RBC x IBM

# Bootcamp for watsonx Code Assistant for Z

Revolutionizing Mainframe Application Modernization with AI

# Table of Contents



**Notices and disclaimers**

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.

# 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)

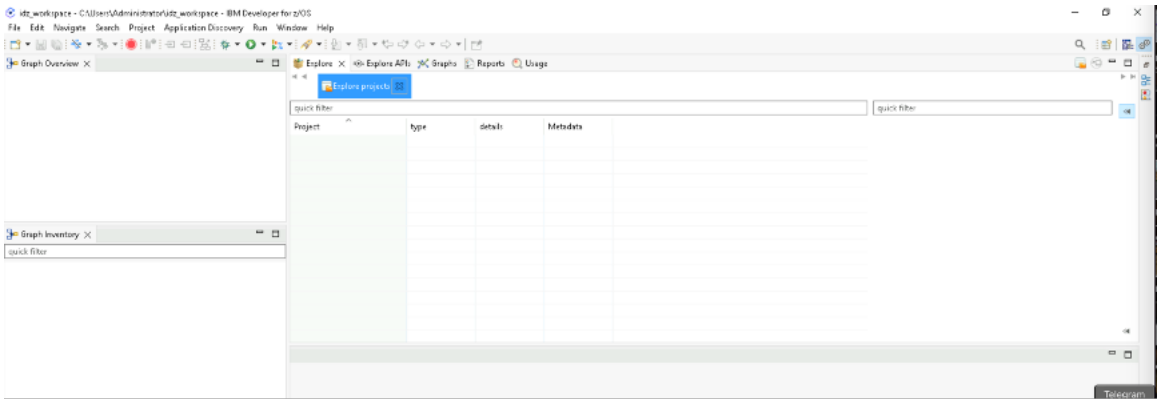
ADDI (Application Discovery and Delivery Intelligence) is a graphical tool used to analyze and understand your application landscape. For this lab, we've already loaded the necessary source code.

⇒ Open IBM Developer for z/OS (IDz)

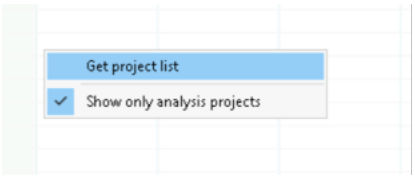
⇒ The default workspace should load automatically. If prompted for a workspace location, enter:
*C:\Users\Administrator\idz_workspace*

**Navigating the Application in ADDI**

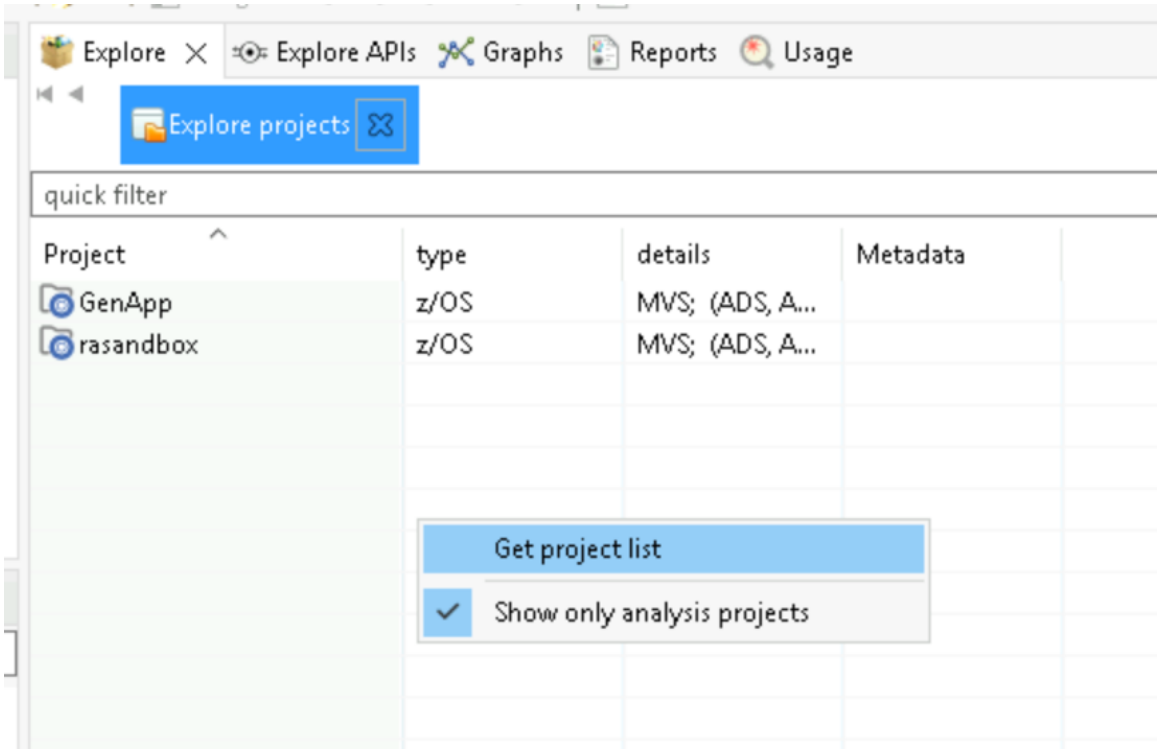
⇒ Open the Explore view in ADDI.



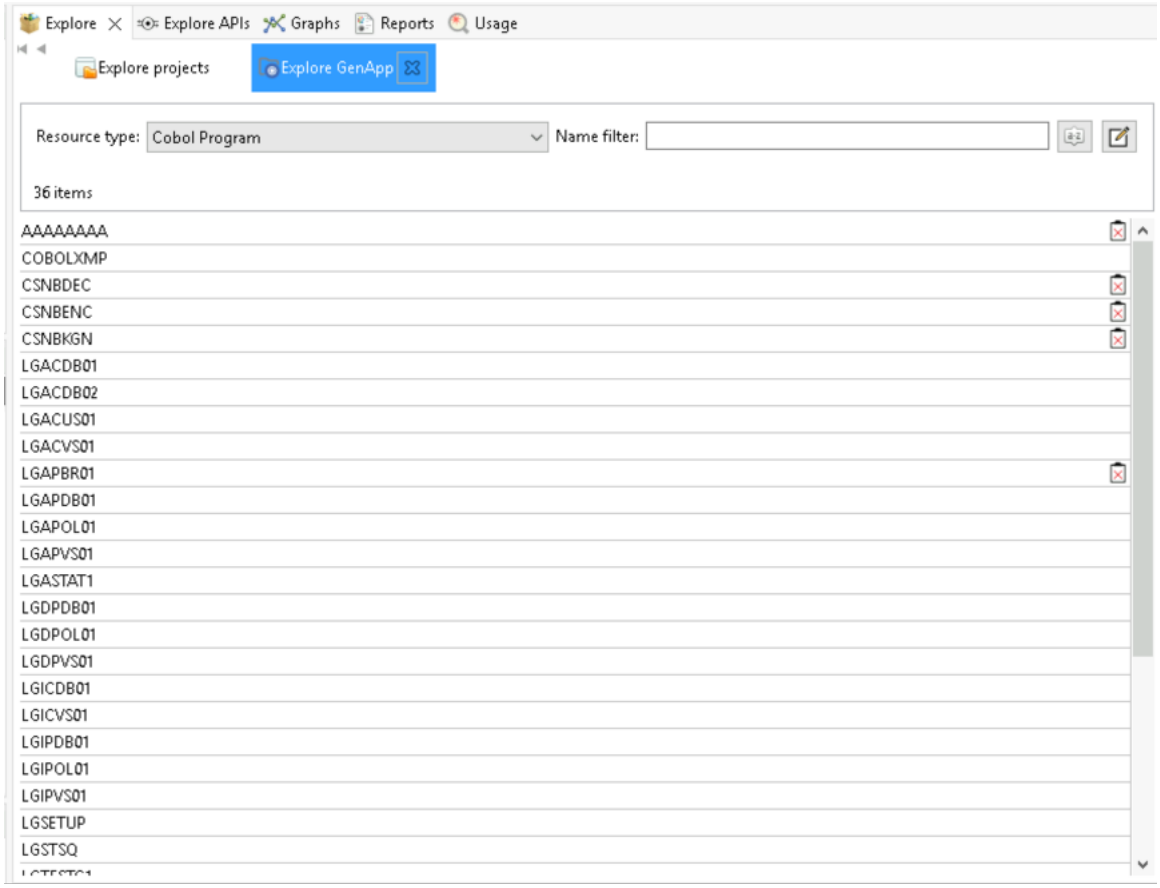
⇒ If this is your first time launching IDz (or if a new build was completed), refresh the project list:
- Right-click anywhere in the project list panel.
- Select "Get project list."



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




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

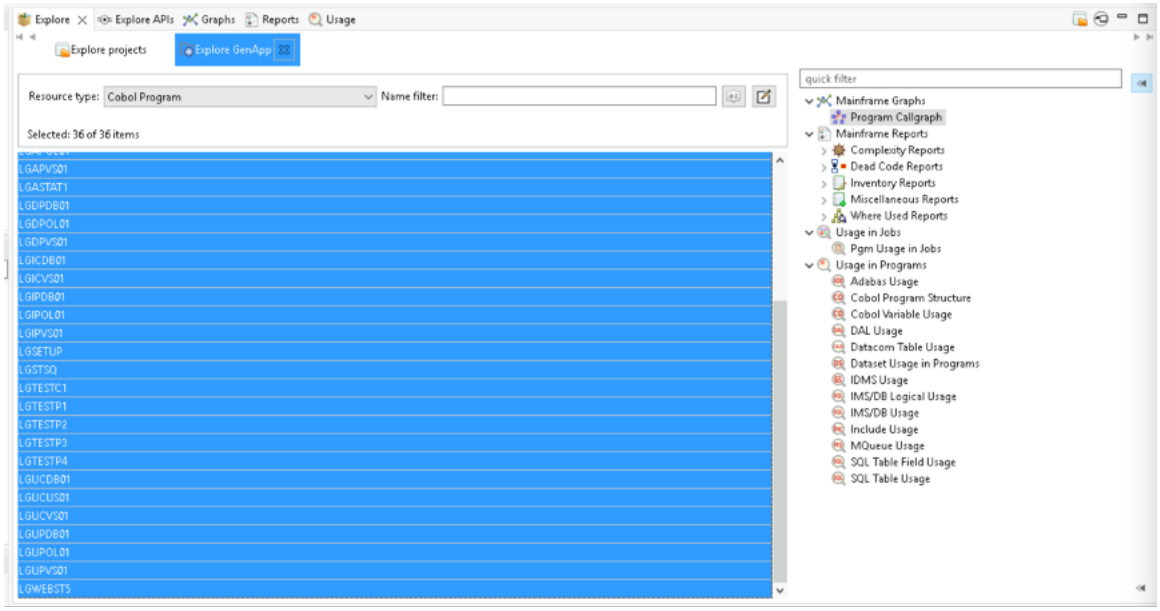

Explore
Explore APIs
Graphs
Reports
Usage
Explore projects
Explore GenApp
Resource type:
Cobol Program
Name filter:
36 items
AAAAAAAA
COBOLXMP
CSNBDEC
CSNBENC
CSNBKGN
LGACDB01
LGACDB02
LGACUS01
LGACVS01
LGAPBR01
LGAPDB01
LGAPOL01
LGAPVS01
LGASTAT1
LGDPDB01
LGDPOL01
LGDPVS01
LGICDB01
LGICVS01
LGIPDB01
LGIPOL01
LGIPVS01
LGSETUP
LGSTSQ

# 4 Visualize the entire Application Landscape

To view the entire application and understand the interconnections:

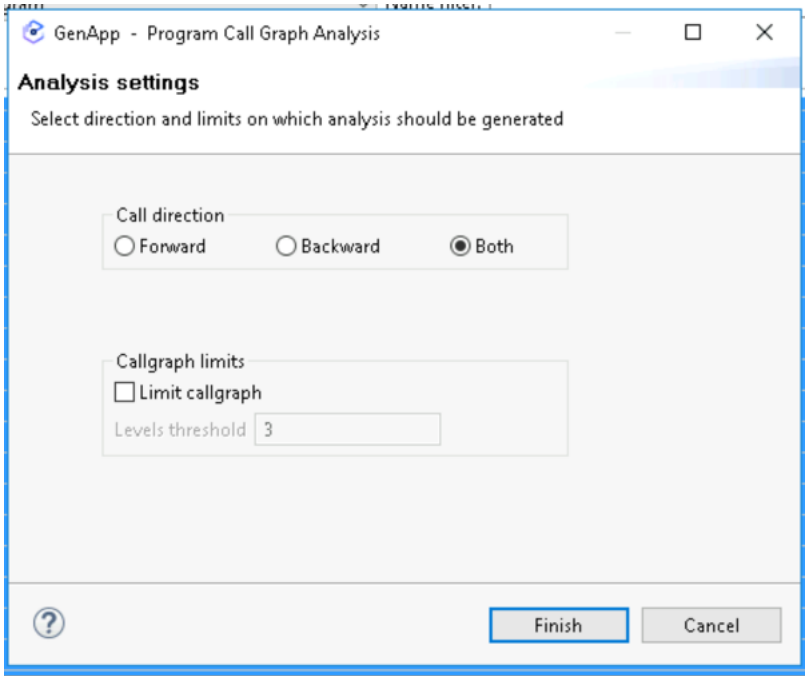
⇒ Select all programs within GenApp:
- Click the first program.
- Hold Shift and click the last program in the list.

⇒ With all selected, go to the right-hand side panel and click on **Program Callgraph**.

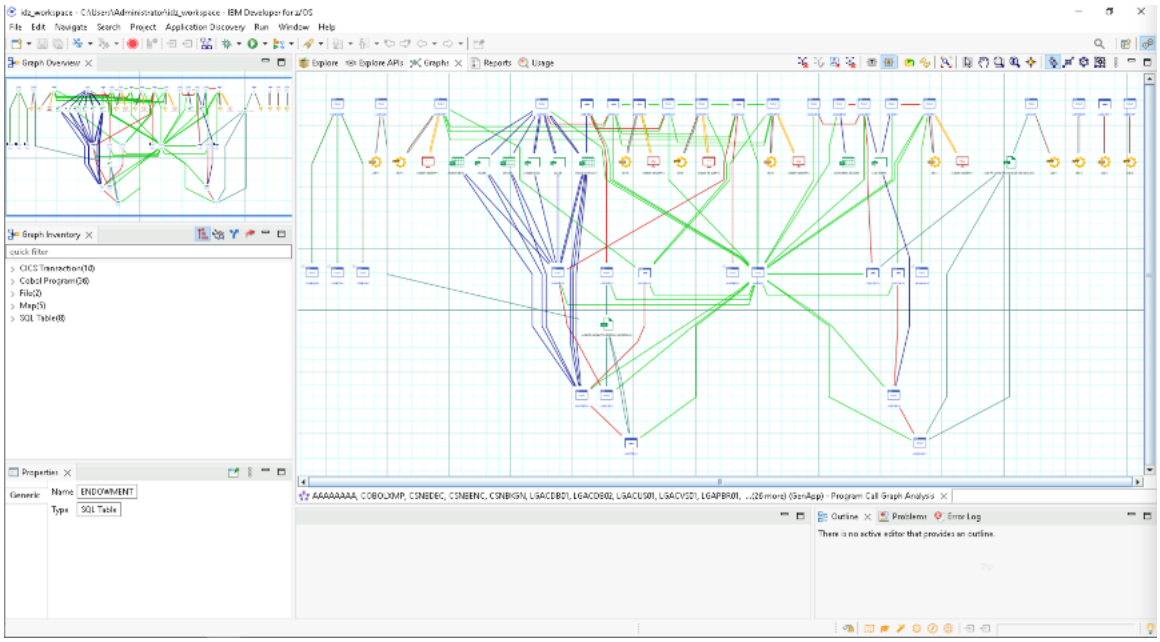



⇒ In the Analysis Settings window:
- Set Call Direction to Both.
- Uncheck the “Limit callgraph” option. See the illustration below

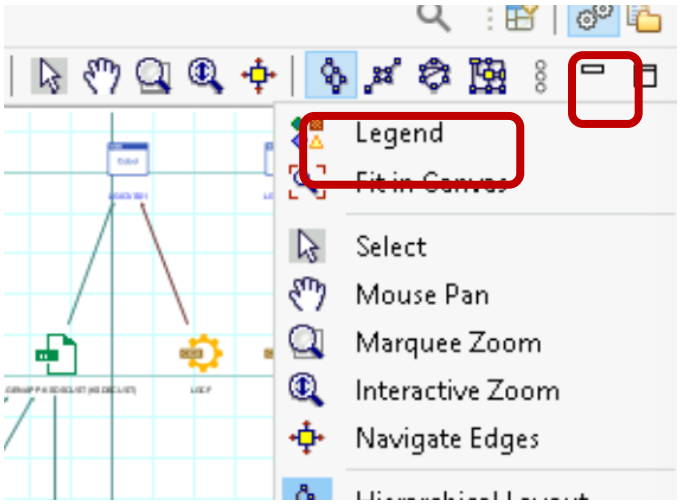
- Click Finish.



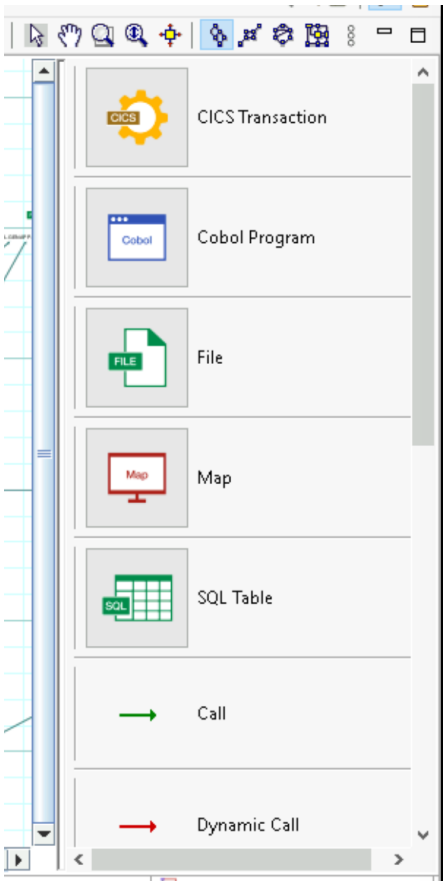

⇒ View the results:



⇒ 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:



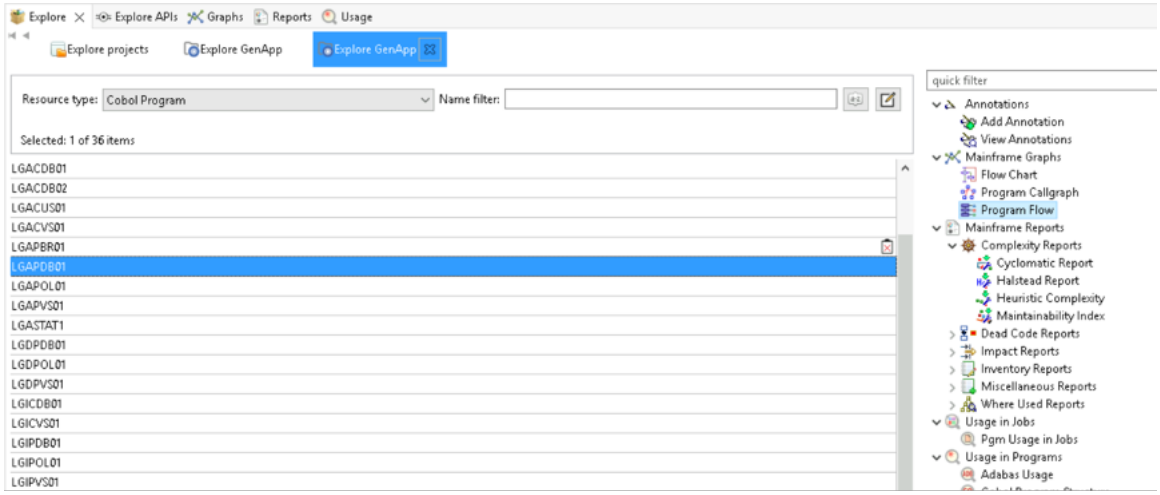
# 5 Deep Dive into Specific Applications

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

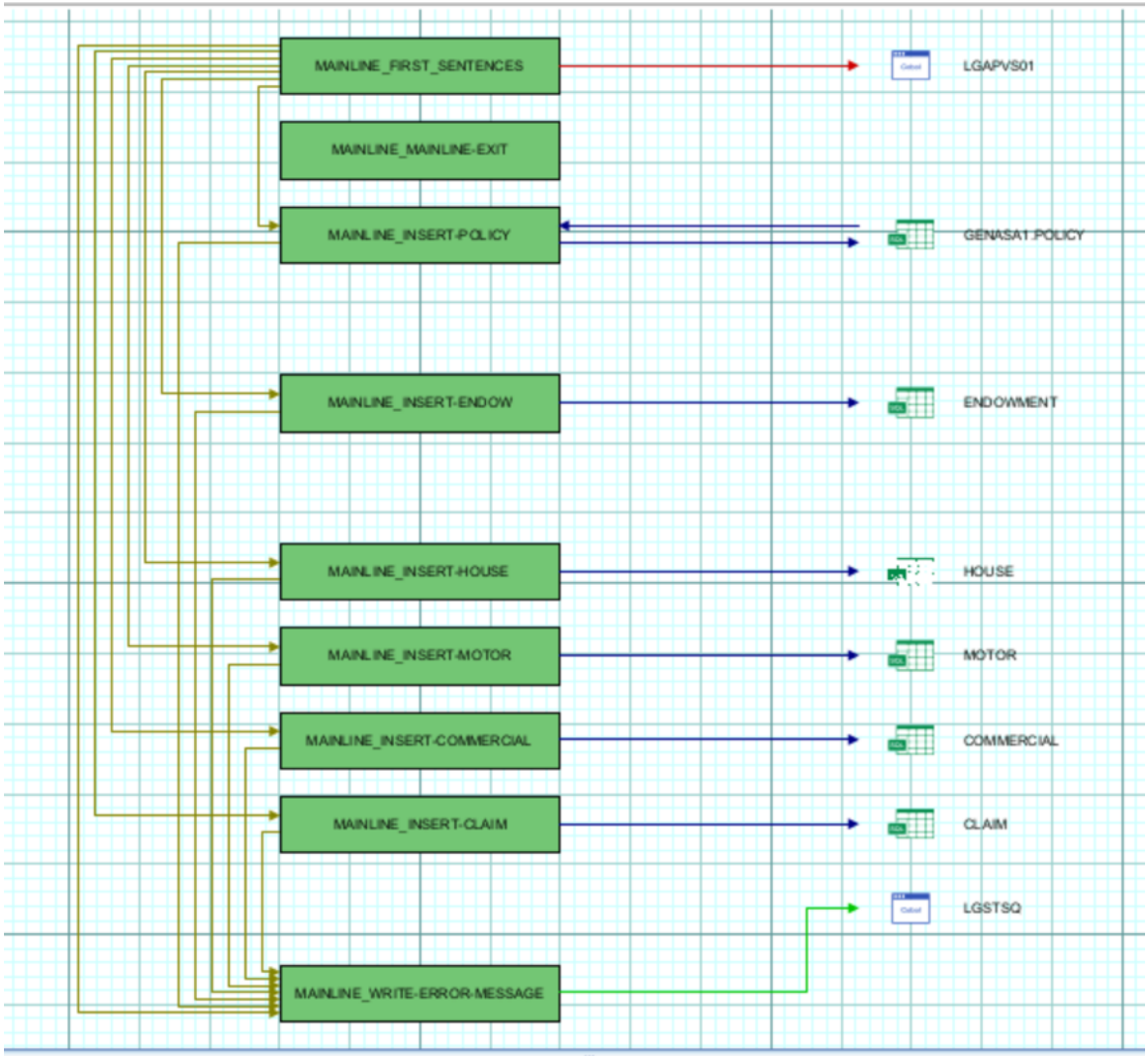
⇒ Navigate back to the **Explore tab** to analyze a specific program in more detail.

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**



⇒ 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.

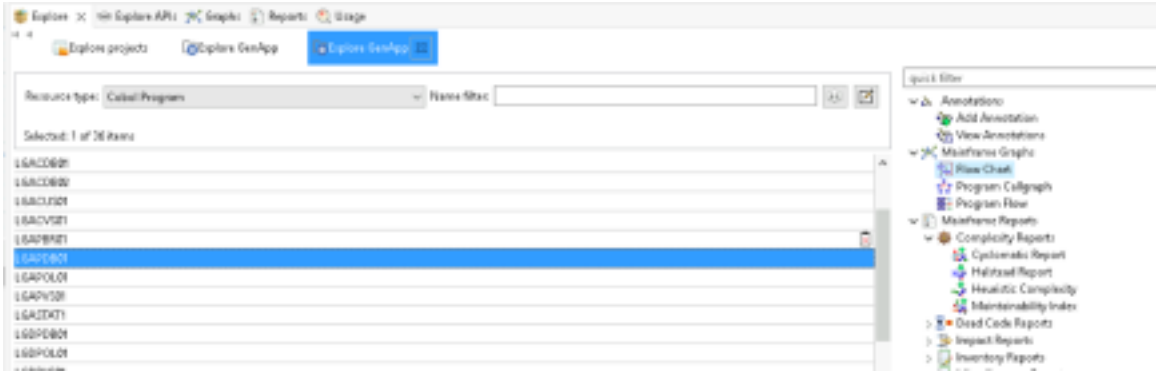



⇒ Click on any box and it will open the source code to the start of the specific paragraph.

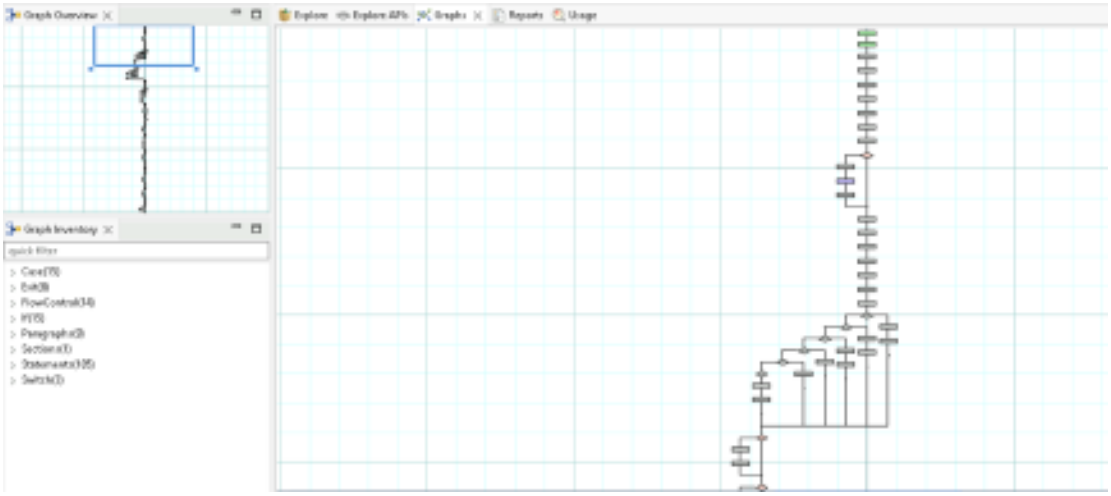
⇒ Play around with different views of the graph by clicking different layout options in the top right of the menu bar.

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

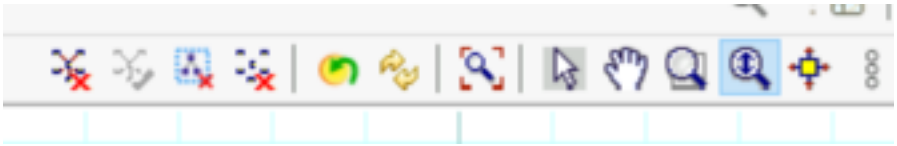
⇒ Next, we can look at the Flow chart of the application for a more granular view within the specific paragraphs.



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



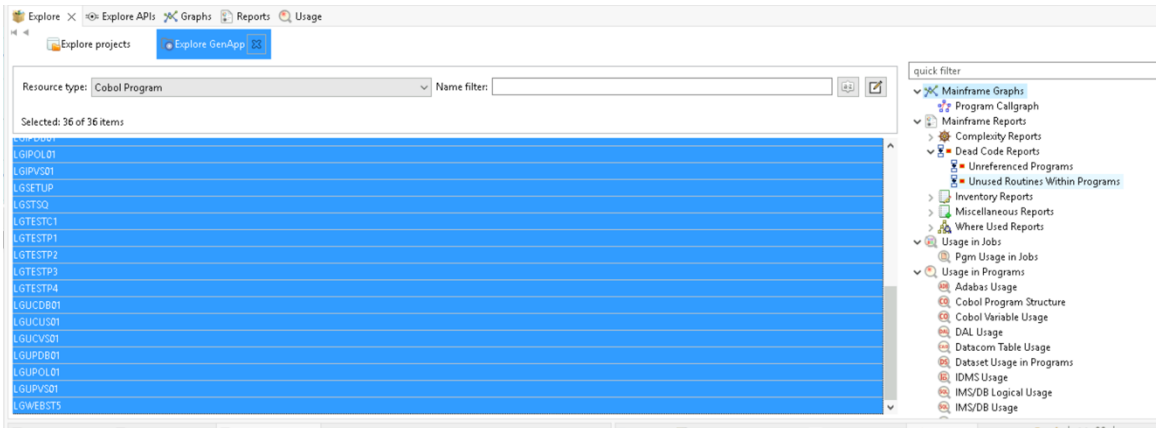
⇒ Notice the Graph overview on the top left so you can see where you are within the application and notice the toolbar on the top right for different zooming options



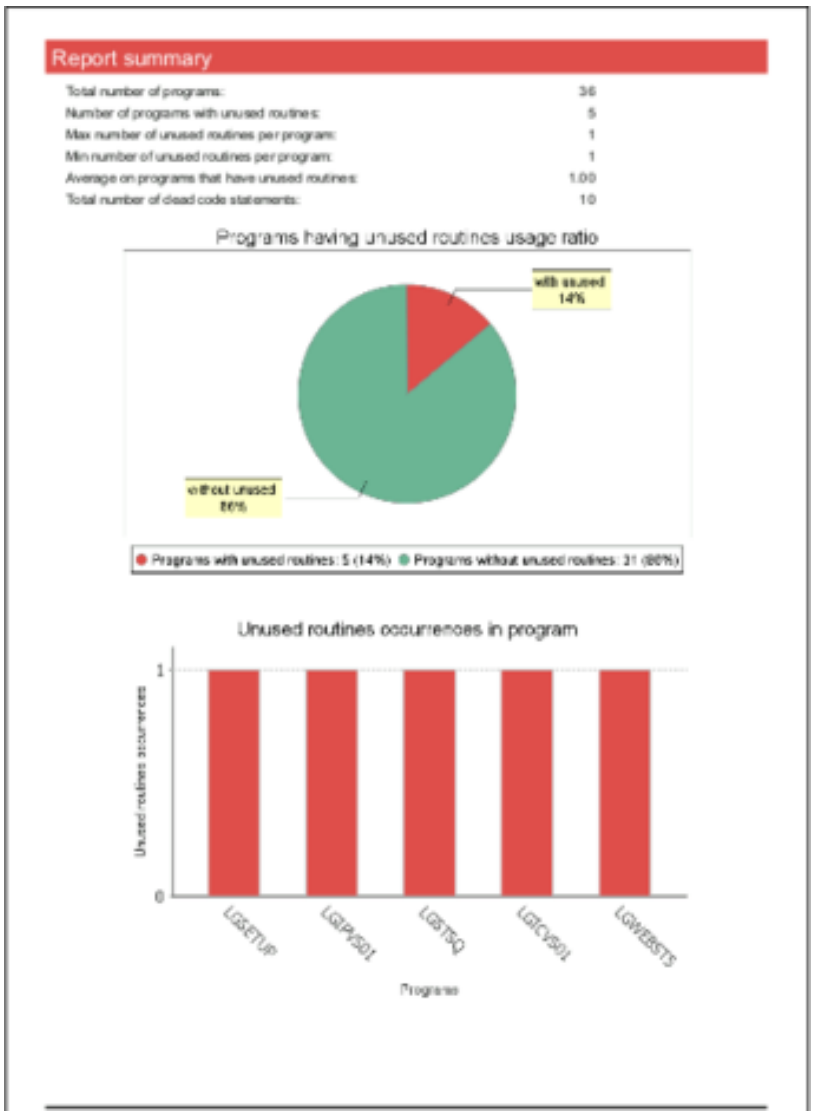
## More Reports and views:

⇒ 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**




⇒ 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

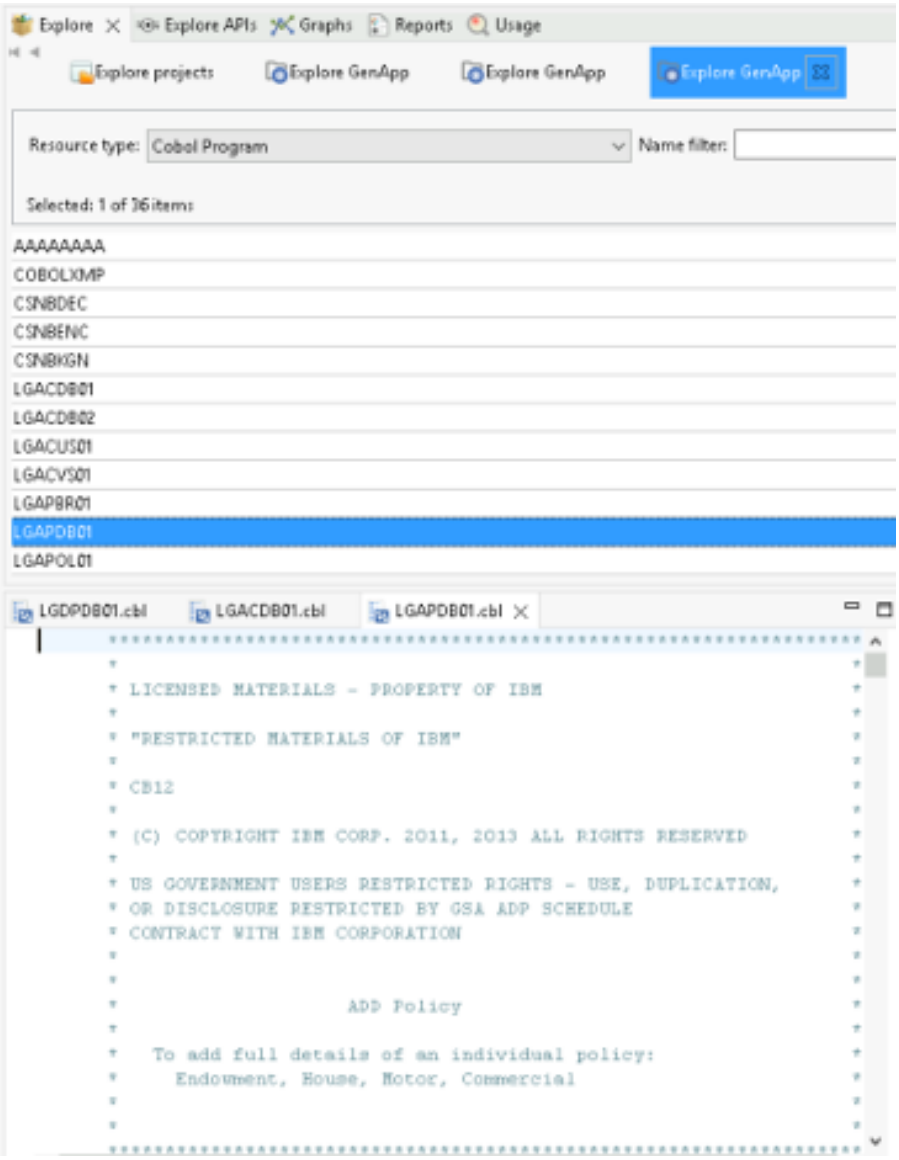
- Complexity Report
- Dead Code Report

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

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

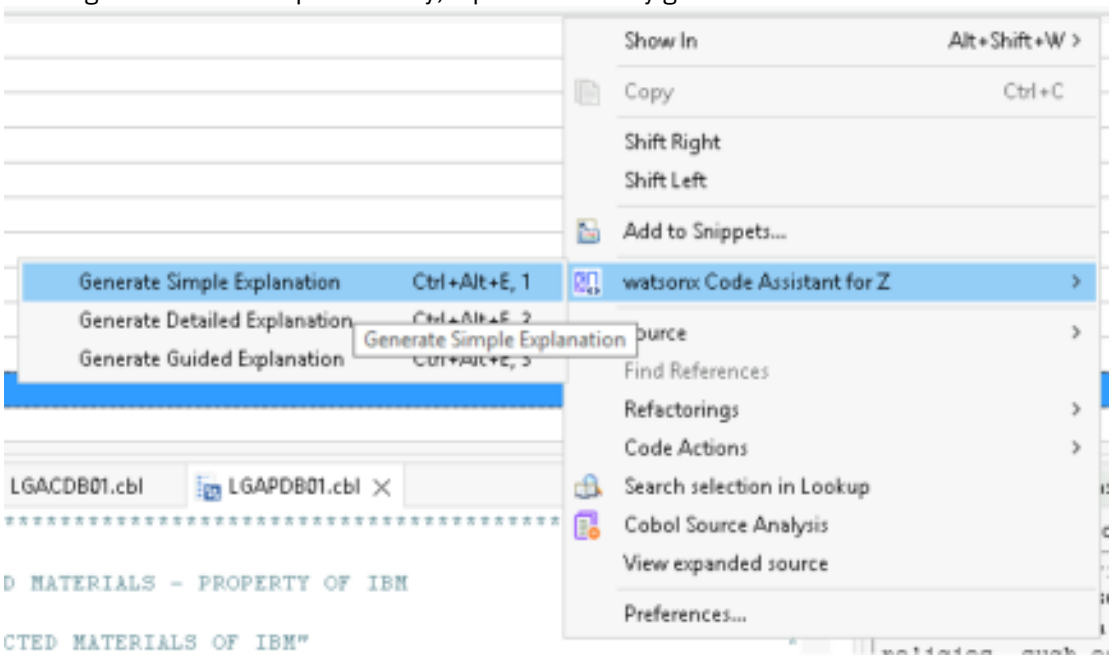
We're going to look at the Code Explanation feature. The code explanation feature is available in both IDz as well as VSCode, we're going to look at IDz first because we've already got it open.

⇒ 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.



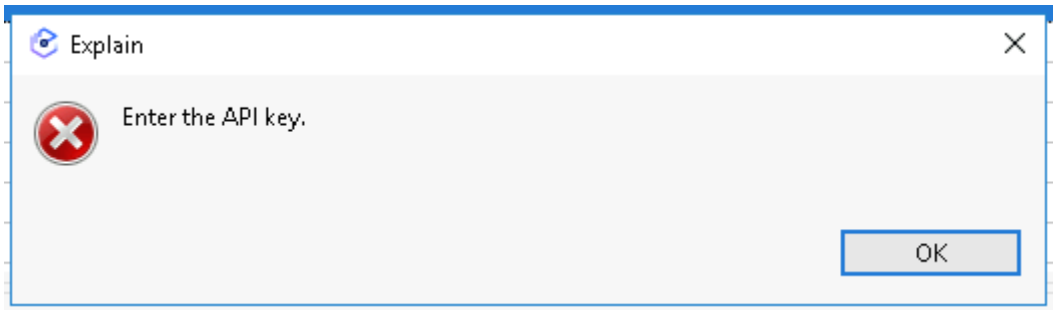

⇒ Right click anywhere on the source code and select:
**watsonx Code Assistant for Z > Generate Simple Explanation**

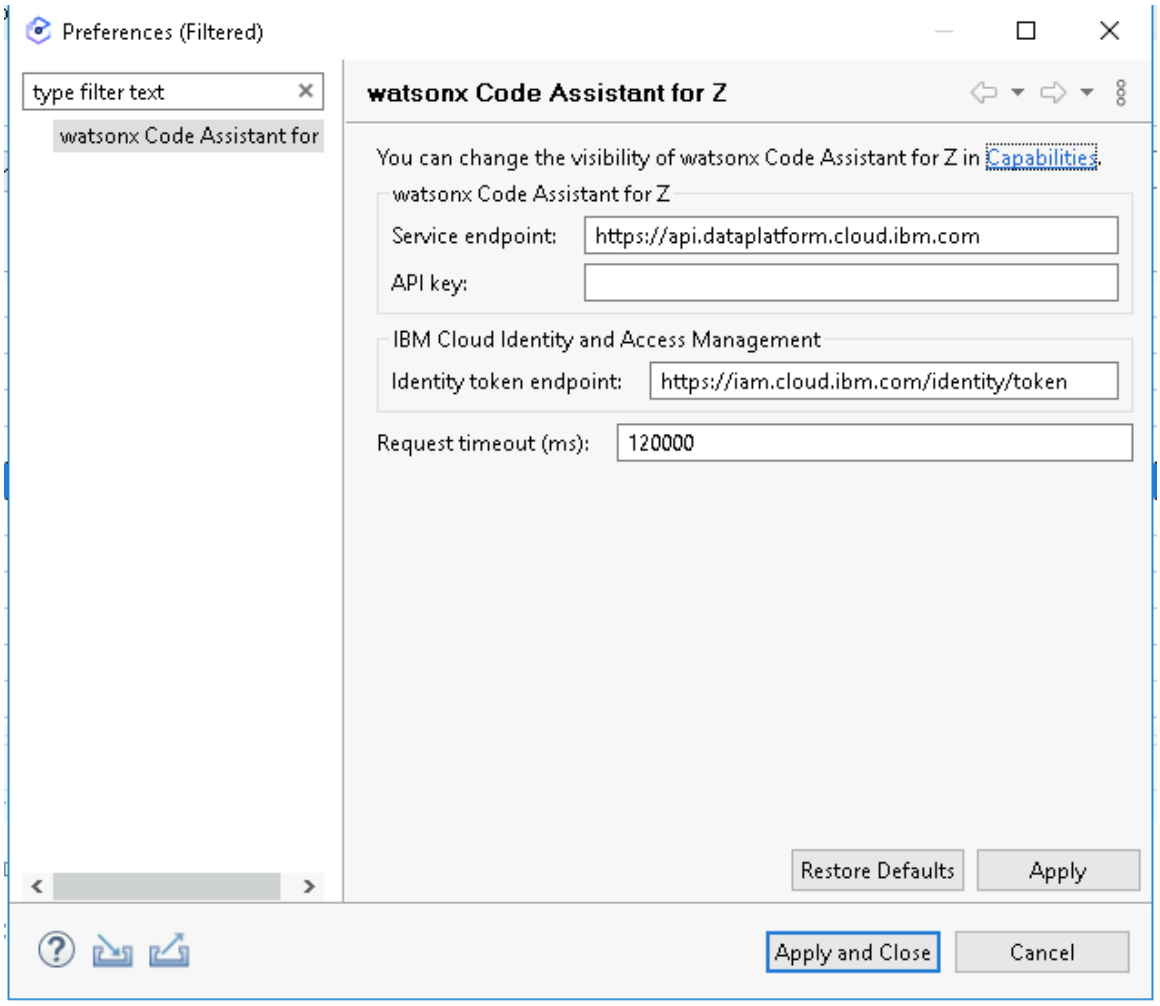
You might be asked to input API Key, input the API key generated in IBM Cloud.




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

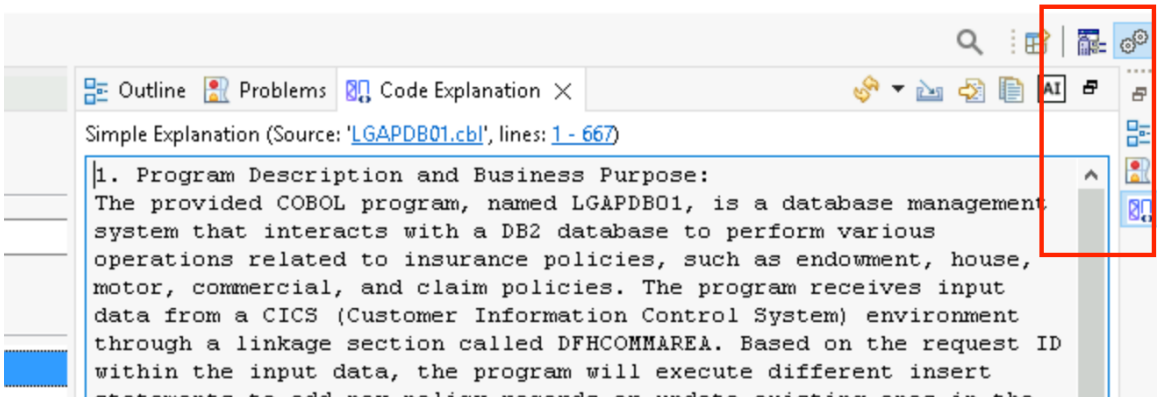
⇒ You might get an error saying you need to input your API key, this API key should have been sent to you ahead of time. Check for APIKey.txt on the desktop. Contact the Lab runner if you don't have one.



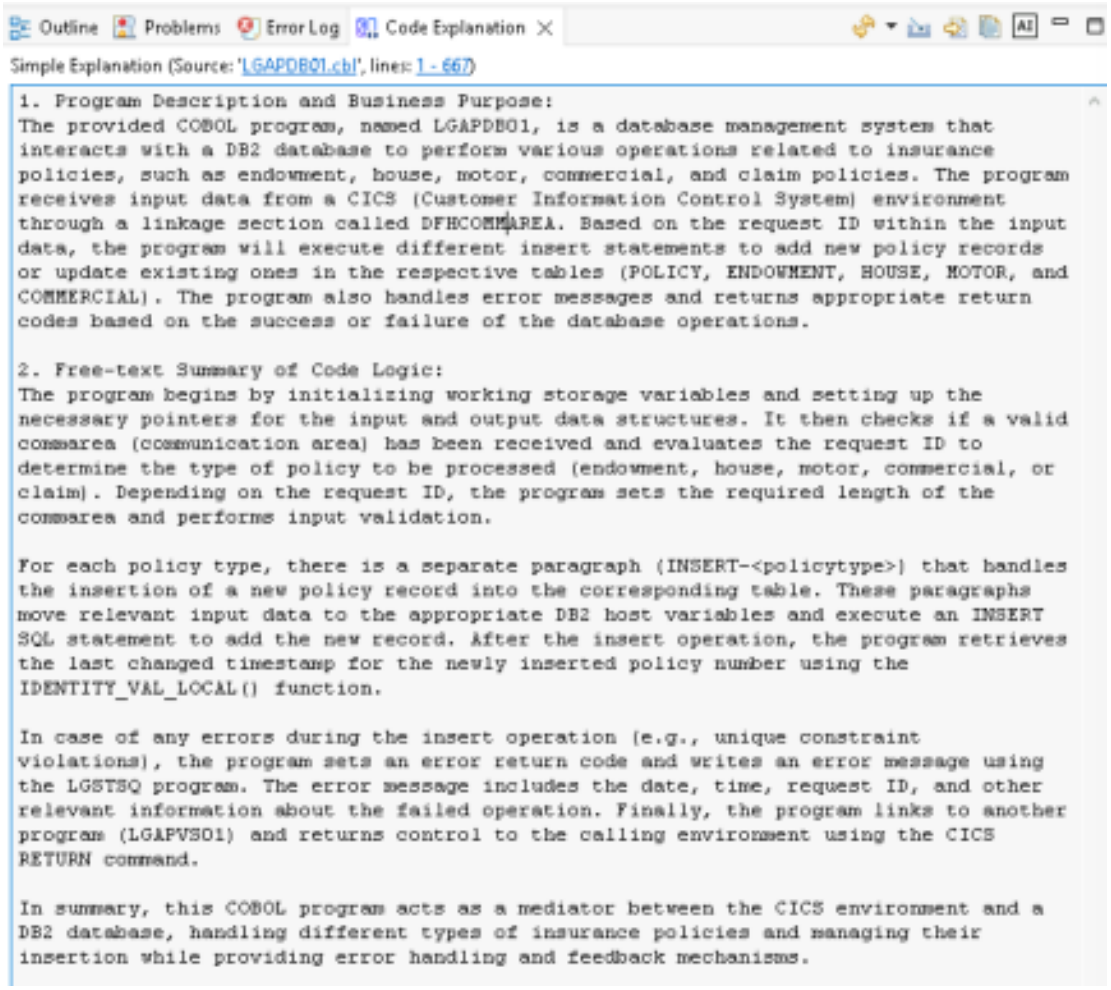



⇒ Once you've inputted the API key, try to generate the explanation again

⇒ 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.




⇒ 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)

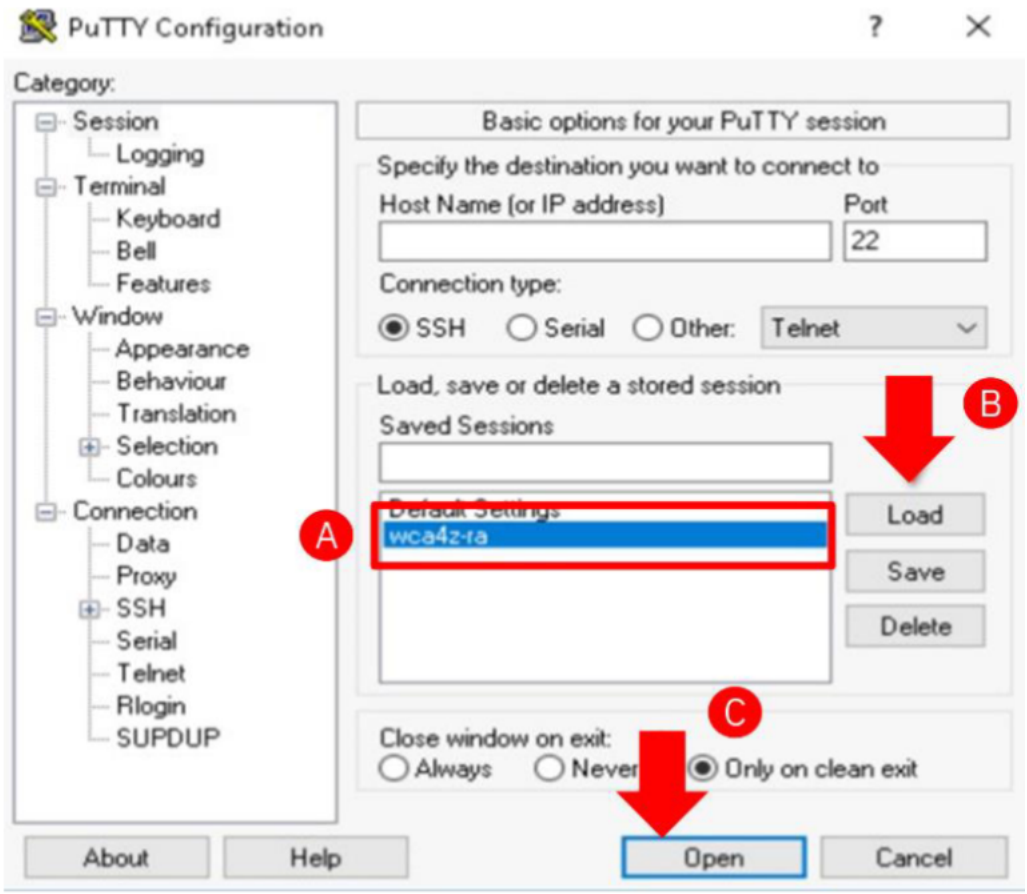
Refactor Assistant is accessed via Visual Studio Code.

Pre-work Completed:

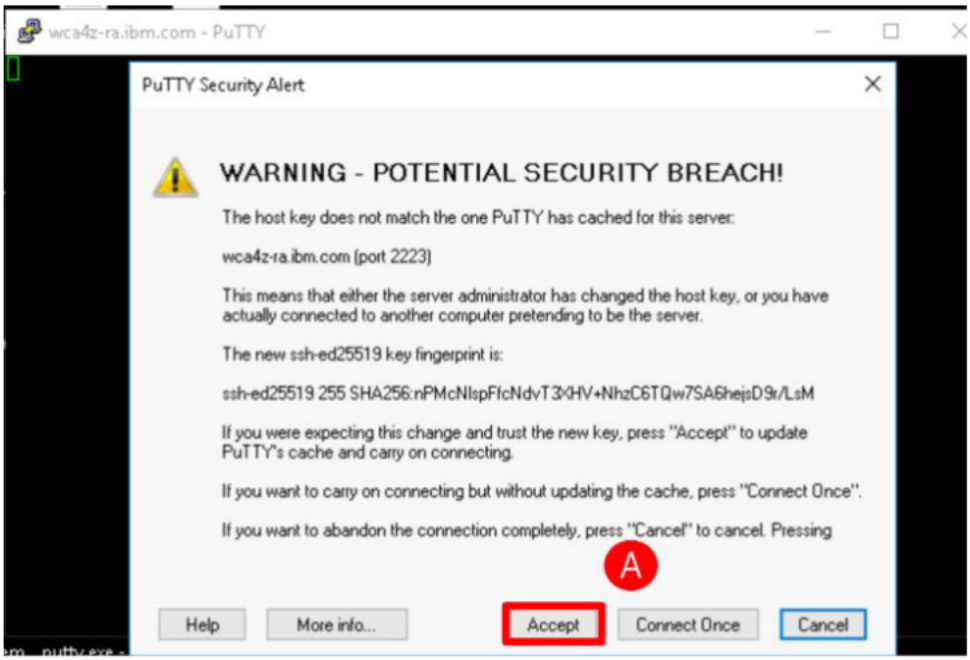
- The WCA4Z plugin is already installed, we've also pre-done most of the configuration to connect the servers together. To start RA follow these steps

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

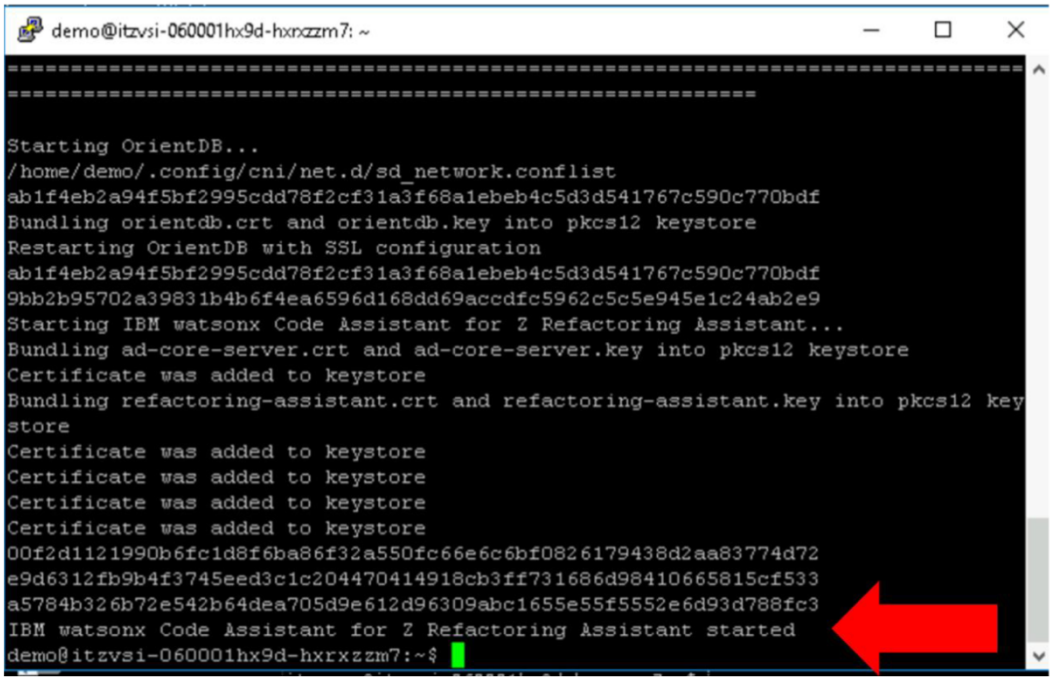

⇒ Within PuTTY, click on the wca4z-ra profile, click Loan and then Open.



⇒ If you see this warning, click Accept




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

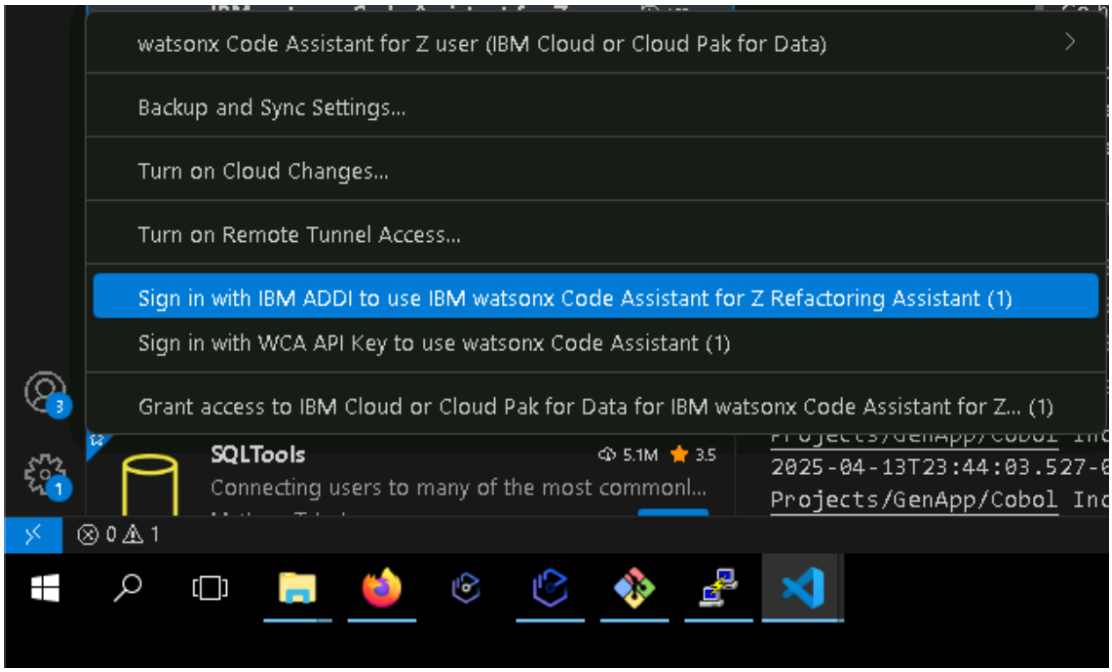



⇒ The pre-work for connecting RA is done, you can proceed to use RA.

### 7.1.1 Complete the following steps

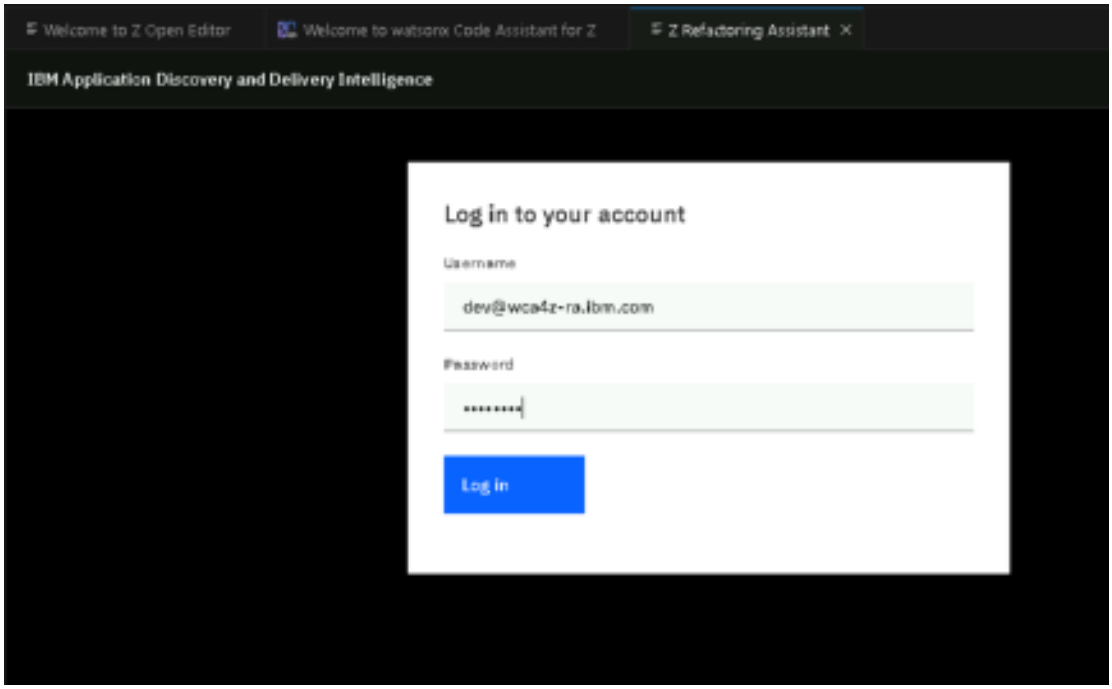
⇒ Double-click the VS Code icon on the desktop.


⇒ In the bottom-left corner, then click:
**Sign in with IBM ADDI to use IBM watsonx Code Assistant for Z Refactoring Assistant.**



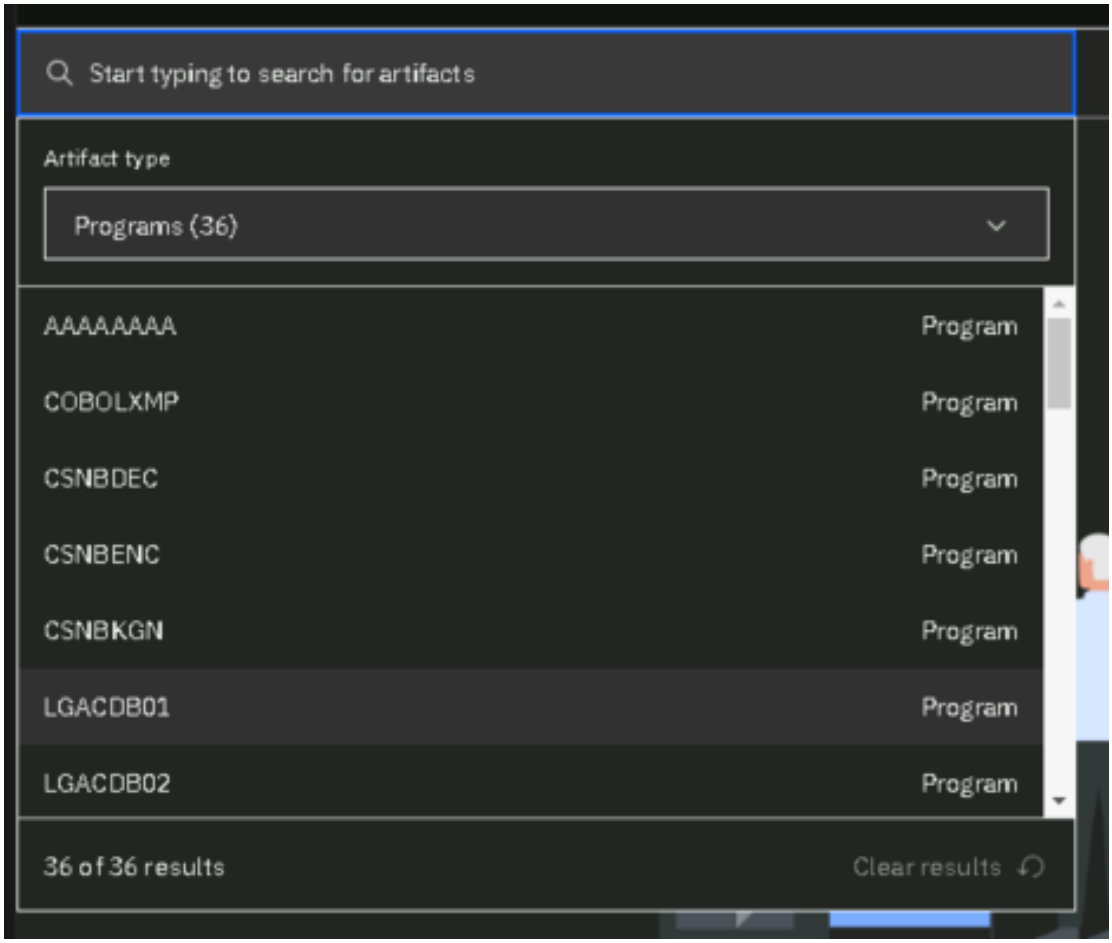

⇒ When prompted, use the following credentials:
Username: dev@wca4z-ra.ibm.com

Password: password

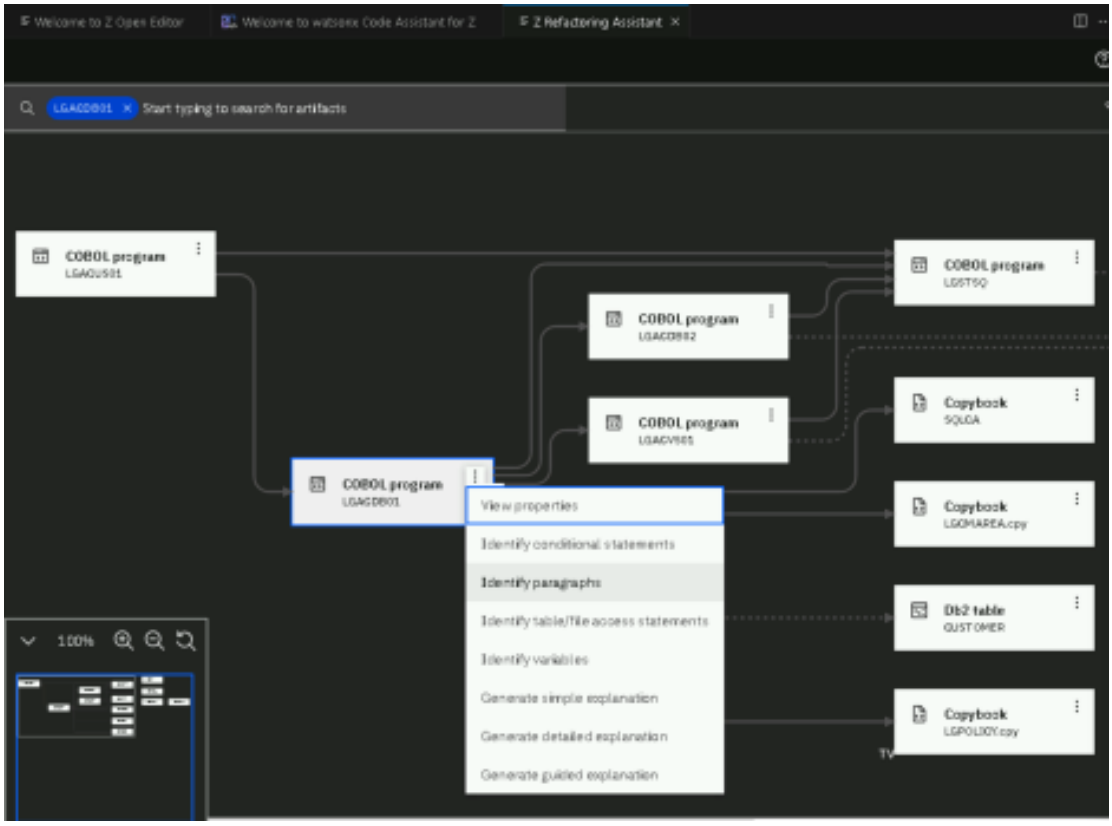



⇒ From the top menu you will see a drop down to select the project to work with. Select GenApp (This may take 30 seconds to load)

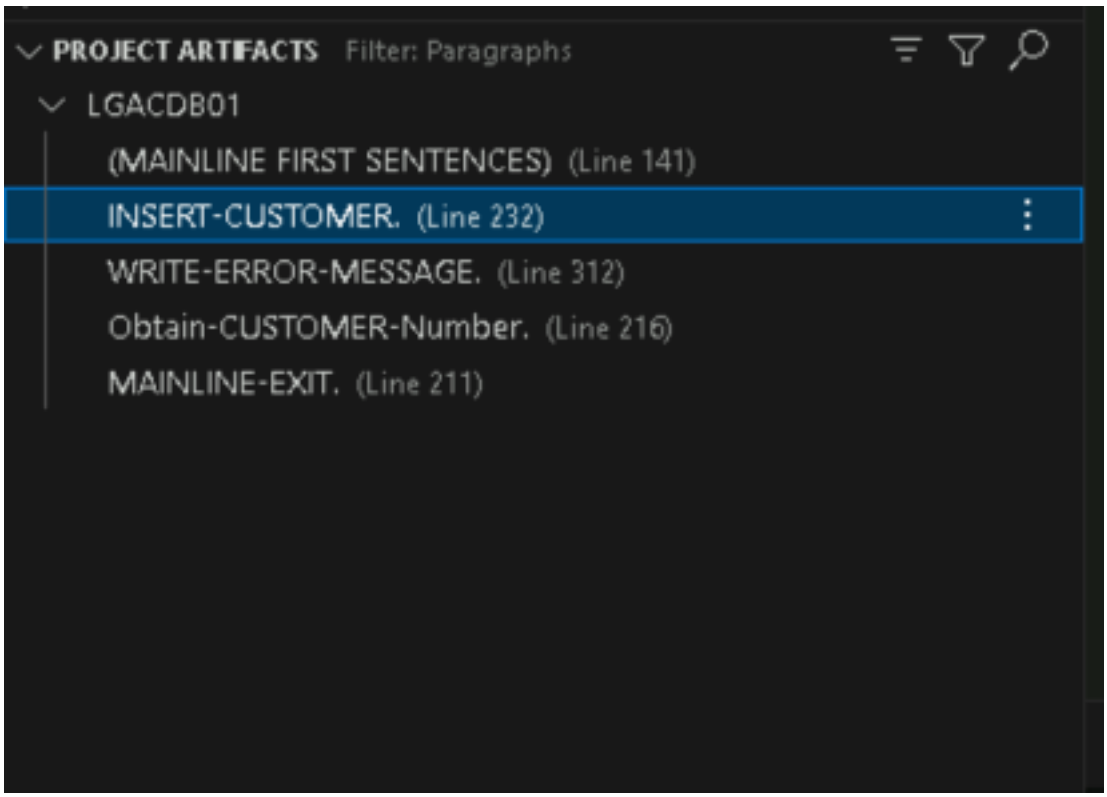
⇒ You should see the screen below, Filter by *Programs* and select **LGACDB01**



⇒ Find the COBOL program LGACDB01, click on the 3 vertical dots and select:
**Identify paragraphs.**

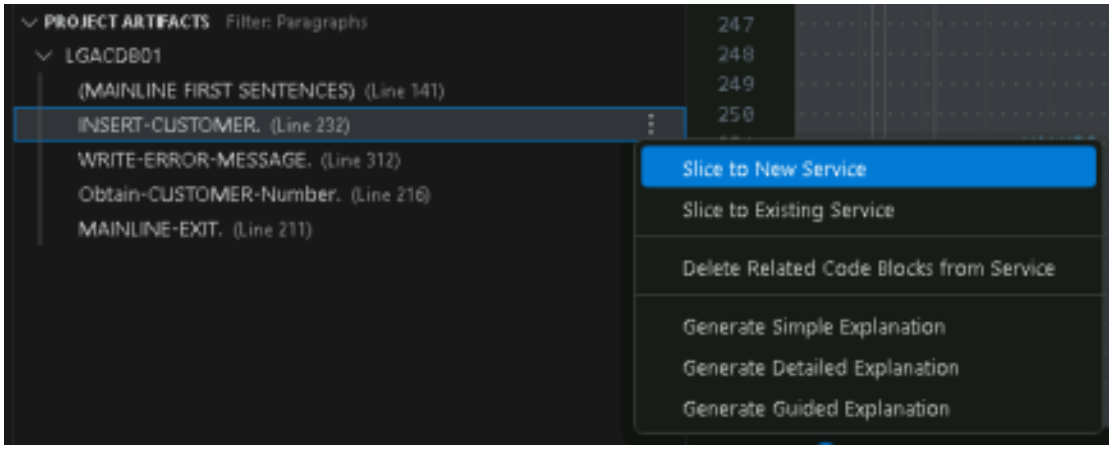



⇒ On the left side you should see a list of all the Paragraphs for the application.

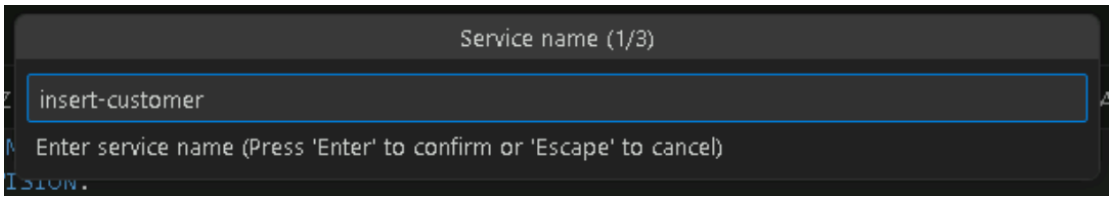


⇒ We're going to isolate **INSERT-CUSTOMER** into its own standalone service.

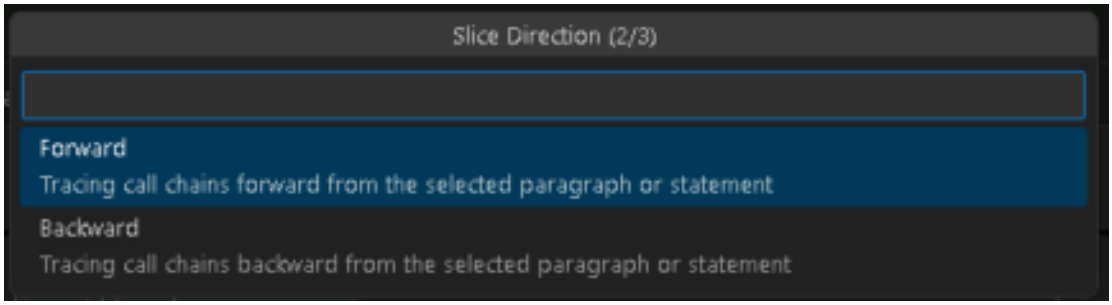
⇒ Select the three dots on the right side and select **Slice to New Service**



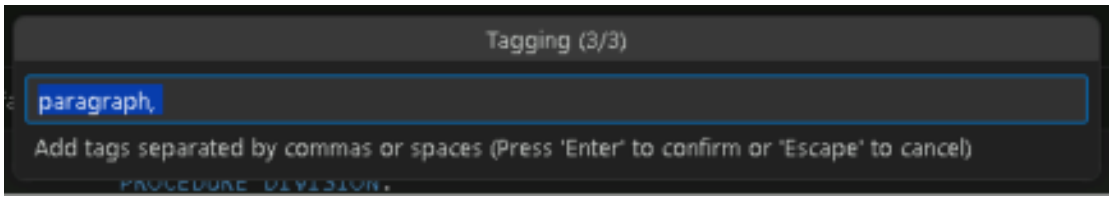

⇒ On the top you will see a text bar open asking to **Enter service name**, you can name it whatever you want, but we'll name it *insert-customer*




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

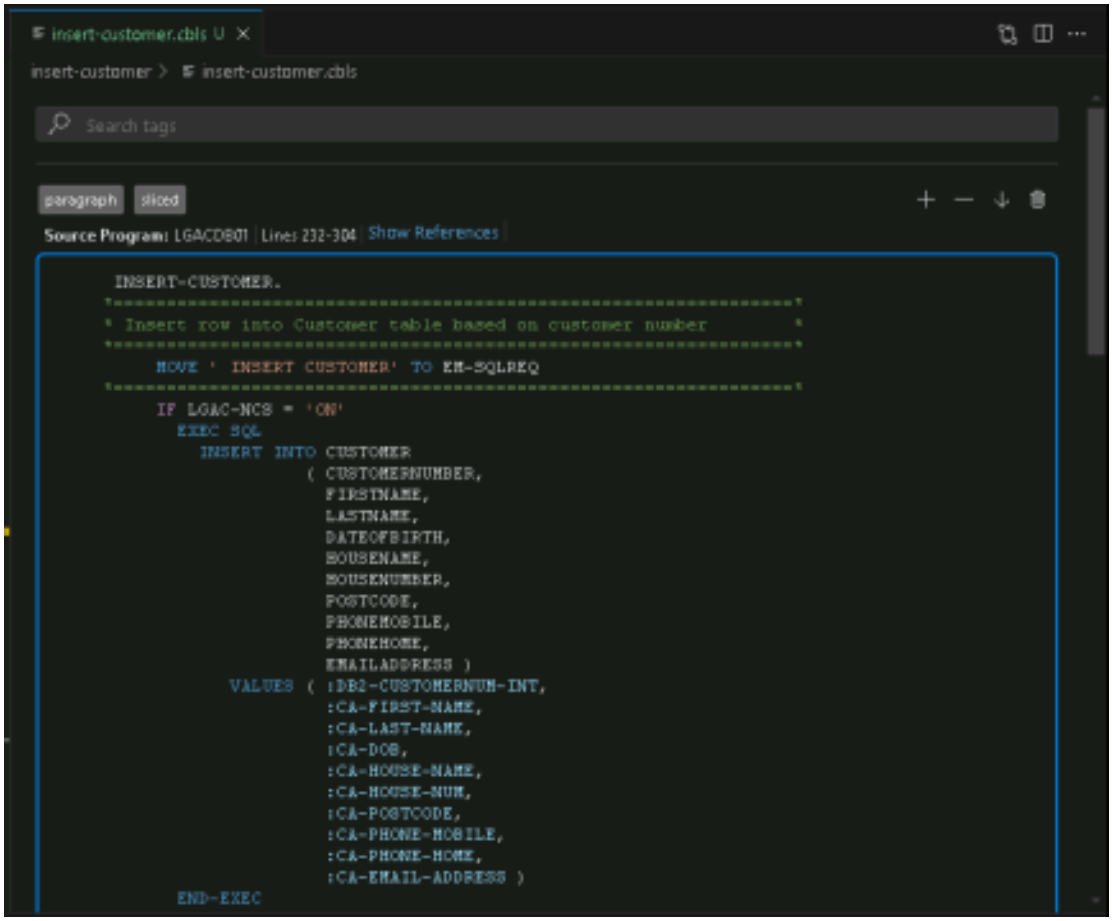



⇒ Lastly it will ask for Tagging, leave the default value **paragraph**, and hit Enter.

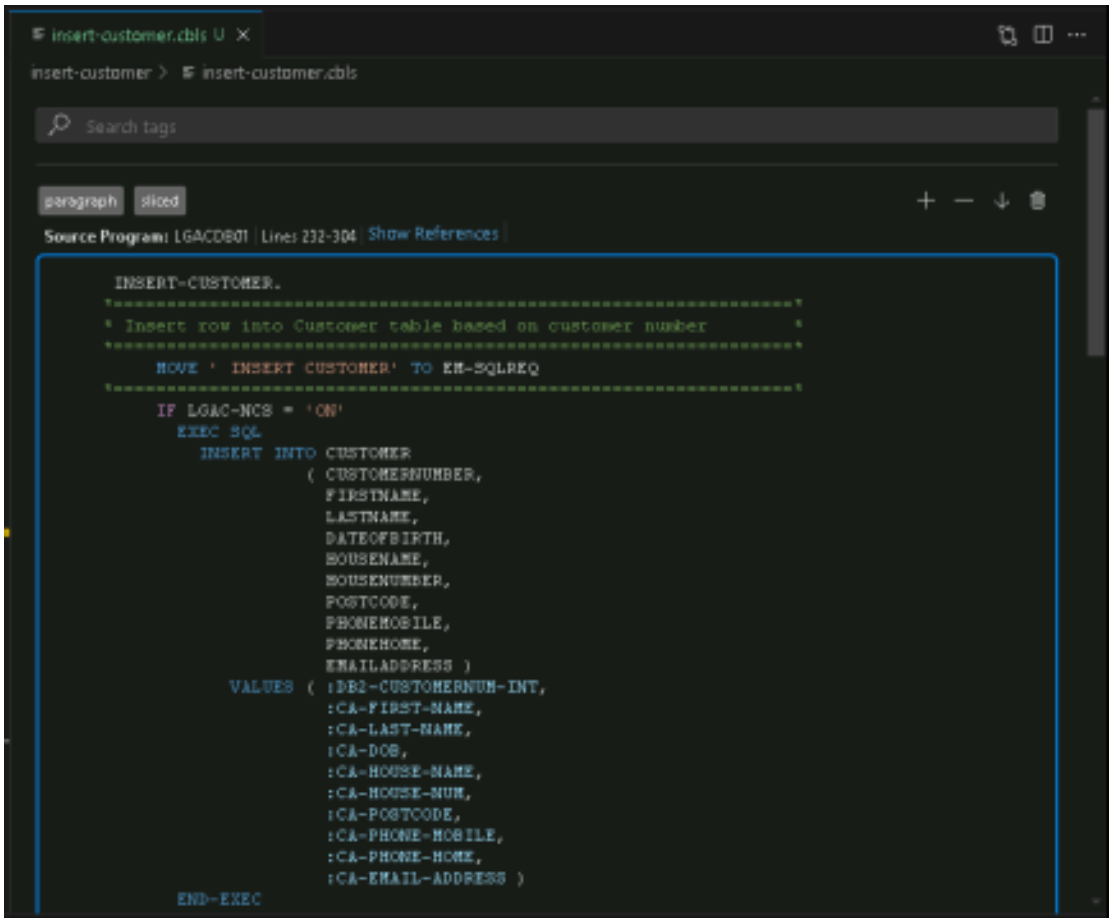



⇒ On the right side you'll see a new tab open called **insert-customer.cbls**
.cbls is a cobol slice. This contains the paragraph you selected, but if you scroll down you'll notice it also included all the paragraphs in which this paragraph calls. It does this recursively. In this case it's only 2

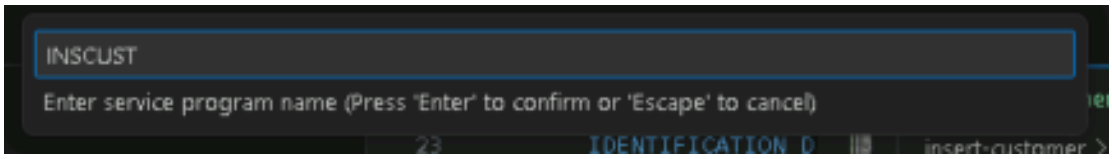
paragraphs, but in larger applications this can be a much larger slice




⇒ 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**

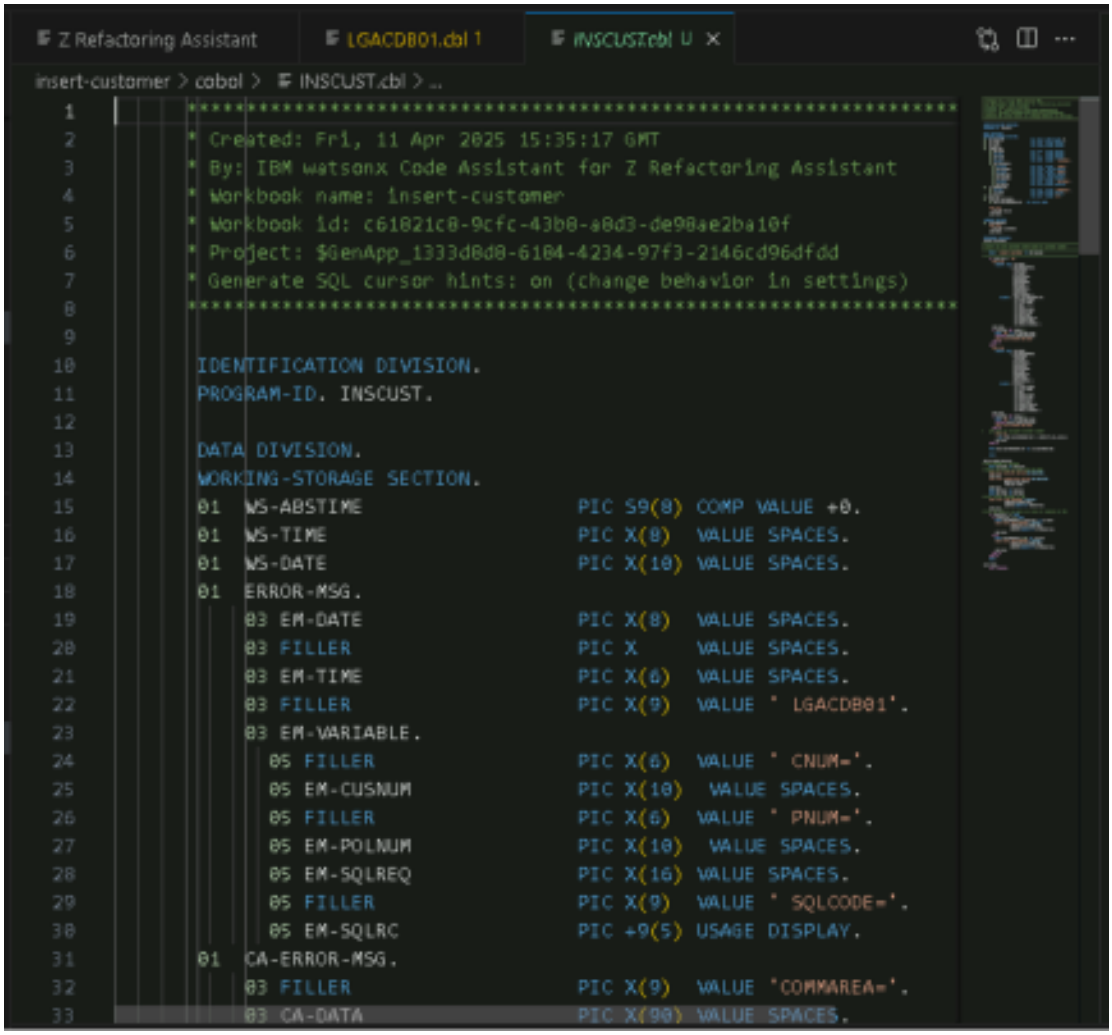



⇒ You will be prompted to Enter service program name. This can only be 8 characters long. Let's use INSCUST and press Enter




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

dependencies.  Review this application.



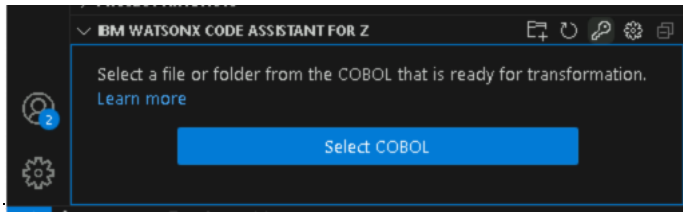

**Congratulations, you've refactored your first application!**

## Code Explanation in VS Code

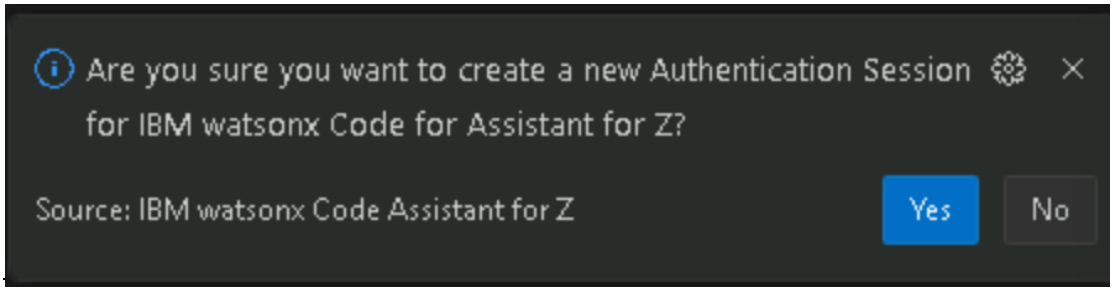
~~Pre-work:~~

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

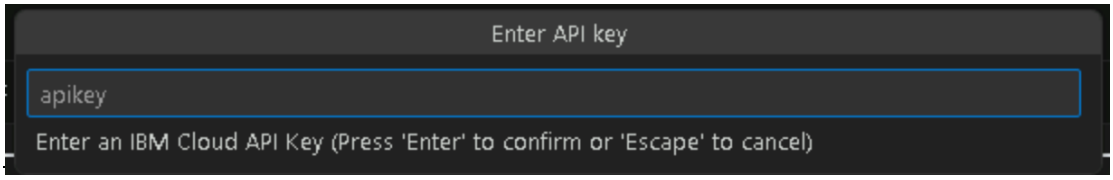
⇒ ~~In the bottom left, there is a section called IBM WATSONX CODE ASSISTANT FOR Z, and there is a small key icon. Click it~~



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



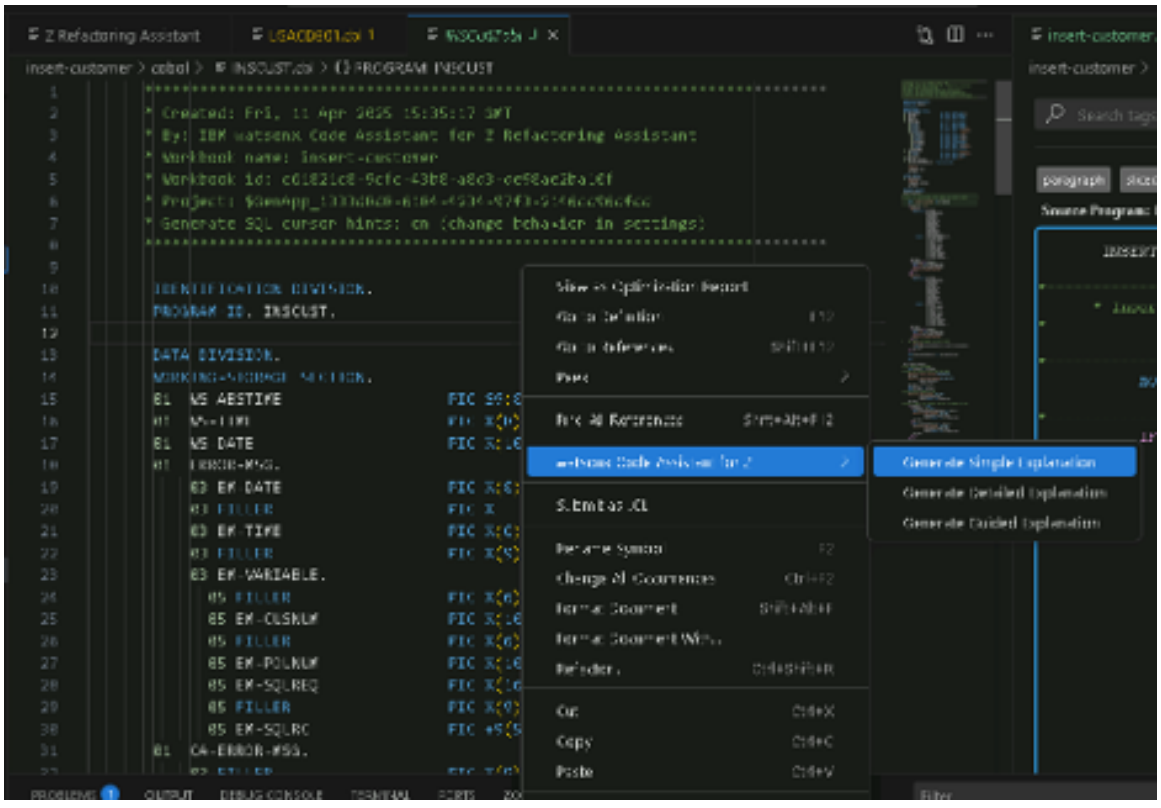

⇒ ~~Enter the API key in the popup at the top~~




⇒ ~~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 cbl 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.

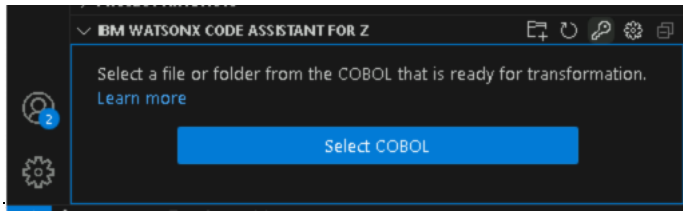
# 8 Transformation – Converting Cobol to Java

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.

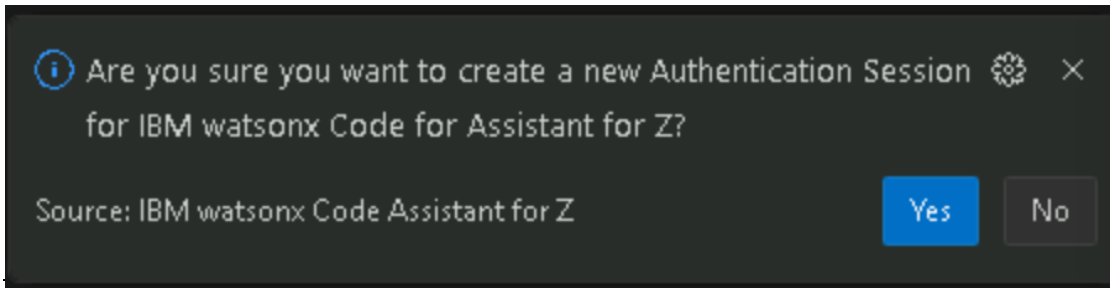
~~Pre-work:~~

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

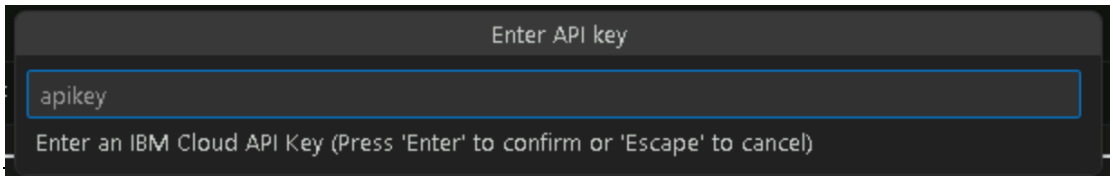
⇒ ~~In the bottom left, there is a section called IBM WATSONX CODE ASSISTANT FOR Z, and there is a small key icon. Click it~~




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



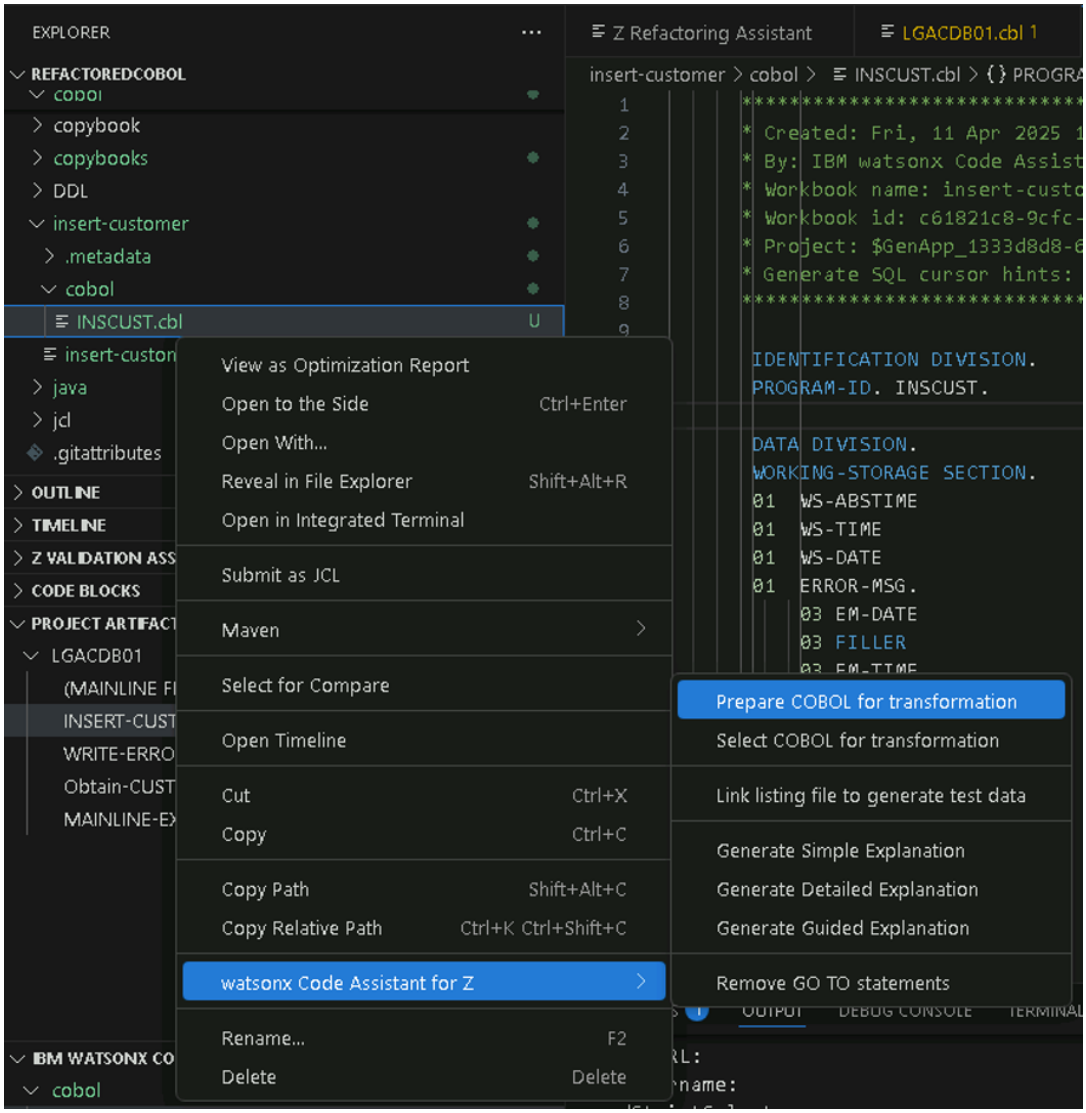

⇒ ~~Enter the API key in the popup at the top~~




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

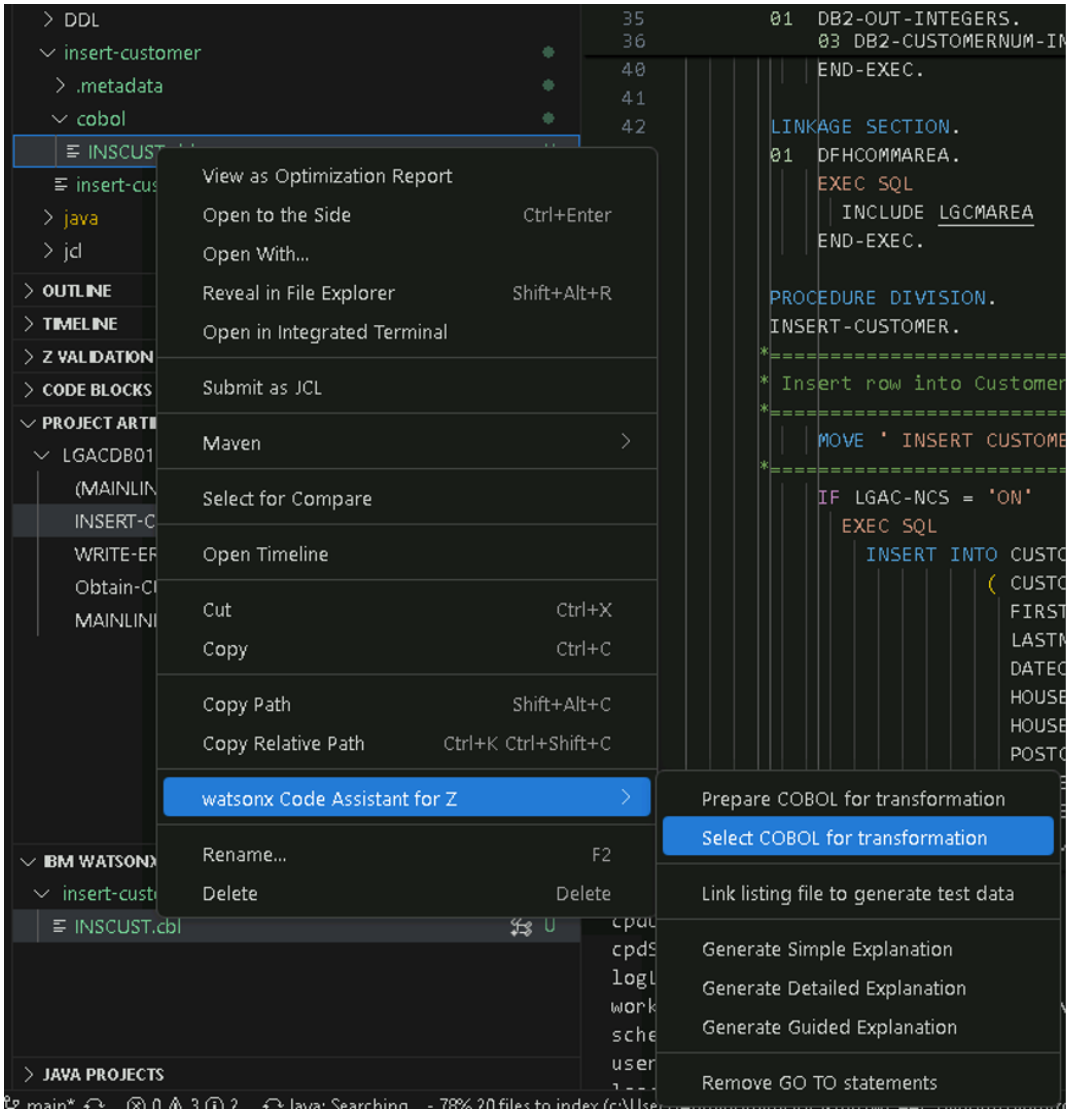
Now we're going to work on transforming a portion of the code from COBOL to Java. The ideal scenario for migration is to issolate a service from the application and transform that to Java, so we're going to work with the refactored piece of code from above.

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

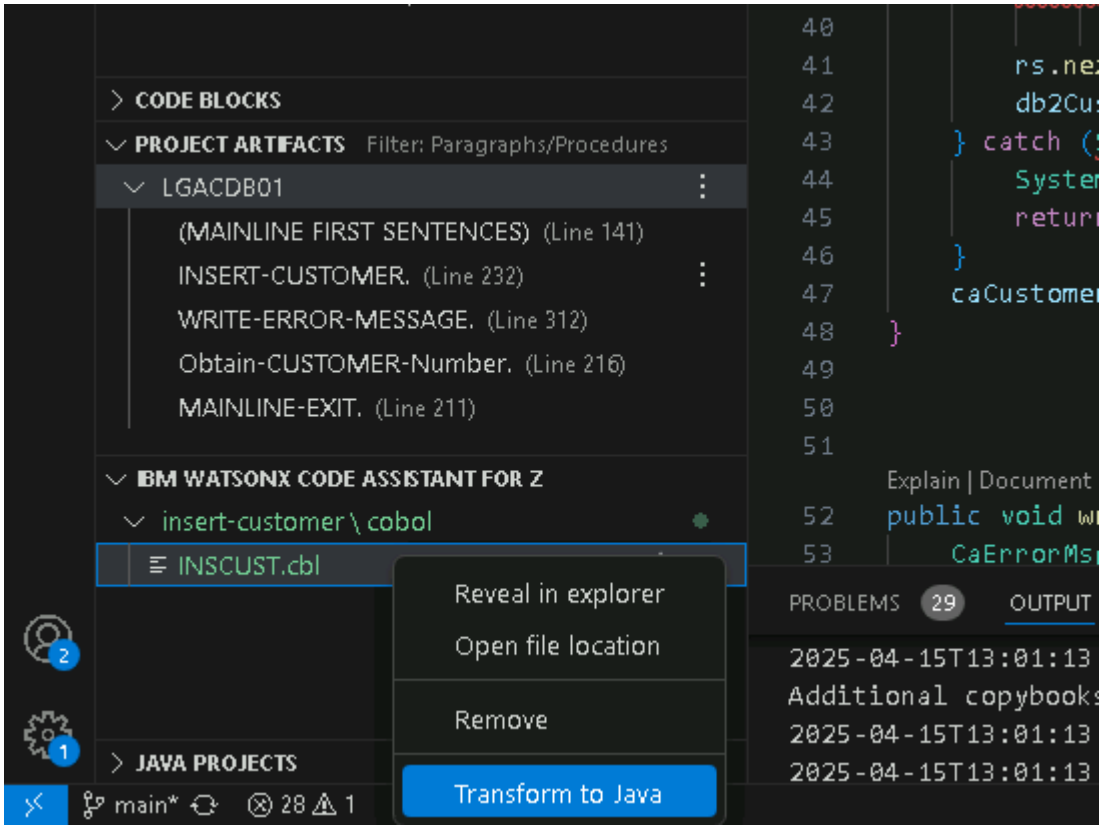



⇒ This will sync the code to IBM cloud database. Once this is complete, you can right click the same application again and select:

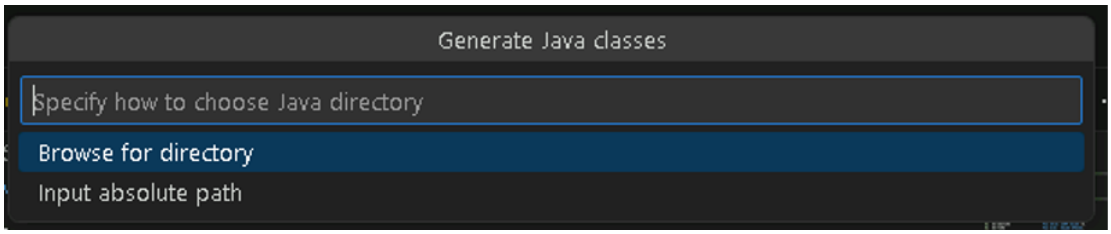
**watsonx Code Assistant for Z > Select COBOL for transformation**



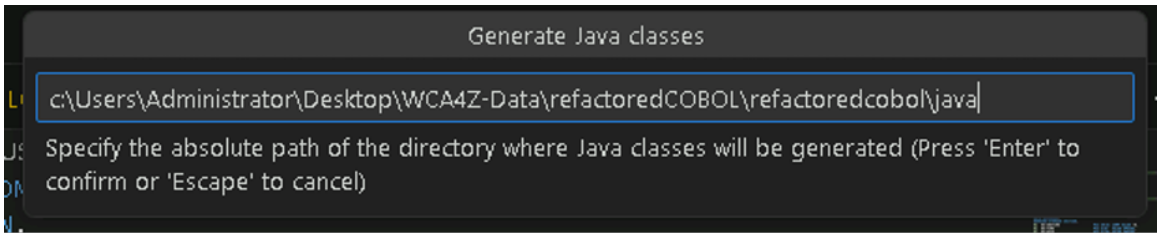

⇒ You'll see the application selected on the bottom left. Select the application and select **Generate Java classes.**




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

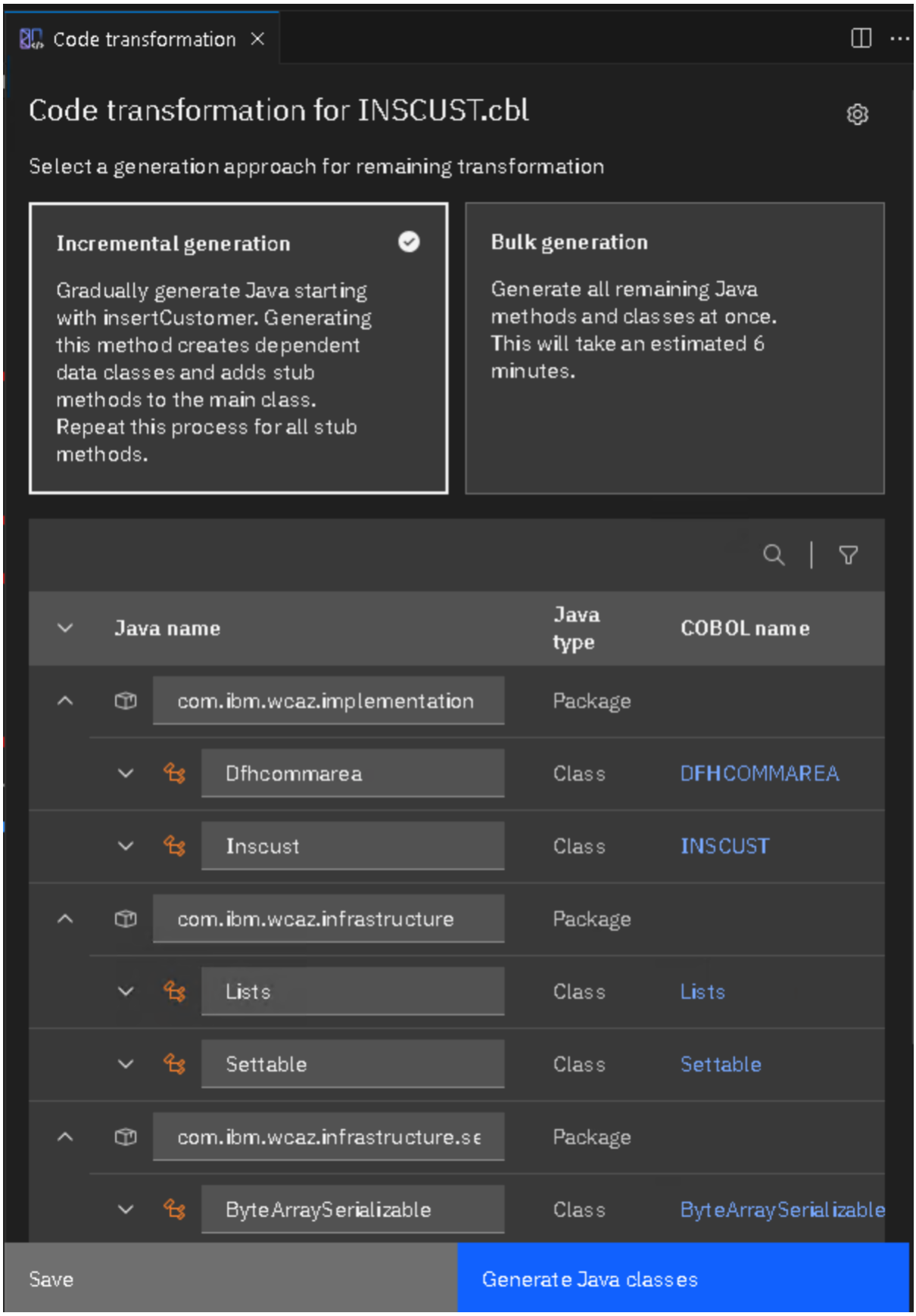



⇒ This will populate with the current directory, add **/java** to the end of the path

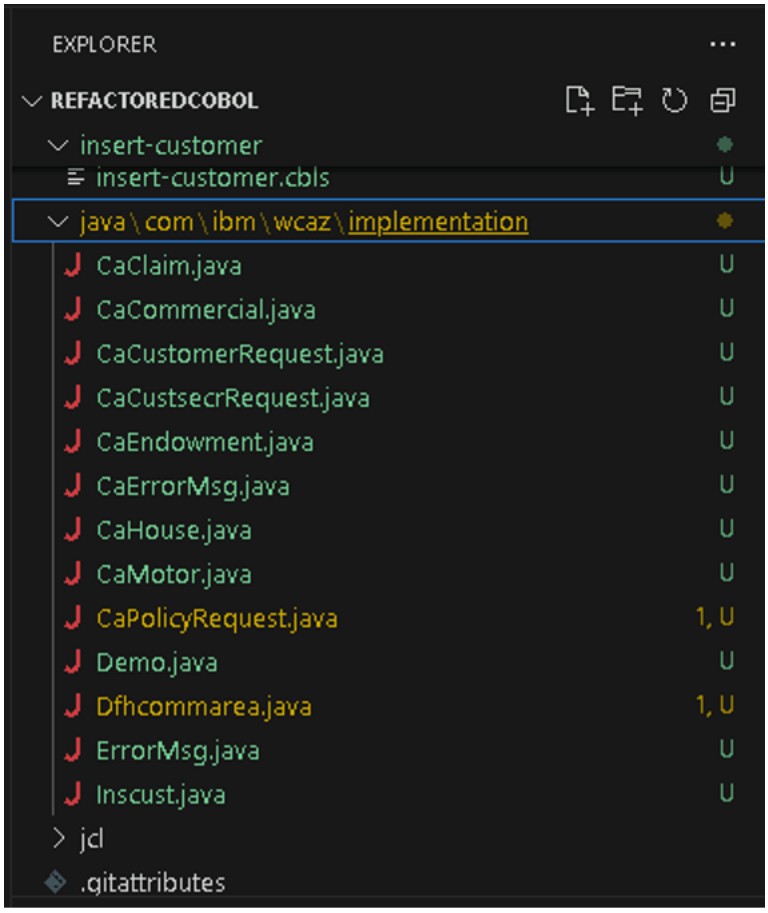



⇒ You'll see a new tab open with a list of all the classes that will be generated. Here you can change the name of the Java classes to be generated, though we recommend taking the suggested names and refactor after that fact if desired.

⇒ Select Incremental generation


Code transformation
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.

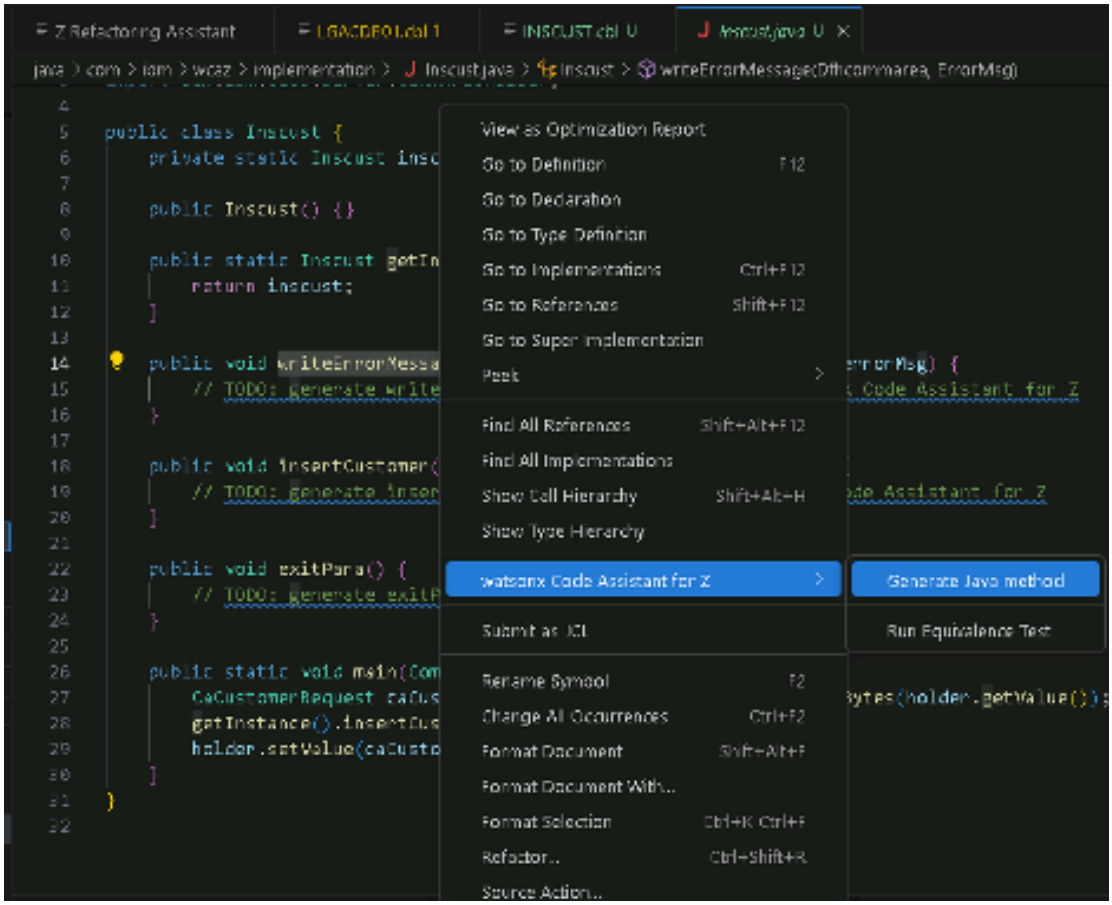



⇒ This process created all the Java objects and classes which map to the COBOL objects. However, none of the actual logic has been migrated yet. Our job next is to migrate the actual paragraph logic. Java method stubs are generated, with a comment saying that the method still needs to be generated. For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

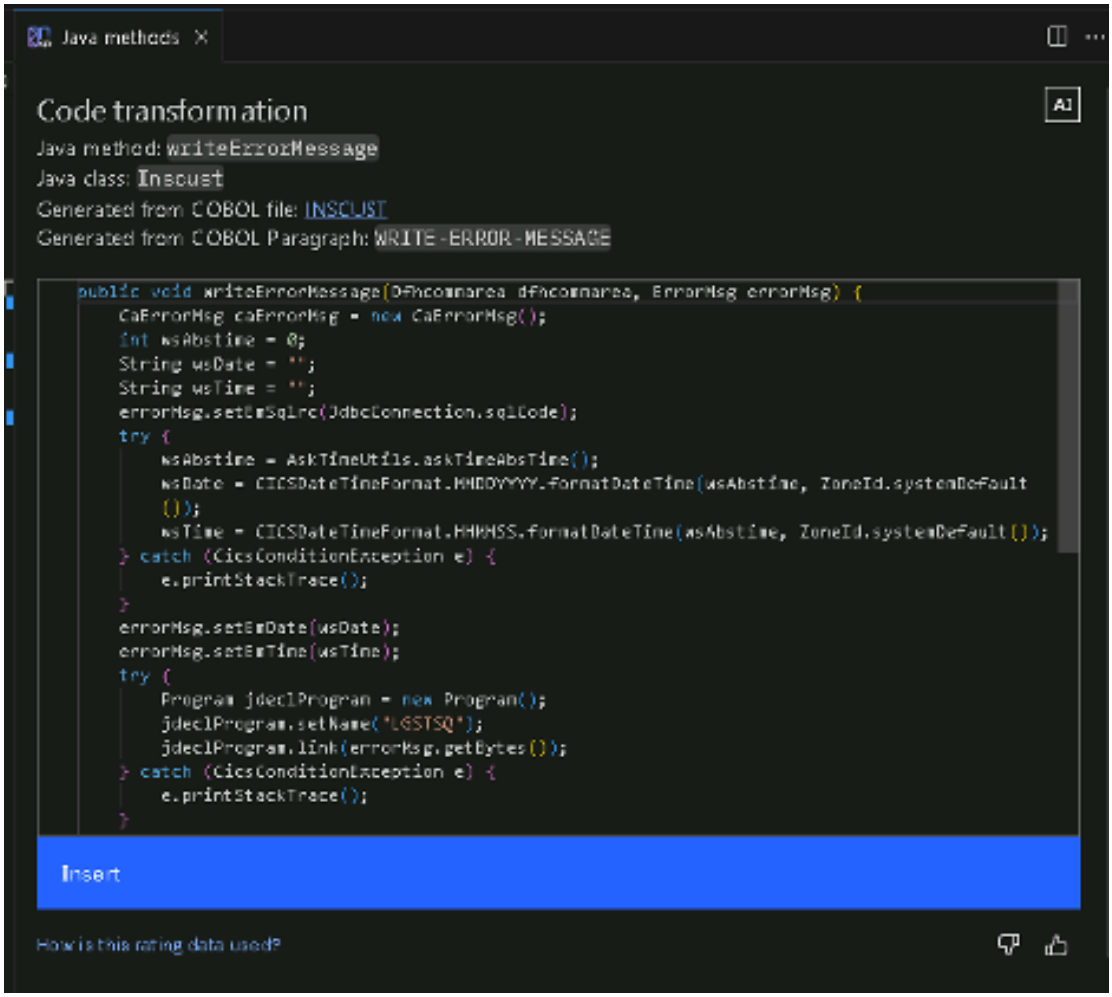
```
17
18      public void insertCustomer(CaCustomerRequest caCustomerRequest) {
19          // TODO: generate insertCustomer() method with IBM watsonx Code Assistant for Z
20      }
21
```

⇒ 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**



⇒ 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.




⇒ Click Insert.

⇒ Repeat the above steps for any other method which need to be migrated. A quick way to find all the instances if 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 – vandepol@ca.ibm.com