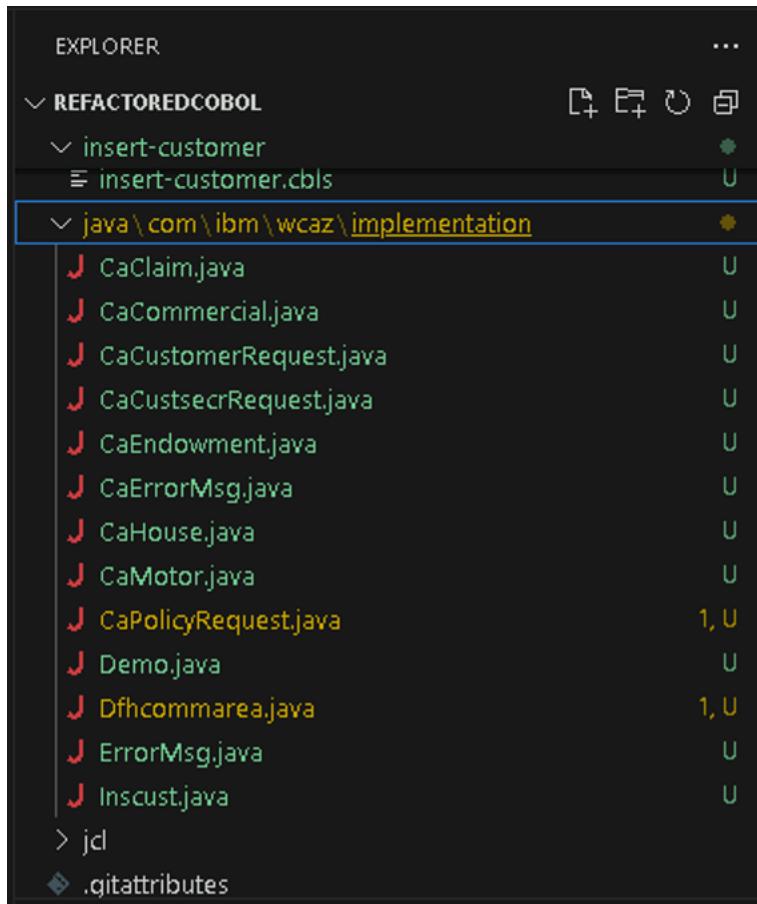
Bulk generation

Generate all remaining Java methods and classes at once. This will take an estimated 6 minutes.

Java name	Java type	COBOL name
com.ibm.wcaz.implementation	Package	
Dfhcommarea	Class	DFHCOMMAREA
Inscust	Class	INSCUST
com.ibm.wcaz.infrastructure	Package	
Lists	Class	Lists
Settable	Class	Settable
com.ibm.wcaz.infrastructure.se	Package	
ByteArraySerializable	Class	ByteArraySerializable

Save **Generate Java classes**

⇒ Next, we click **Generate Java classes**. View the classes generated in the explorer window.



⇒ This process created all the Java objects and classes which map to the COBOL objects. However, none of the actual logic has been migrated yet. Our job next is to migrate the actual paragraph logic. Java method stubs are generated, with a comment saying that the method still needs to be generated. For example:

```
17
18     public void insertCustomer(CaCustomerRequest caCustomerRequest) {
19         // TODO: generate insertCustomer() method with IBM Watsonx Code Assistant for Z
20     }
21
```

⇒ Next, we open our main class Inscust.java, select one of the methods which haven't been generated, right click on the method and select

watsonx Code Assistant for Z > Generate Java method

The screenshot shows a Java code editor with the following code:

```
4
5  public class Inscust {
6      private static Inscust insc
7
8      public Inscust() {}
9
10     public static Inscust getIn
11         return inscust;
12     }
13
14     public void writeErrorMessa
15         // TODO: generate write
16     }
17
18     public void insertCustomer(
19         // TODO: generate inser
20     }
21
22     public void exitPara() {
23         // TODO: generate exitP
24     }
25
26     public static void main(Com
27         CaCustomerRequest caCust
28         getInstance().inscustIns
29         holder.setValue(caCusto
30     }
31 }
32 }
```

A context menu is open over the line containing the call to `holder.setValue(caCusto`. The menu items are:

- View as Optimization Report
- Go to Definition F12
- Go to Declaration
- Go to Type Definition
- Go to Implementations Ctrl+F12
- Go to References Shift+F12
- Go to Super Implementation
- Peek > errorMsg) {
 ↓ Code Assistant for Z
- Find All References Shift+Alt+F12
- Find All Implementations
- Show Call Hierarchy Shift+Alt+H
 ↓ Code Assistant for Z
- Show Type Hierarchy
- watsonx Code Assistant for Z > Generate Java method
- Submit as JCL
- Run Equivalence Test
- Rename Symbol F2
- Change All Occurrences Ctrl+F2
- Format Document Shift+Alt+F
- Format Document With...
- Format Selection Ctrl+K Ctrl+F
- Refactor...
- Source Action...

- ⇒ On the right, watsonx will generate the Java equivalent of the Cobol code. Spend some time looking at this code and comparing it to the original Cobol.

The screenshot shows the WatsonX interface with the title "Java methods" and a tab labeled "AI". The main area is titled "Code transformation" and displays the following Java code:

```

public void writeErrorMessage(DPCommonArea dPCommonArea, ErrorMsg errorMsg) {
    CaErrorMsg caErrorMsg = new CaErrorMsg();
    int wsAbstime = 0;
    String wsDate = "";
    String wsTime = "";
    errorMsg.setDbSqlIn(JdbcConnection.sqlCode);
    try {
        wsAbstime = AskTimeUtils.askTimeAbstTime();
        wsDate = CICSDateTimeFormat.MHDDVYYY.formatDateTime(wsAbstime, ZoneId.systemDefault());
        wsTime = CICSDateTimeFormat.HHHHSS.formatDateTime(wsAbstime, ZoneId.systemDefault());
    } catch (CicsConditionException e) {
        e.printStackTrace();
    }
    errorMsg.setEdate(wsDate);
    errorMsg.setETime(wsTime);
    try {
        Program jdec1Program = new Program();
        jdec1Program.setName("LGSTSQ");
        jdec1Program.link(errorMsg.getBytes());
    } catch (CicsConditionException e) {
        e.printStackTrace();
    }
}

```

At the bottom of the code editor, there is a blue button labeled "Insert". Below the code editor, there is a question "How is this rating data used?" with a thumbs-up and thumbs-down icon.

- ⇒ Click Insert.

- ⇒ Repeat the above steps for any other method which need to be migrated. A quick way to find all the instances is to search all files for “// TODO: generate”. In our case we only have the 3 methods in the Inscust.java file, but for larger cases these may be in multiple Java classes.
- ⇒ Notice that exitPara() doesn't generate any code, this is because the paragraph is empty in COBOL.

You may notice there are some errors in the Java code. For this application there are a couple causes for these issues. First the tool isn't a 100% one click migration effort, and we wanted to make sure we showed a real-world example. We can look at a few of the steps we'll have to change and why.

- This program connects to a database, but watsonx had no way to determine which database it connected to, so it inputs a generic Database connection. This will have to be modified with the drivers and login credentials for the Database you're connecting to.

- Import statements are outside of the method login, so we need to import them manually. VS code helps with quick fixes.
- Some calls are to other Cobol programs which we haven't migrated yet, such as ASKTIME and FORMATTIME etc.

⇒ This is where we'll end the migration. Feel free to try different pieces of code throughout. Feedback section

8.1 Reach out to the Lab instructor.

David VandePol – yandepol@ca.ibm.com