Introduction to Modern Application Development
Persistent Computer Institute

Lecture 8
Command Line _ Practice Questions - Part 2

(Refer Slide Time: 00:11)

Hello everyone, welcome to the second question session of the second week of the course on introduction to modern applications development. These questions are for your practice, we hope you use them to understand the material that has been presented so far. There are about ten of them and we will go through each of them slowly.

1. Consider the command line program in the first video session of the second week. **From the moment the command line arguments are presented on the command line, how many times is the command line information transferred from one point of the program to another before it is first used in processing?**

   Answer: 2 times (or 3 times if you also count do setup).

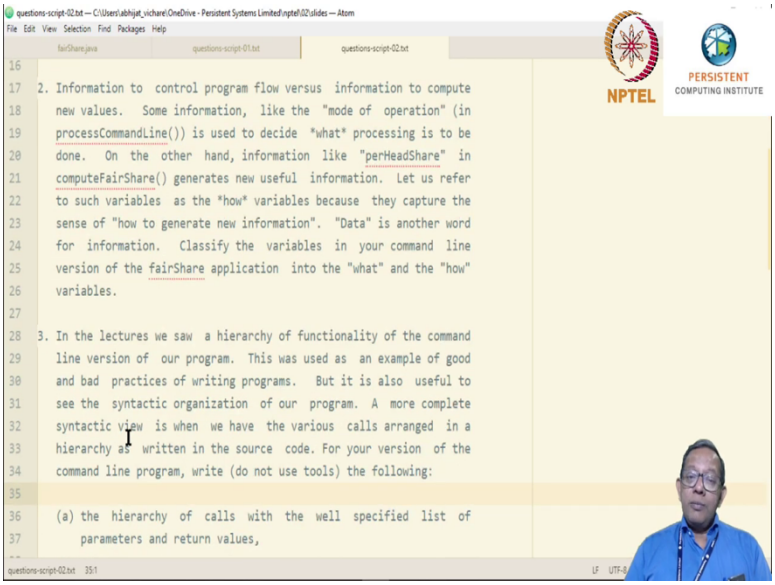
   a. In the first step it arrives at *main* via parameters an array of strings called as args.
   b. Then it arrives at *doSetup* as parameters again an array of strings called args.
   c. Finally, it arrives at *processCommandLine* method again as an array of strings called args.

   Within the method *processCommandLine* it is used or processed to obtain, for example, the mode in which the application is being run, like register mode, expense mode… etc.

Therefore, the number of times it is transferred from the moment it is presented on the command line to the point of its actual use is twice (or thrice if you consider do setup). Note that it would be useful to be explicit about *how you count*. For example, you should be explicit if you do count the *doSetup*. But also note that it may possibly be disagreeable to some, but then it is a very positive thing as otherwise it might result in misunderstanding. If, on the other hand, you just ignore it and do not write it, then it is possible that I may consider *doSetup* as an important part and see that you have not counted it, and therefore penalize. This kind of a thing occurs very much in practice between human beings. It is therefore useful to be explicit and clear about the way you answer things.

If there are any assumptions, for example if you assume that do setup should not be counted, then please explicitly say so. If you do not say so, there is no way one can be sure about the way you count. We will not be able to give credits whenever there is an ambiguity. So, please try to be as clear as possible whenever you answer questions.

(Refer Slide Time: 03:37)



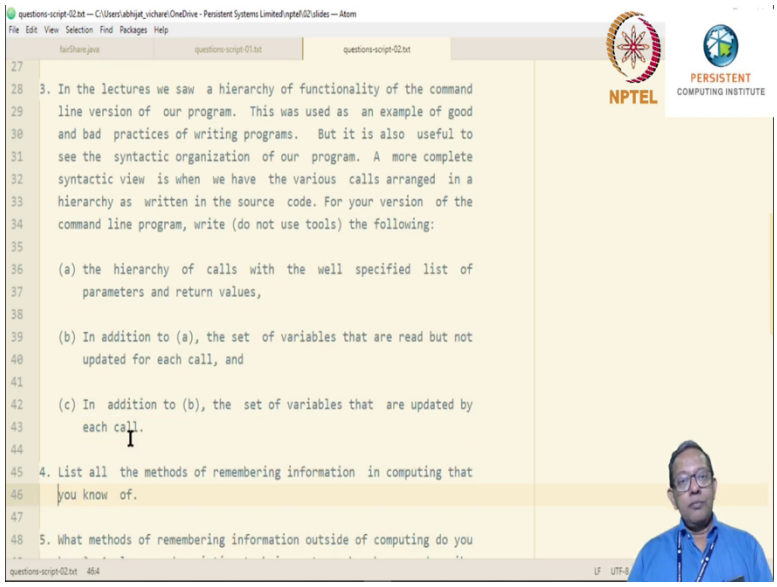
2. Let us look at the second question: **information to control program flow versus information to compute new values**. This question is trying to distinguish between information that is used to control the flow of programs versus that is used to compute new values. For example, in `processCommandLine` method, the information on the

command line arguments is used to decide should the program be doing registration, expenses, or reporting. Hence, the information used was not about "how to do" instead it is about "what to do". So, we will call such information as the "what kind of a variable" or **"what variables".**

In contrast there are other piece of information which will actually generate new values. For example, `perHeadShare` in that `computeFairShare` method: it generates new and useful information from whatever is available so far. Although they are variables which just remember information, we will call them as "**how variables**" in the sense that they capture the sense of "how to generate new information".

This question asks you to classify variables in your programs as "what variables" and "how variables".

(Refer Slide Time: 05:12)

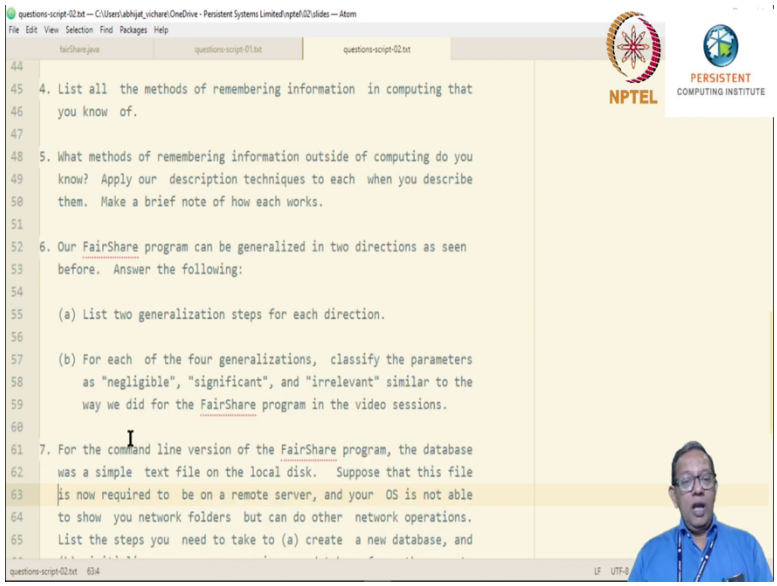


3. In the lectures we saw a hierarchy of functionality of the command line version of our program, this was used as an example of good and bad practices of writing programs, but it is also useful to see the syntactic organization, or **call graphs** as it is also called, of our programs. A more complete syntactic view, or call graphs again, is when we have various calls arranged in a hierarchy as written in the source code.

For your version of the command line program write the following (when we say right please do it by your hands, do not use tools to generate that information that is asked; it is useful to in the initial phases to actually work out things by hand):

A. The hierarchy of calls with well specified list of parameters and return values
B. In addition to A, the set of variables that are red but not updated for each call.
C. And finally, in addition to B the set of variables that are also updated by each call.

(Refer Slide Time: 06:33)



4. **List all the methods of remembering information in computing that you know of.** It is possible that you might miss some, so try to visit your library, read books, recall from your memory whatever you have learned so far, and maybe even use Google or the internet to find out. But try to be as comprehensive and as complete as you can.

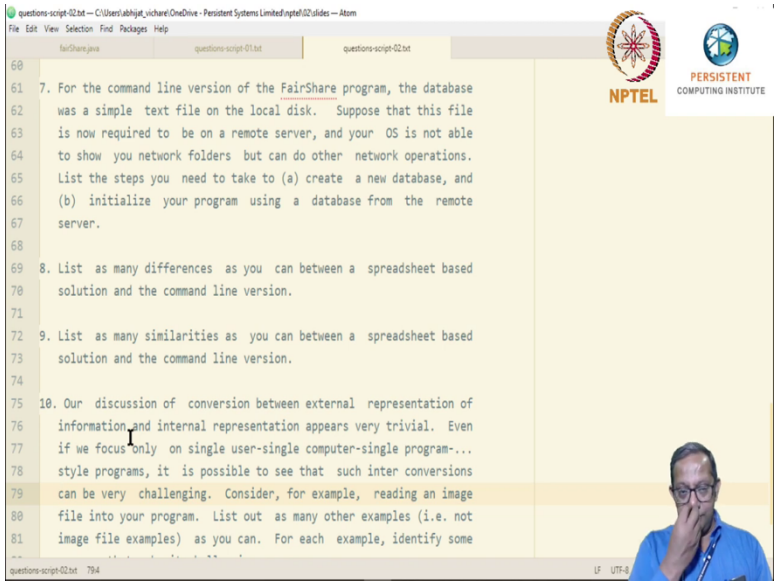
5. **What methods of remembering information outside of computing do you know?** Apply the description techniques to each when you describe them, make a brief note of how each works. The intent of this question is to help you see that the ideas that you have been using in computing are also actually around you.

6. Our fair share program can be generalized in two directions as seen before. Answer the following:

A. List two generalization steps for each direction. Note that this gives us a total of 4 different generalization steps.
B. For each of the 4 generalizations classify the parameters as “negligible”, “significant”, and “irrelevant” similar to the way we did for the fairShare program in the video session.

Questions like the #6, and in fact many of them that we have seen so far, are just trying to get you to look at your program and various parts of it in different, different ways. These ways could be useful for us as we progress through this course. We therefore urge you to try your best to answer these questions.

(Refer Slide Time: 08:58)



7. For the command line version of this program the database was a simple text file on the local disk. Suppose that this file is now required to be on a remote server, so it remains as a simple text file it is a simple text file except that it is not on the local disk but on some remote server. And your OS is not able to show network folders. In other words, although the file is on the remote server on your operating system you do not have a program which shows you as if it is a local file.

   However, your system is able to do networking; it can do network operations although it is not showing you the files in one single folder. **In such a case list the steps you need to take in order to**

A. Create a new database

B. Initialize your program using a database from the remote server.

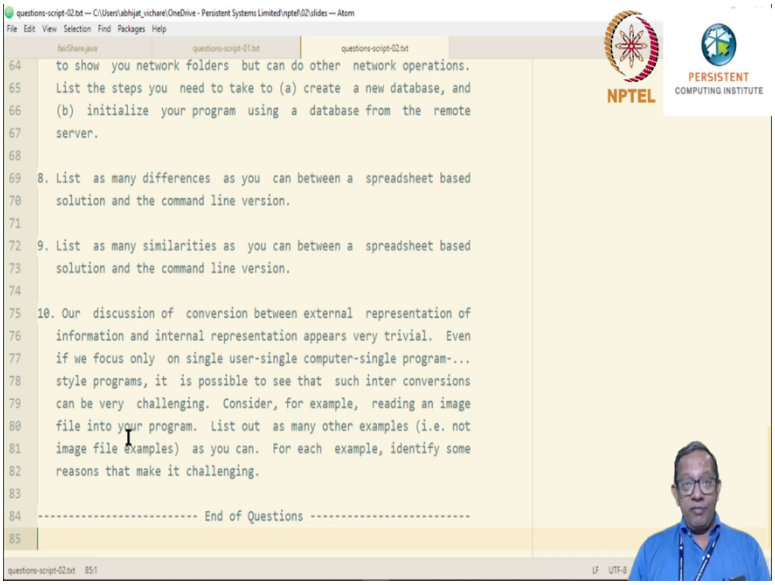
It basically asks you to list out the steps you would need to implement create database and initialize from database in the source code.

8. **List as many differences as you can between a spreadsheet-based solution and a command line version.**

9. **List as many similarities as you can between a spreadsheet-based solution and a command line version.**

Both these questions, in a sense, actually just compare and contrast the two solutions with each other. There are some advantages of one solution over the other, there are some disadvantages of one solution or the other. Every approach really has both these aspects, the pros and the cons, and in practical life we need to balance them.

(Refer Slide Time: 11:18)



10. Our discussion of conversions between external representation of the information and internal representation appears very trivial. As in in our command-line discussion, the kind of conversions that we have done for input and for output appear to be very trivial. Even if we focus only on a single user, single computer, single program… etc., style

programs, it is still possible to see that such inter conversions can be challenging. Our example may make it appear trivial but that is perhaps because our example was not a very good one.

But it is not always the case, for example, consider that you want to read an image file into your program. When you want to read an image file you need to know the format of the image file and depending on the way the format is organized reading that file can become tedious.

List out as many other similar examples, not the image file example obviously, as you can where conversion is tedious, also for each example identify the reasons why it might make it challenging.