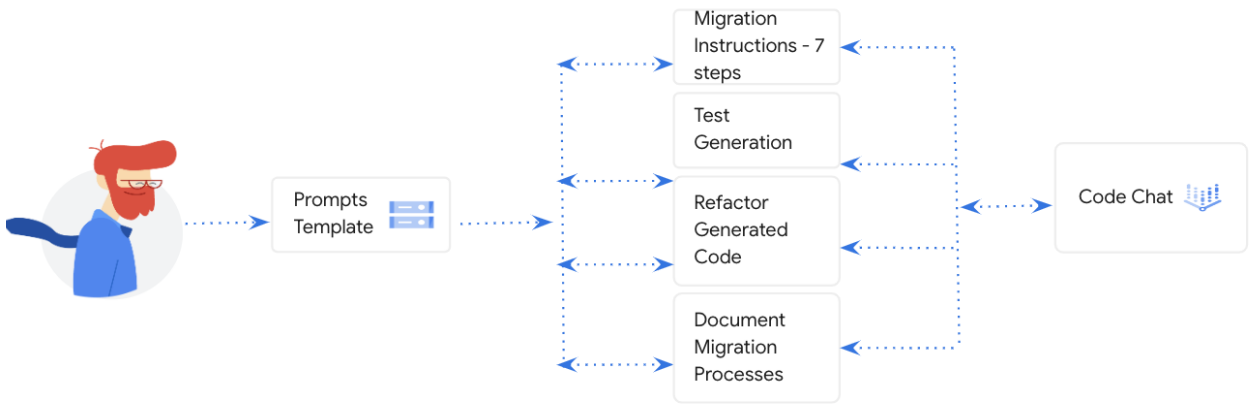
Demo Guide

Lei Pan 11/6

Example - Migrate Code from COBOL to Java with Prompt Templates



API Demo

Step 1: Enable Vertex AI in your GCP project

1. Go to the Vertex AI Console: https://console.cloud.google.com/vertex-ai/.
2. Click the Enable Vertex AI button.
3. Follow the on-screen instructions to enable Vertex AI.

Step 2: Install libraries

Step 4: Run code

- All the steps are in the notebook
- Refer to the videos for more details

Notes:

Here are all the prompts that's used in the notebook:

```
prompt 1:

You are great at migrating code from COBOL to Java. Here is the COBOL code:
IDENTIFICATION DIVISION.
```

```
PROGRAM-ID.  CPSEQFR.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

    SELECT INFILE ASSIGN  TO 'INFILE1'

           FILE STATUS IS INPUT-FILE-STATUS.

    SELECT OUTFILE ASSIGN TO 'OUTFILE1'

        FILE STATUS IS OUTPUT-FILE-STATUS.

DATA DIVISION.

FILE SECTION.

FD  INFILE

    LABEL RECORDS ARE STANDARD

    DATA RECORD IS INPUT-RECORD

    RECORD CONTAINS 40 CHARACTERS

    RECORDING MODE IS F

    BLOCK CONTAINS 0 RECORDS.

01  INPUT-RECORD.

    05 INPUT-FIRST-10      PIC X(10).

    05 INPUT-LAST-30       PIC X(30).



FD  OUTFILE

    LABEL RECORDS ARE STANDARD

    DATA RECORD IS OUTPUT-RECORD

    RECORD CONTAINS 40 CHARACTERS

    RECORDING MODE IS F
    BLOCK CONTAINS 0 RECORDS.

01  OUTPUT-RECORD.

    05 OUTPUT-FIRST-30     PIC X(30).

    05 OUTPUT-LAST-10      PIC X(10).



WORKING-STORAGE SECTION.

01  WorkAreas.

    05  INPUT-FILE-STATUS  PIC X(02).

        88  GOOD-READ      VALUE '00'.

        88  END-OF-INPUT   VALUE '10'.

    05  OUTPUT-FILE-STATUS PIC X(02).

        88  GOOD-WRITE     VALUE '00'.

    05  RECORD-COUNT       PIC S9(5) COMP-3.



PROCEDURE DIVISION.

    OPEN INPUT INFILE

    IF NOT GOOD-READ

        DISPLAY 'STATUS ON INFILE OPEN: ' INPUT-FILE-STATUS

        GO TO END-OF-PROGRAM

    END-IF

    OPEN OUTPUT OUTFILE

    IF NOT GOOD-WRITE

        DISPLAY 'STATUS ON OUTFILE OPEN: ' OUTPUT-FILE-STATUS

    END-IF

    PERFORM UNTIL END-OF-INPUT
```

```
             READ INFILE
             IF GOOD-READ
                 MOVE INPUT-FIRST-10 TO OUTPUT-LAST-10
                 MOVE INPUT-LAST-30 TO OUTPUT-FIRST-30
                 WRITE OUTPUT-RECORD
                 IF GOOD-WRITE
                      ADD 1 TO RECORD-COUNT
                 ELSE
                     DISPLAY 'STATUS ON OUTFILE WRITE: '
                             OUTPUT-FILE-STATUS
                     GO TO END-OF-PROGRAM
                 END-IF
             END-IF
         END-PERFORM
         .
     END-OF-PROGRAM.
         DISPLAY 'NUMBER OF RECORDS PROCESSED: ' RECORD-COUNT
         CLOSE INFILE
         CLOSE OUTFILE
         GOBACK.
```

Please covert it to Java by following the prompt instructions below to do that:

Step 1: Generate Java classes from COBOL data structures. Each COBOL data structure should correspond to a Java class. Ensure proper data type mapping and encapsulation.

```
Step 2: Translate COBOL file input/output operations to Java file handling operations

Step 3: Migrate COBOL business logic to Java. Convert COBOL procedures, paragraphs,
and sections to Java methods. Ensure equivalent functionality

Step 4: Convert COBOL conditional statements (IF, ELSE, etc.) to Java if-else
statements and loops (PERFORM, etc.) to Java loops (for, while, etc.). Ensure logical
equivalence

Step 5: Replace COBOL-specific functions and operations with Java equivalents. This
includes arithmetic operations, string manipulations, and date/time functions.

Step 6: Generate Java constants from COBOL copybooks. Each COBOL constant should be
converted to an equivalent Java constant

Step 7: Update COBOL variable names and identifiers to follow Java naming conventions.
Ensure proper camelCase or PascalCase formatting


prompt 2:

Generate a few unit test cases and data to validate the migrated Java code. Ensure
that the Java code functions correctly and produces the same results as the original
COBOL code.

prompt 3:

Refactor the generated Java code to adhere to Java best practices, coding standards,
and design patterns. Optimize the code for performance and maintainability
```

```
prompt 4:

Generate documentation for the code migration process. Include details of the changes
made, data type mappings, and any issues encountered during migration




Optional:

Step 8: Translate COBOL database interactions to Java using JDBC (Java Database
Connectivity). Migrate SQL queries and database connections



Step 9: Implement error handling in Java for equivalent COBOL error handling
mechanisms. Add exception handling in Java to handle exceptions and errors.
```