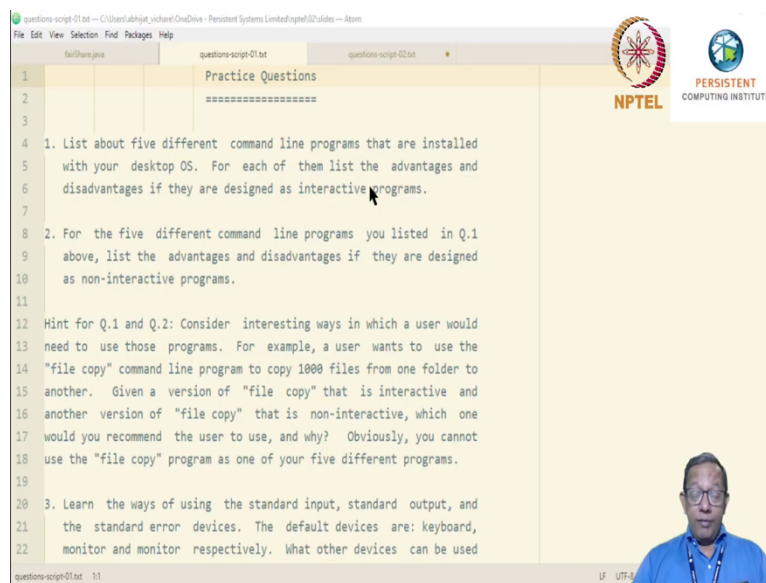


Introduction to Modern Application Development Persistent Computer Institute

Lecture 7 Command Line _ Practice Questions - Part 1

(Refer Slide Time: 00:11)



The screenshot shows a presentation slide titled "Practice Questions" with two numbered questions. The first question asks for five different command-line programs installed on a desktop OS, listing their advantages and disadvantages if they are interactive. The second question asks for the same but for non-interactive programs. A hint is provided for both questions, suggesting to consider interesting ways a user would use the programs, using "file copy" as an example. A small video inset of a speaker is visible in the bottom right corner of the slide.

```
1 Practice Questions
2 =====
3
4 1. List about five different command line programs that are installed
5 with your desktop OS. For each of them list the advantages and
6 disadvantages if they are designed as interactive programs.
7
8 2. For the five different command line programs you listed in Q.1
9 above, list the advantages and disadvantages if they are designed
10 as non-interactive programs.
11
12 Hint for Q.1 and Q.2: Consider interesting ways in which a user would
13 need to use those programs. For example, a user wants to use the
14 "file copy" command line program to copy 1000 files from one folder to
15 another. Given a version of "file copy" that is interactive and
16 another version of "file copy" that is non-interactive, which one
17 would you recommend the user to use, and why? Obviously, you cannot
18 use the "file copy" program as one of your five different programs.
19
20 3. Learn the ways of using the standard input, standard output, and
21 the standard error devices. The default devices are: keyboard,
22 monitor and monitor respectively. What other devices can be used
```

Hello everyone, welcome to the question session of the second week. This is the first question session in which we will look at a few questions based on the first lecture of this week. Here are few questions:

1. **List about 5 different commands (command-line program) that are installed on your desktop operating system.** We are a bit categoric here, please you look at your desktop operating systems in contrast do not look at your mobile operating system, for instance.

For each of the command line programs that you see on your operating system list out the advantages and disadvantages if they are designed as interactive programs. This question is about interactivity.

2. The second question is again on the same curve as the first question except that it considers the issue of non interactivity. **If those same commands that you have used and have listed in question 1 are non-interactive, or suppose they behave non-**

interactively, then what would be the advantages or disadvantages of this. These questions are meant to get you thinking about aspects of program design.

Hint: Consider the interesting ways in which a user would use the program. Take file copy program as an example (note that the file copy program must not be a part of your list of 5 programs). Your operating system be it windows GNU/Linux, Mac OS... etc., will typically have a command line version of a file copy program.

Consider the scenario when a user has to, say, copy a huge number of files, say 1000, would an interactive version of file copy program be an advantage or a disadvantage? In what case will a non-interactive version of this problem would be a challenge.

3. The third question gets you learning about using the standard input, output and error devices of your system. The default devices are keyboard monitor and monitor respectively for input standard output and standard error. **What other devices can be used instead of these default ones? Try to find that out for your operating system.**

4. The sample command line program used in the video lecture is a correctly working program; it is correct in the sense that it performs the necessary calculations correctly. It works in the sense when the user gives the correct user commands, it performs a correct operation corresponding to that command. However, it is not yet a software product. This question asks you to **list at least three distinct improvements in the command line version of the program that will make it a software product.** Note that the program must remain a command line program.

So, an improvement which says convert this program into a graphical one is not something what we are looking for in your answer. Your program will remain a command line program but nevertheless becomes a software product, which it is not right now.

5. **Develop a detailed proof of correctness of our method used in the fair share program. Try to ensure that each step has one and only one clear change from the previous step.** For each step be sure to note the reason behind the change that you did. From our experience we know that you may have to repeat this exercise a few times to get it right. The idea of this exercise is to have a step by step reasoning about the correctness of our program. Usually when we think about things like this we very quickly

jump to conclusions. This exercise is meant to take us step by step with every step justified by some reason. It might be slow and you may realize that you have forgotten some steps in the version that you wrote, so you may have to rewrite it again.

6. The command line program we have used in the video lectures illustrates an important principle of design of user interaction. The information must be displayed consistently with any required conversions between internal and displayed information. Recall that when we displayed a report when a roommate actually owed some money then we said that that particular roommate the amount owed is minus some number. We also remarked that negative values are our internal understanding that they represent the money to be paid. But when you display that money to be paid as a negative number, and moreover saying that this is the amount owed, a typical user would be very confused. It is very important to be consistent in the way the information is displayed. This is the principle that we did not do well in our program. **This question asks you to list out few other such principles that you may recall from your studies.**

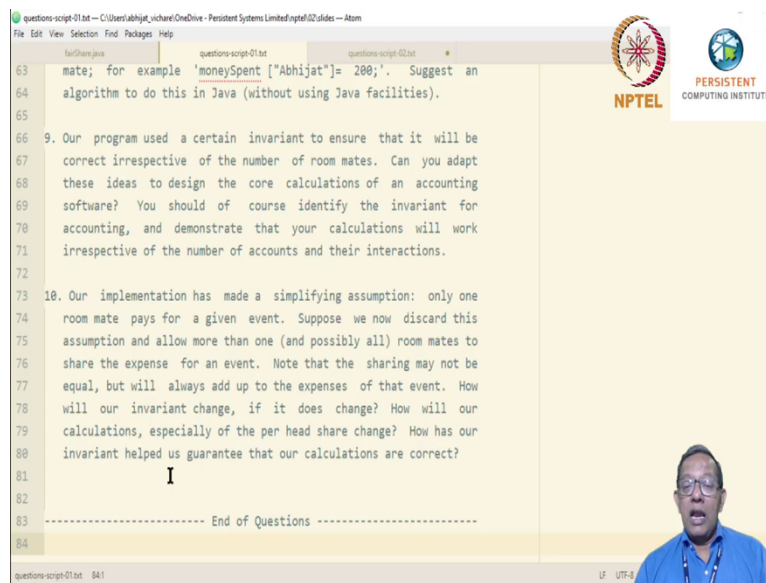
7. The fair share program can be written in procedural style as well as object-oriented style. We suggest that you write the program in both the styles, that way you will understand the differences and the advantages between the two styles. Therefore, **this question asks you to write in the other style than the one that you have written.** If we have written in an object-oriented style then this question asks you to rewrite the program in a procedural style. If we have written it in a procedural style then this program asks you this question asks you to rewrite the program in the other one.

8. The arrays in our program are indexed using numbers, for example the `moneySpent [getIndex("Abhijat")] = 200`, where `getIndex("Abhijat")` returns the against which the money spent by Abhijat is stored. It would be nice if we could index them directly using the names of the roommates. So, instead of saying money spent by get indexed directly it would be great if we could write it as `moneySpent ["Abhijat"] = 200`. **This exercise just asks you to suggest an algorithm in Java how this could be done.** Of course, if Java has any facilities which support this already then you may not use them; you have to write the algorithm independent of Java approach.

9. Our program used a certain invariant to ensure that it will be correct irrespective of the number of roommates, **can you adapt these ideas to design the core calculations of an accounting software.** You should of course identify the invariant for accounting and demonstrate that your calculations will work irrespective of the number of accounts and their interactions.

In this question we try to take our understanding of the invariant and correctness ideas for the fair share application and apply it to an accounting problem. There are many ways this can be discussed. We hope that you actually use the forums to discuss questions like these in detail. **This question requires you to think about how to use the ideas from one problem that you have solved and applying them to another problem.**

(Refer Slide Time: 11:45)



The screenshot shows a presentation slide with a code editor on the left and a list of questions on the right. The code editor contains the following text:

```
63 mate; for example 'moneySpent ["Abhijat"] = 200;'. Suggest an
64 algorithm to do this in Java (without using Java facilities).
65
66 9. Our program used a certain invariant to ensure that it will be
67 correct irrespective of the number of room mates. Can you adapt
68 these ideas to design the core calculations of an accounting
69 software? You should of course identify the invariant for
70 accounting, and demonstrate that your calculations will work
71 irrespective of the number of accounts and their interactions.
72
73
74 10. Our implementation has made a simplifying assumption: only one
75 room mate pays for a given event. Suppose we now discard this
76 assumption and allow more than one (and possibly all) room mates to
77 share the expense for an event. Note that the sharing may not be
78 equal, but will always add up to the expenses of that event. How
79 will our invariant change, if it does change? How will our
80 calculations, especially of the per head share change? How has our
81 invariant helped us guarantee that our calculations are correct?
82
83 ----- End of Questions -----
84
```

The slide also features the NPTEL logo and the Persistent Computing Institute logo. A small video inset in the bottom right corner shows a man speaking.

10. Our implementation has made a simplifying assumption only one roommate pays for a given event. Suppose, we now discard this assumption and allow more than one and possibly all roommates to share the expense for an event. Note that the sharing may not be equal but will always add up to the expense of that event. So, if more than one roommate share then they do not have to share equally, one roommate may spend a little more than and the other roommate.

In this new scenario, how will our invariant change, if it does change? How will our calculations, especially of the per head share, change? How has our invariant helped us to guarantee the correctness of our calculations?

Apart from programming exercise we suggest that you also try to answer these questions for yourself. Of course, you are welcome to discuss them on forums, at various points we will give hints of the solutions, thank you.