JCL1

JCL: Making Things Happen

Condition Code: Hero

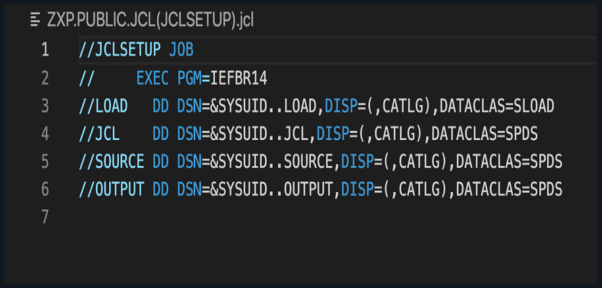
15 steps 90 minutes

THE CHALLENGE

You've seen some JCL already, but we haven't really gone in depth. In these challenges, we'll learn a little more about what JCL is used for, why it is important in a Z environment, and how you can take those skills even further. JCL is an essential part of making things happen in z/OS and getting comfortable with the concepts and syntax will let you power through many challenges you might face as you explore.

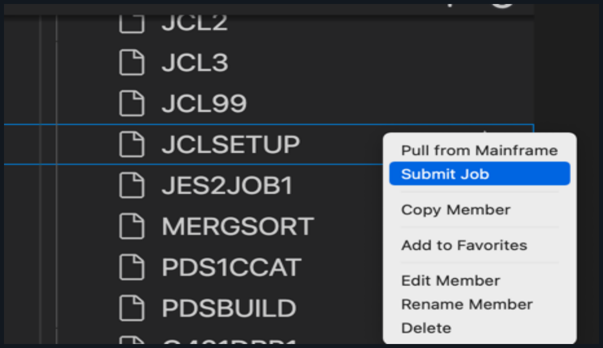
BEFORE YOU BEGIN

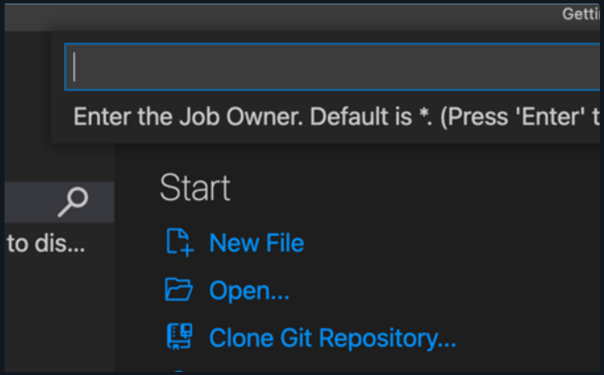
You should have completed the FILES1 challenge, all about Data Sets and Members. If you've got those concepts down, you're all set to continue with the steps in this challenge.



ZXP.PUBLIC.JCL(JCLSETUP).jcl

```
1  //JCLSETUP JOB
2  //      EXEC PGM=IEFBR14
3  //LOAD   DD DSN=&SYSUID..LOAD,DISP=(,CATLG),DATACLAS=SLOAD
4  //JCL    DD DSN=&SYSUID..JCL,DISP=(,CATLG),DATACLAS=SPDS
5  //SOURCE DD DSN=&SYSUID..SOURCE,DISP=(,CATLG),DATACLAS=SPDS
6  //OUTPUT DD DSN=&SYSUID..OUTPUT,DISP=(,CATLG),DATACLAS=SPDS
7
```





Enter the Job Owner. Default is *. (Press 'Enter' t

1. LOAD IT UP

Look in ZXP.PUBLIC.JCL for a member named JCLSETUP. This is a fairly simple job that will allocate a few new data sets you need for this, and other challenges. Line #4 creates your own JCL data set.

You can see in Line #3, it mentions &SYSUID..LOAD. The Ampersand (&) with SYSUID after it is what's known as a *Symbolic*, and when the system sees that, it will replace &SYSUID with your userid.
This means everyone can use the same job, and it will automatically replace &SYSUID with their userid.
How convenient!

2. SUBMIT IT

Right-click on that job and select Submit Job.

A note on the word "Job": JCL is used to describe to the system exactly what you'd like it to do. The task which we hand over to the system is known as a "Job", and the component of z/OS that accepts, processes, and manages the output of these jobs is known as JES, the Job Entry Subsystem.

So, for this challenge, we've sent a job off to JES for it to process the tasks that we looked at in Step 1.

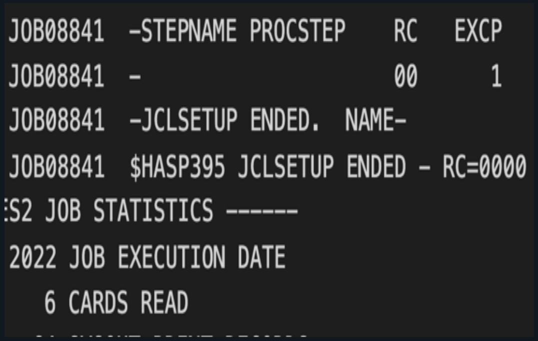
3. FILTER AND FIND IT

You should already have a profile under JOBS on the left side of VS Code. Click on the magnifying glass (🔍) to the right of it.

Enter your userid for the Job Owner
Enter an asterisk (*) for the Job Prefix
Hit Enter (blank, no data) for Job Id Search.

You can redo this filter by clicking on the magnifying glass again and selecting Owner/Prefix or Job Id. You should be able to find the job you just submitted in here. We'll dig into that next.

IBM Z

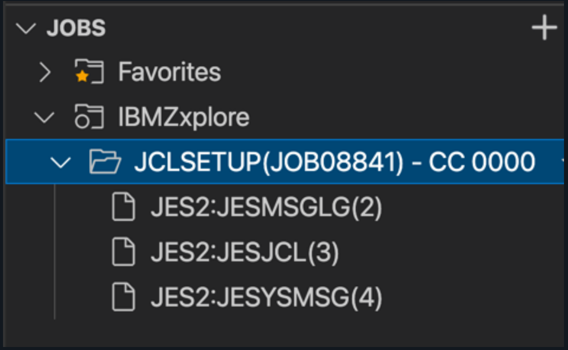


```
JOB08841  -STEPNAME PROCSTEP    RC   EXCP
JOB08841  -                     00      1
JOB08841  -JCLSETUP ENDED.  NAME-
JOB08841  $HASP395 JCLSETUP ENDED - RC=0000
S2 JOB STATISTICS ------
2022 JOB EXECUTION DATE
   6 CARDS READ
```

4. YOU GOT A ZERO. PERFECT!

Open up the triangle "twistie" next to the JCLSETUP job you just submitted. There will probably be other jobs in there as well, but we're specifically looking for JCLSETUP. If you submitted it more than once, find the one with **CC 0000** to the right.

You'll also see this number in the JESMSGLG member once you open the twistie. A condition code (CC) of zero means everything ran as expected, with no errors, so that's good! If you got any other number, then there's probably something worth investigating and possibly fixing.

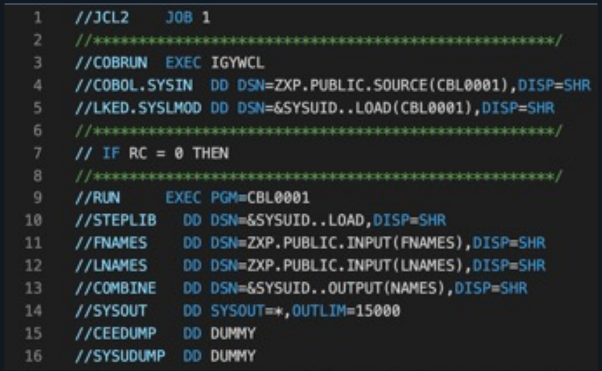


```
JOBS
  Favorites
  IBMZxplore
    JCLSETUP(JOB08841) - CC 0000
      JES2:JESMSGLG(2)
      JES2:JESJCL(3)
      JES2:JESYSMSG(4)
```

5. JUMP RIGHT TO IT

You may have noticed that after submitting JCL, a little message pops up in the bottom right corner of VS Code. Rather than digging through your JOBS output, you can usually just click on that message, and it will take you right to the output.

A job will start out in ACTIVE while it's being run. You can refresh the status of a job by closing and then re-opening the twistie to the left of the job name.



```
 1  //JCL2     JOB 1
 2  //**************************************************/
 3  //COBRUN  EXEC IGYWCL
 4  //COBOL.SYSIN  DD DSN=ZXP.PUBLIC.SOURCE(CBL0001),DISP=SHR
 5  //LKED.SYSLMOD DD DSN=&SYSUID..LOAD(CBL0001),DISP=SHR
 6  //**************************************************/
 7  // IF RC = 0 THEN
 8  //**************************************************/
 9  //RUN     EXEC PGM=CBL0001
10  //STEPLIB   DD DSN=&SYSUID..LOAD,DISP=SHR
11  //FNAMES    DD DSN=ZXP.PUBLIC.INPUT(FNAMES),DISP=SHR
12  //LNAMES    DD DSN=ZXP.PUBLIC.INPUT(LNAMES),DISP=SHR
13  //COMBINE   DD DSN=&SYSUID..OUTPUT(NAMES),DISP=SHR
14  //SYSOUT    DD SYSOUT=*,OUTLIM=15000
15  //CEEDUMP   DD DUMMY
16  //SYSUDUMP  DD DUMMY
```

6. KICKING OFF SOME COBOL

Copy JCL2 from ZXP.PUBLIC.JCL to your own ZXXXXX.JCL data set, and then open it from your JCL data set. You may need to close and re-open your DATA SETS triangle to refresh the view, so your JCL shows up. This JCL is used to compile and run some COBOL code. After compiling, it will put the resulting program in your LOAD data set.

In JCL, statements that define where data is coming from or going to are known as Data Definition Statements, or simply, DD Statements.

You can see the input data set (the source) on Line 4, and where it's putting the output on Line 5.

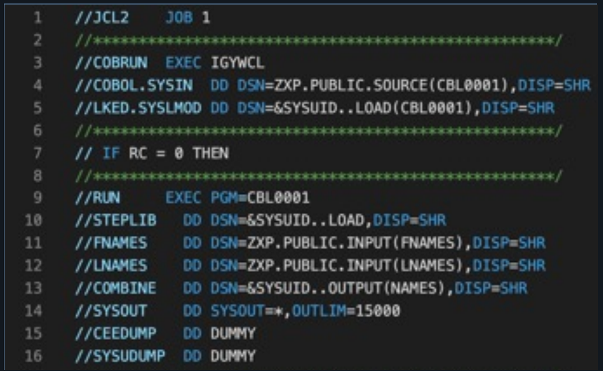
Reading further, if it gets a Return Code of 0 (because everything went without any problems) it will then run the program. Make sense so far? We're using JCL to compile, and then run some code.

"WHY ARE JES AND JCL IMPORTANT? WHY CAN'T I JUST RUN PROGRAMS?"

When you submit JCL, it goes to the Job Entry Subsystem (shortened to JES). JES looks through the JCL you submitted and gathers all of the resources needed to accomplish the task. On a heavily-loaded system, it may be necessary to prioritize some jobs lower than others so that important work gets done faster.

Think of JCL as the order that a waiter writes up, and JES as the kitchen staff that looks at the order and decides how they're going to handle it. The L in JCL stands for Language, but it really isn't a programming language as much as it is a way for us to effectively describe tasks to the system.

Everything else that shows up in the job output (from step 4) is information about how the job ran. As you can see, there's a TON of information in here.



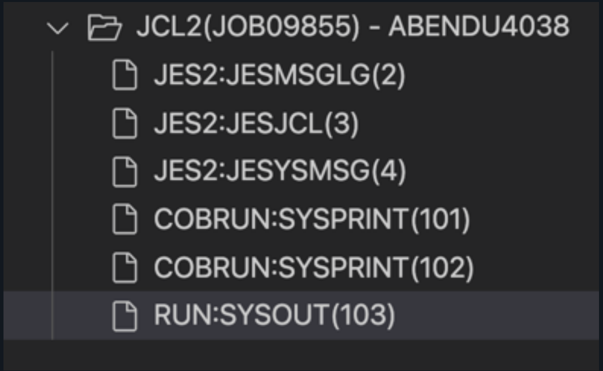
```
1   //JCL2     JOB 1
2   //*****************************************************/
3   //COBRUN   EXEC IGYWCL
4   //COBOL.SYSIN  DD DSN=ZXP.PUBLIC.SOURCE(CBL0001),DISP=SHR
5   //LKED.SYSLMOD DD DSN=&SYSUID..LOAD(CBL0001),DISP=SHR
6   //*****************************************************/
7   // IF RC = 0 THEN
8   //*****************************************************/
9   //RUN      EXEC PGM=CBL0001
10  //STEPLIB    DD DSN=&SYSUID..LOAD,DISP=SHR
11  //FNAMES     DD DSN=ZXP.PUBLIC.INPUT(FNAMES),DISP=SHR
12  //LNAMES     DD DSN=ZXP.PUBLIC.INPUT(LNAMES),DISP=SHR
13  //COMBINE    DD DSN=&SYSUID..OUTPUT(NAMES),DISP=SHR
14  //SYSOUT     DD SYSOUT=*,OUTLIM=15000
15  //CEEDUMP    DD DUMMY
16  //SYSUDUMP   DD DUMMY
```

7. RUN, CODE, RUN

After compiling the COBOL code, it will run the code (Line 9) and tell it where the find the input data sets (Lines 11-12) as well as where to put the output (Line 13).

The name of the DD statement is what comes directly after the double slashes, so FNAMES and LNAMES, for example.

Any lines starting with //* are comments and are ignored by JES. Commented lines are helpful for providing informational information, or holding lines of code we might use later, but don't need right now.



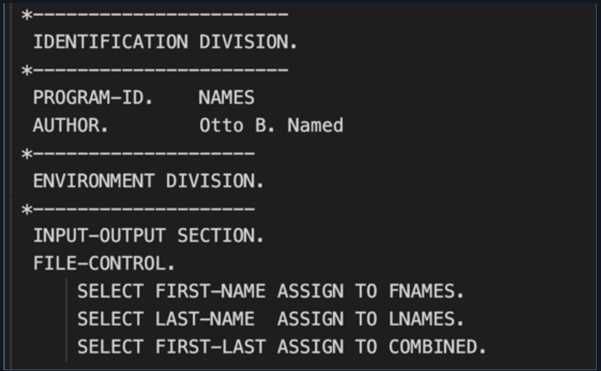
```
JCL2(JOB09855) - ABENDU4038
    JES2:JESMSGLG(2)
    JES2:JESJCL(3)
    JES2:JESYSMSG(4)
    COBRUN:SYSPRINT(101)
    COBRUN:SYSPRINT(102)
    RUN:SYSOUT(103)
```

8. THEY CAN'T ALL BE ZEROES

Submit JCL2 from your ZXXXXX.JCL data set and then look at the output, using what you learned from steps 1-5. We got an ABEND (short for Abnormal End) so something didn't go right.

But don't fret! We'll get to the bottom of this.

In the next step, we'll look at the COBOL code and see how the actual COBOL code compares with the JCL code we're using to compile and kick it off.



```
*-----------------------
 IDENTIFICATION DIVISION.
*-----------------------
 PROGRAM-ID.    NAMES
 AUTHOR.        Otto B. Named
*--------------------
 ENVIRONMENT DIVISION.
*--------------------
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FIRST-NAME ASSIGN TO FNAMES.
     SELECT LAST-NAME  ASSIGN TO LNAMES.
     SELECT FIRST-LAST ASSIGN TO COMBINED.
```

9. COMPARE THE CODE

The JCL statement beginning **//COBOL.SYSIN** points to the COBOL code we want to compile, so let's start there. Open up that code and start looking at the FILE-CONTROL area.

This is where we get the names for the program which we need to reference in our JCL. For example, FIRST-NAME is what we reference in the COBOL code, and that is assigned (or connected to) the FNAMES DD statement in the JCL.

Open up the JCL and the COBOL code, look at the output, and you should be able to figure out what **very** simple change needs to be made in order for everything to line up. It actually may help to draw everything out on a piece of paper.

When you've fixed the problem ***in JCL2***, it should run with a CC=0, and you *will* find **the correct output in the correct member** of your OUTPUT data set. This one might take some detective work

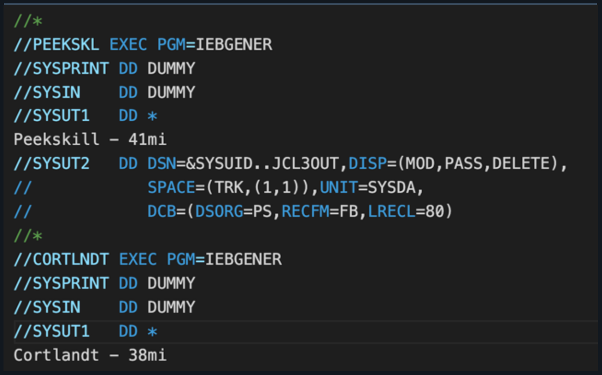
"WHAT DOES COMPILE MEAN? WHAT IS COBOL?"

COBOL is a programming language used in many financial, healthcare, and government institutions. Its high degree of mathematical precision and straightforward coding methods make it a natural fit when programs need to be fast, accurate, and easy to understand.

Code that gets written by humans needs to be turned into *Machine Code* for it to run as a program. Compiling is a step that performs this transformation. Our JCL has two main steps, compiling the source code into machine code, and then running the program (machine code).

This particular program also requires two input files, and writes to an output file, so we will specify those files (data sets) in the JCL as well.

IBM Z



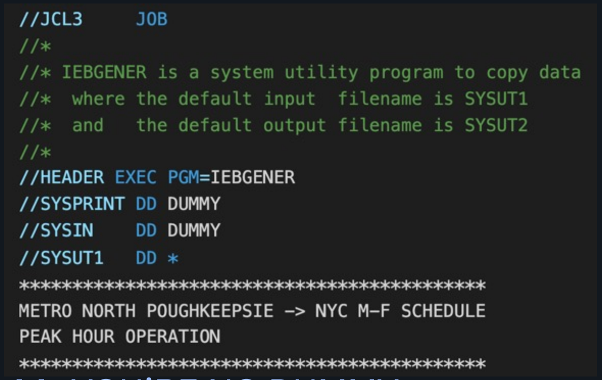
```
//*
//PEEKSKL EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT1   DD *
Peekskill - 41mi
//SYSUT2   DD DSN=&SYSUID..JCL3OUT,DISP=(MOD,PASS,DELETE),
//            SPACE=(TRK,(1,1)),UNIT=SYSDA,
//            DCB=(DSORG=PS,RECFM=FB,LRECL=80)
//*
//CORTLNDT EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT1   DD *
Cortlandt - 38mi
```

10. ALL ABOARD

Copy JCL3 from ZXP.PUBLIC.JCL into your own JCL data set. Load it up and take a look at what's inside. You'll see this JCL contains a number of steps, with each one using the IEBGENER program to direct an ingredient into a sequential data set.

There's a header, some station information for the peak stops between Poughkeepsie, NY and Grand Central Terminal in NYC, then some text about operating hours.

Looks pretty straightforward... let's see what the tricky part is.

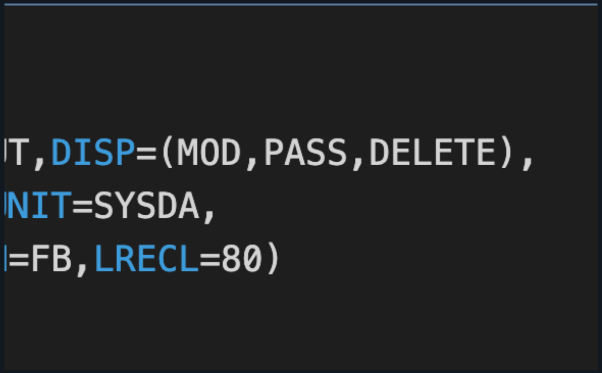


```
//JCL3     JOB
//*
//* IEBGENER is a system utility program to copy data
//*  where the default input  filename is SYSUT1
//*  and   the default output filename is SYSUT2
//*
//HEADER EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT1   DD *
********************************************
METRO NORTH POUGHKEEPSIE -> NYC M-F SCHEDULE
PEAK HOUR OPERATION
********************************************
```

11. YOU'RE NO DUMMY

You may have also noticed lots of mentions of DUMMY. Don't worry, this JCL isn't calling anyone names, it's just a way of saying "This parameter is required, but we aren't going to do anything with it, so it doesn't matter".

We use it here because the program we're running in each step (IEBGENER) requires an input statement, and requires a SYSPRINT statement, but we're not going to be making use of it, so DUMMY is a way of saying "It doesn't matter, don't waste your time setting this up"



```
T,DISP=(MOD,PASS,DELETE),
NIT=SYSDA,
=FB,LRECL=80)
```

12. A FRIENDLY DISPOSITION

In every piece of JCL we've used so far, we've encountered some sort of DISP (disposition) parameter. DISP parameters are used to describe how JCL should use or create a data set, and what to do with it after the job completes.

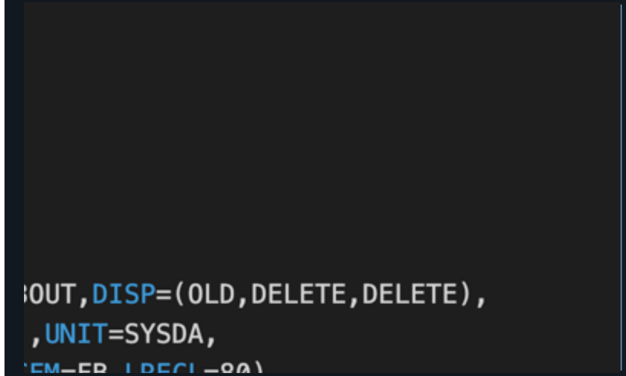
A standard DISP parameter has three parts. The first parameter is the status, which can be any of the following:

NEW Create a new data set
SHR Re-use an existing data set, and let other people use it if they'd like
OLD Re-use an existing data set, but don't let others use it while we're using it
MOD For sequential data sets only. Re-use an existing data set, but only append new records to the bottom of it. If no data set exists, create a new one.

"JOB OUTPUT? DD STATEMENTS? HELP ME UNDERSTAND, PLEASE"

When a job runs, it produces output. This might include the data you're looking for, reasons a job didn't run successfully, or maybe just information about the system and the steps it took to make the work happen. There's a lot of information in here you probably *don't* need, but it's always better to have it than to be confused about the state of an important job.

A big part of your JCL is the DD statements, which specify which data sets (and members) to use for the input and output of the job you're submitting to the system. The DD in DD Statement stands for Data Definition, and they start with //DD. They will specify the name of the data set (or member), whether it already exists or needs to be created (known as the disposition), where the output should go, how much space it should take up, and a number of other variables.



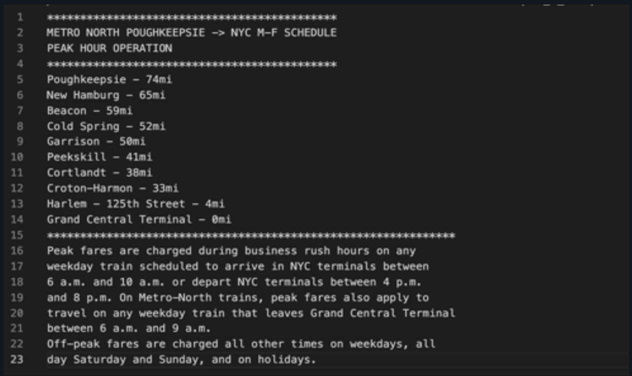
```
OUT,DISP=(OLD,DELETE,DELETE),
,UNIT=SYSDA,
```

13. WHAT'S YOUR STATUS

Field 2 of the DISP parameter describe what should happen to the data set in the case of a normal completion, and the third field is what should happen to it in the case of a failure.

There are a number of values we can use here, but for this challenge, you only need to know about the following:

DELETE Erase it from storage completely
CATLG Record the data set so we can use it later
PASS After this step completes, hold on to it so steps that come after this one can use it



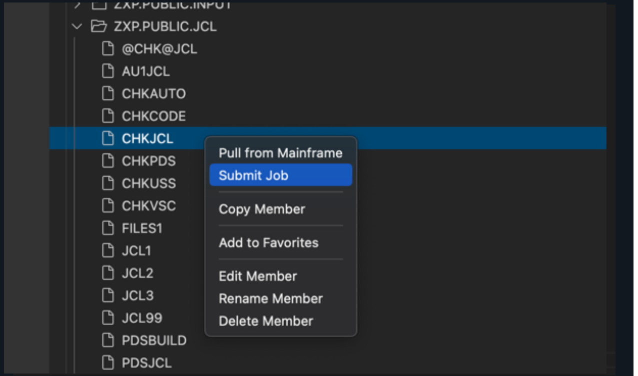
```
*******************************************
METRO NORTH POUGHKEEPSIE -> NYC M-F SCHEDULE
PEAK HOUR OPERATION
*******************************************
Poughkeepsie - 74mi
New Hamburg - 65mi
Beacon - 59mi
Cold Spring - 52mi
Garrison - 50mi
Peekskill - 41mi
Cortlandt - 38mi
Croton-Harmon - 33mi
Harlem - 125th Street - 4mi
Grand Central Terminal - 0mi
**************************************************************
Peak fares are charged during business rush hours on any
weekday train scheduled to arrive in NYC terminals between
6 a.m. and 10 a.m. or depart NYC terminals between 4 p.m.
and 8 p.m. On Metro-North trains, peak fares also apply to
travel on any weekday train that leaves Grand Central Terminal
between 6 a.m. and 9 a.m.
Off-peak fares are charged all other times on weekdays, all
day Saturday and Sunday, and on holidays.
```

14. RIGHT ON TIME

Submit the JCL3 job and look at the output. There are two edits you need to make in order for this job to run 100% correctly, and produce a full listing of 10 stops, from Poughkeepsie to Grand Central Terminal, plus the information at the top and bottom of the data set.

You should also have no repeat stops. If Beacon or Cortlandt is listed in there twice, something needs fixing.

When completed, you should have the full output in your JCL3OUT sequential data set, totaling 23 lines (records).



15. WHO KNEW?

Now submit CHKJCL1 in ZXP.PUBLIC.JCL to validate correct output from JCL2 and JCL3, and hope to see completion code (CC) of 0.
CHKJCL1 returns CC=0127? Double-check and adjust your work for JCL2 (reread Step 9) and JCL3, resubmit those jobs if needed. Recheck by running CHKJCL1 again until you get CC=0. You may want to delete the JCL3OUT output data set before re-submitting JCL3.

You've accomplished a lot, and understand the requirement for following the details ... let's try some UNIX now!

NICE JOB! LET'S RECAP

JCL can seem a little tricky, and maybe even a little unnecessary at first. We're not used to using code to start programs, we usually just double-click on them and they run!

However, once you start getting into the types of applications that keep Z systems busy 24/7, you start to build an appreciation for the precision and power that the structure gives you. Suffice to say, JCL is a necessary skill for any true Z professional.

NEXT UP...

Unix System Services (USS)