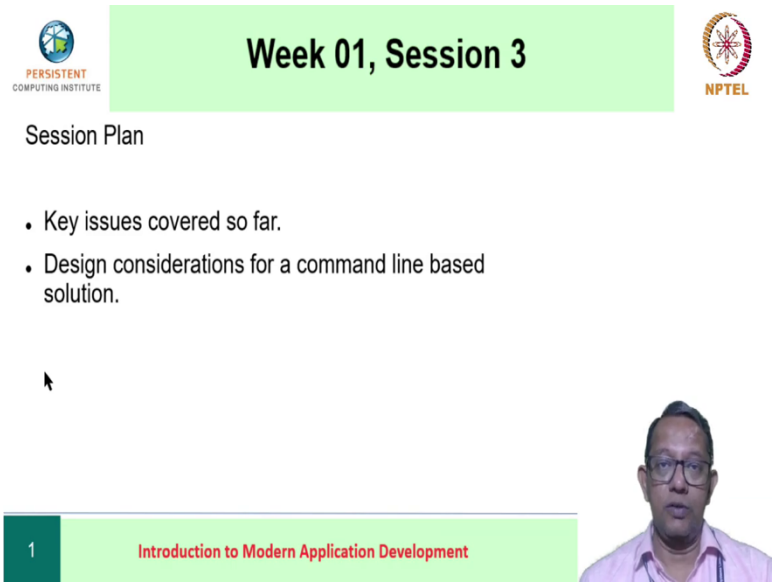
Introduction to Modern Application Development
Persistent Computer Institute

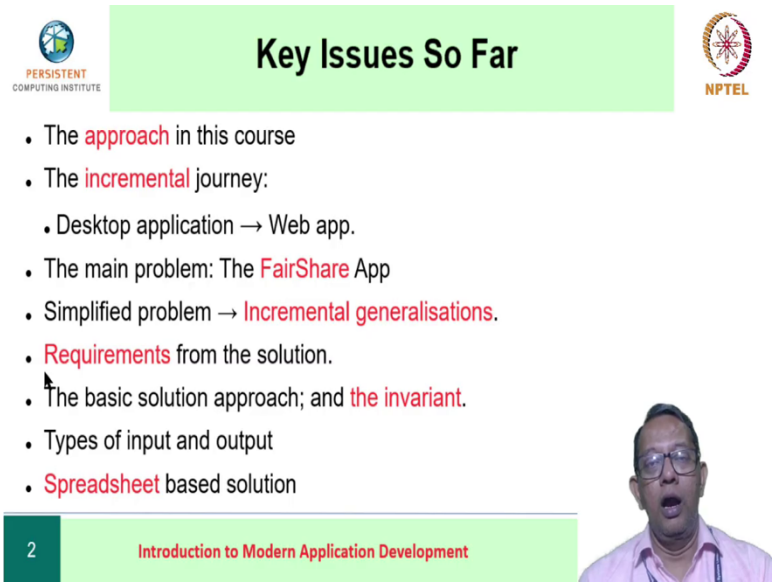
Lecture 9
Command Line _ Practice Questions - Part 2

(Refer Slide Time: 00:16)



Hello everyone, welcome to the third session of the first week. Let us begin by reviewing what we have learned so far, the key issues and then go on to look at the design considerations that we will need when we want to implement our solution as a command-line application.

(Refer Slide Time: 00:43)



The key aspect of this course is the approach that is taken it is an incremental approach by that we mean that we would start from something that we know how to do a desktop application. And gradually stepwise go on enhancing it and extending it so that eventually it becomes a web-based application, which is the way modern applications are typically done. The main problem that we are going to use to help us do this is the fair share problem. The story behind the problem is about a set of roommates who undergo a lot of activities like going for dinner or maybe movies or buying some things, and for each such activity some roommates pay.

Eventually there comes a time when they want to reconcile all their payments and debts. We have considered a simplified version of this problem, Professor Sane has already illustrated that where we have a few friends like 2 or 3 undergoing a set of activities with the constraint that for any activity one and only one roommate pays. This simple solution was extended a little for a few roommates, and we saw how it can be implemented on a spreadsheet.

The spreadsheet version helps us to clarify our solution ideas, in particular it helps us to see how our solution is correct. The way we do that is to try to observe behaviour of the within the problem that does not change: **the invariant behaviour**. And once this invariant behaviour is identified because the behaviour does not change therefore whatever are the results based on this behaviour will not change whatever be the number of roommates.

This allows us to reason about correctness of our solution: something that is very important because we are making a machine do things for us. There are other advantages of this approach also, for example, it helps us to figure out errors. If for example, the invariant that we calculate is not zero, we can immediately see that there is some problem in our program and something has gone wrong somewhere.

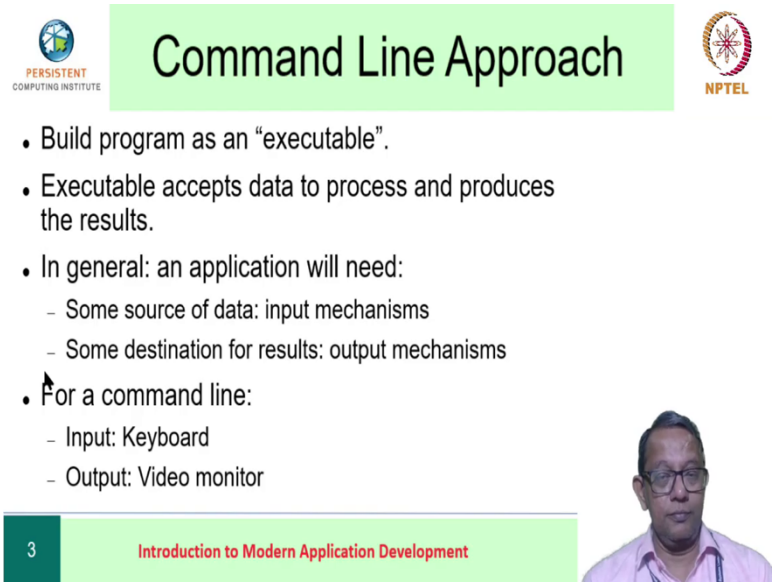
For example, you might have had an extra piece of input or the input amounts exceed the actual payment for that event, or some other issue, but something is not right! It is very much easier to check the correctness about our program when we have some invariant in our problem.

Professor Sane has also spent some time showing us the what is required out of the solution. He has used command line to illustrate the requirements. For example, we would need to have at least 3 modes of operation the **registering** mode, **the expenses** mode, and a **report generation** mode.

We went on further and asked ourselves suppose we want to implement this on a computer, that is we want to make this a computer-based solution, what more do we need to think? Well, we need to worry about the various types of input devices or more appropriately sources from which input can come. Similarly, we have to consider the various output devices to which the output can be shown or provided. Instead of taking all the input devices and all the output devices right at the start, it is perhaps much better to start with a simplified version of the computer-based solution where we just pick up one put device and focus on one output device.

The spreadsheet-based solution went one step further, because spreadsheet itself is an application as an application it takes care of the input and output by itself therefore as users of spreadsheet we just focus on the basic algorithm of our solution. In fact, it is this focus that allows us to directly observe the invariant behaviour in the problem; that is the value of the spreadsheet solution. Not only can we observe what is not changing, the invariant, but we can also use it to check our solution. That is what we saw in the spreadsheet.

(Refer Slide Time: 07:19)



Let us now go ahead by try to by trying to think about a command line version of our solution. The spreadsheet solution gave us a foundation, particularly as it allowed us to focus on the algorithmic component of the solution. But let us now make our solution into a true program,

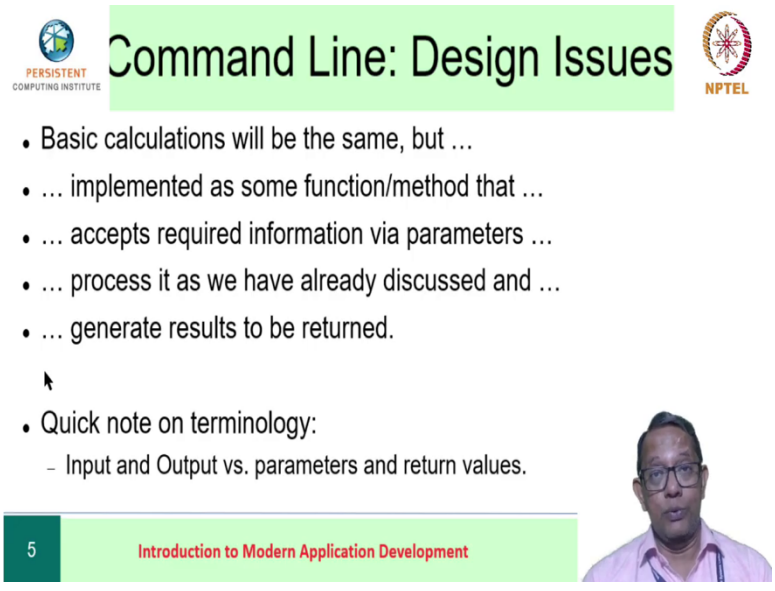
what could a command line application be? To quickly respond to that, it is just building your program as an executable.

But what really is an executable? A conventional answer to that it is the binary version of your program. Why binary? Because your machine, the hardware, is actually able to work with binary objects only, that is why every high level every program in a high-level language has to be translated to a low-level language, the binary, and it has to be executed. And executable will accept data for input process it, and produce the results. In general, the executable will require some source of input via some mechanisms for input which we will have to be programmed into the executable.

Similarly, it will require some output. Whatever output it generates, that generated output has to be put on to output devices. For a command-line application, we will fix our input devices to be the keyboard and our output devices as the monitor. This is very much along the direction that we have proposed in our course where we incrementally go on changing our application solution to our eventually web-based one.

Right now, we are just starting to build our desktop version of the application and we have chosen to focus on keyboard as the input devices instead of the many input devices possible. Similarly, we have chosen to focus on the video monitor as the output devices instead of looking at all the possible way variety of output devices. Typically, **the keyboard-monitor pair is called as a terminal.**

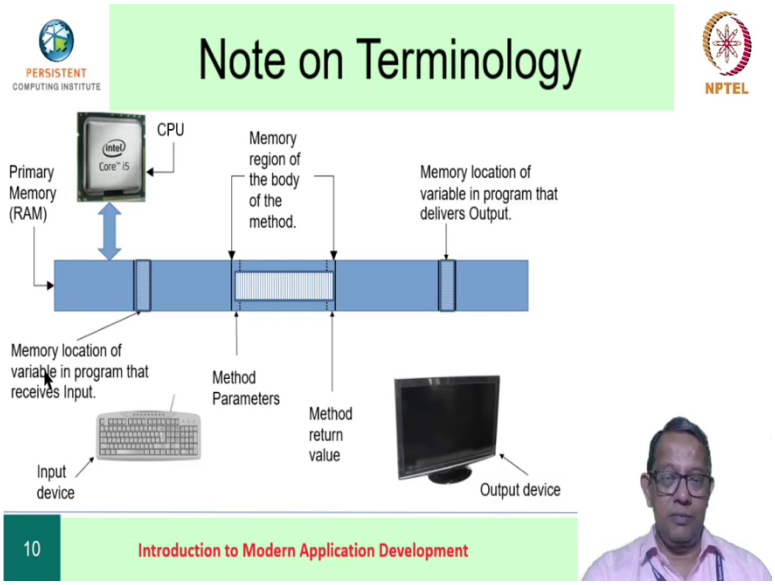
(Refer Slide Time: 10:34)



Let us come to the design of the basic calculation component of our command line version of our program. The basic calculations remain the same, but the it will be implemented as some function or a method. Functions and methods are essentially the same it's just that different languages use the same term in a different way. Therefore, whatever is your favourite language feel free to use that while thinking about this, however as far as this course is concerned, we will be using Java.

Therefore, our basic calculations will be implemented as some method of some class, this method will accept the required information for calculations via parameters. The acquired information will then be processed according to the algorithm that was discussed already. The generated output will be returned back.
This is a good point to stop and digress for a quick note on terminology. What we will be talking about is input and output versus parameters and return values.

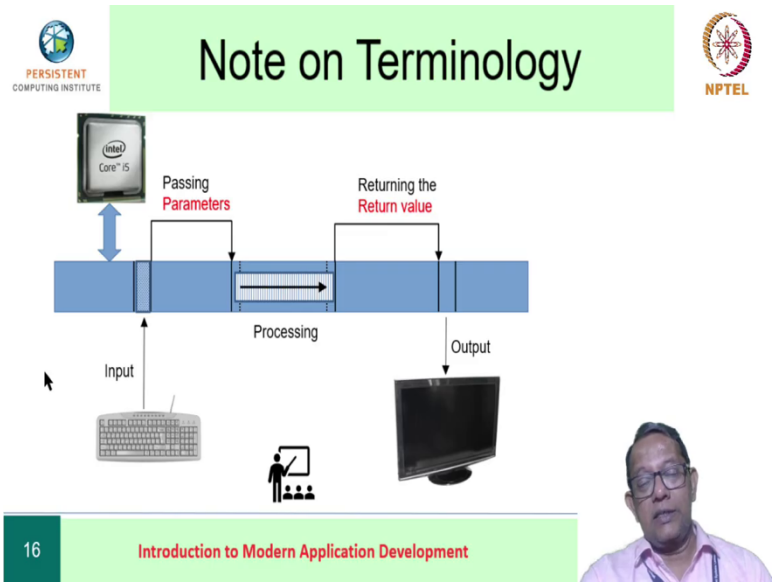
(Refer Slide Time: 12:20)



Here is a sketch of a typical computer system: you see a CPU and a primary memory in blue. On the primary memory is some memory area that is going to collect the input that has been transmitted via the input device. In this case transmission of information means typing on the keyboard and the keyboard connected to the machine; whatever is typed on the keyboard will be collected into some memory location on your RAM.

On the primary memory we also have some area that has the method that is going to process the input. We have also shown some area reserved for parameters of the method and some area reserved for the return value of the method. Finally, we also see some area of the primary memory reserved for collecting the output and this memory is the connection with the output device, in other words whatever is in that memory is eventually transmitted to the output device for display.

(Refer Slide Time: 14:05)



Let us come back to the terminology problem. This is the entire picture of our computer system. Let us continue with the terminology. Whenever we use our I/O devices to transfer or collect information from the computer we will refer to those parts as input and output. So, in particular when we transfer information into the computer using the keyboard that phase will be called as input. Note that whatever is input is collected into some memory location which is some variable of your program. We then passed the value in the memory location as a parameter via a method call.

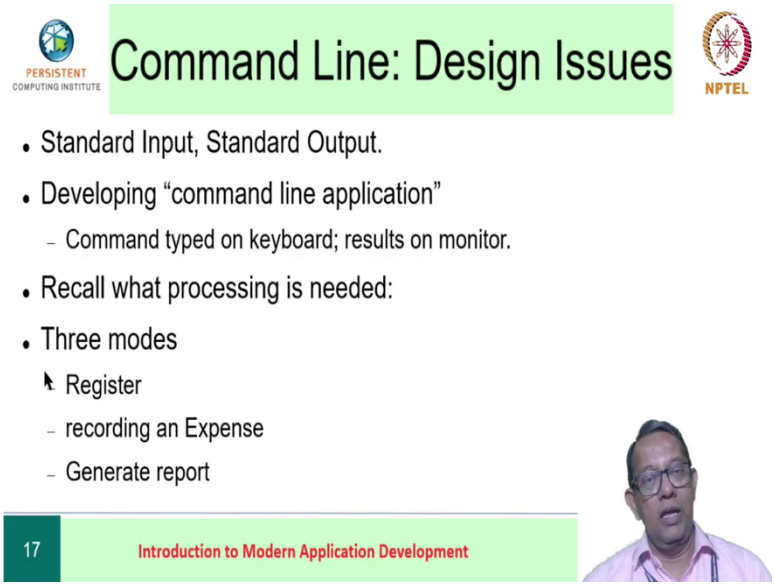
Note that parameters values are passed. You may recall various ways of passing parameter values, for example pass by value, pass by reference, and many others that you may recall. In whatever way the parameters are passed, they will then be processed according to whatever is defined in the method, and after processing the values for output are generated by the method. Those values are returned back to the calling system, typically they are collected into some memory location within your program.

From this memory location the values are now output onto the output device. The phase in which we are passing information from input devices to your computer system is input phase. Whenever we collect information from the computer and display it on an output device, we will call that phase as output.

A method accepts parameters which are passed to it and returns results. To be clear, we will not say that methods parameters are input to methods and methods generate output. Now methods generate return values which are then transferred to the output devices. Input and output typically refer to interaction of the computing system with the outside world via I/O devices. In contrast, passing values or returning them is always within the computer system from one memory area to another memory area, and that is what passing parameters and returning values amounts to.
This is a good time to try review what has been learned so far.

We have started looking at the design considerations of a command-line application and during that process we took up a slight digression to have a clear review about the terminology that we will use.

(Refer Slide Time: 17:56)



Given the variety of input and output devices operating systems typically create an abstraction called as a standard input, which by default is the keyboard, but it could be any source of data and we can request the operating system to change our standard input device. Similarly, the operating systems typically most operating systems typically allow create an abstraction of a

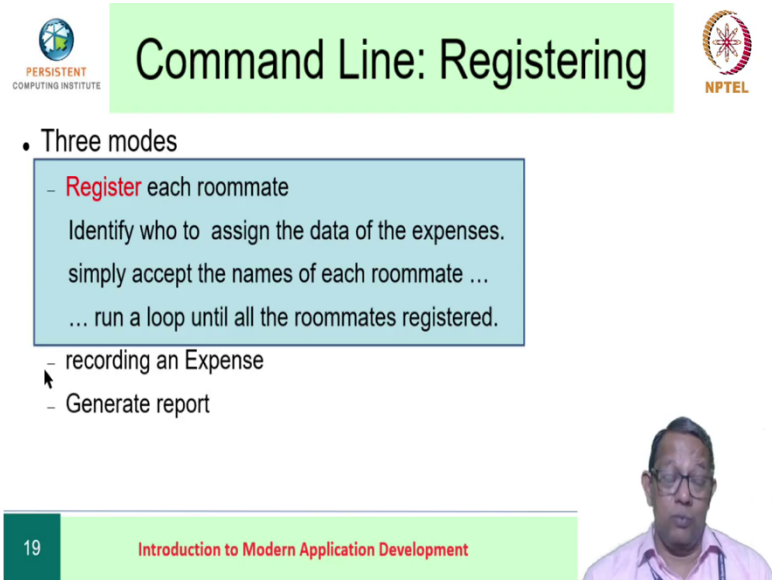
standard output device which by default is the monitor, but it could be any other out suitable output device. Our default input is from keyboard and default outputs race to the monitor, in other words our default IO device is a terminal.

Developing a command-line application means we type the application as a command on the keyboard and whatever are the results of that application in this case the fair share app, the results will be visible on the monitor. Let us now recall what processing is needed, in particular, Professor Sane as already illustrated that there are 3 different modes that our application really needs to worry about. These are the minimum number of modes and you can, depending on what you feel is right, extend the number of modes make them sharper.

But the minimum number of modes that we need are:

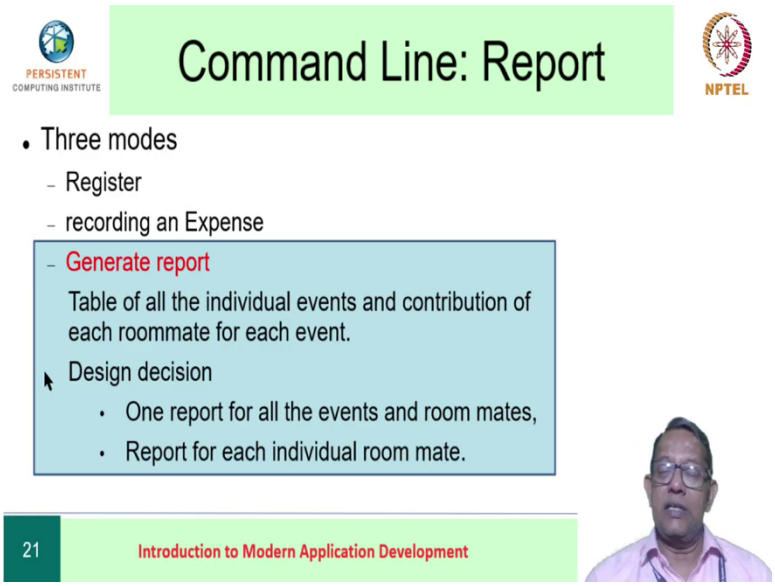
- a mode for the registration of your roommates
- a mode in which a given expense for an event will be recorded
- a mode in which the report will be generated.

(Refer Slide Time: 20:03)



Let us look at each individual modes a little more in detail. What does registering each roommate involve and why do it? Registration act is required so that we can identify who to assign the data of the expenses to. How do we do it? Well, we simply accept the names of each roommates, and how would that be done? Run a loop until all the roommates are registered. This is indeed a very simplistic approach, but nevertheless it works.

(Refer Slide Time: 20:53)



What does recording an expense mean? How would did it be done? Recording an expense would mean collecting all the data that is required for that particular expense. At least two pieces of information are really needed: accepting the name of the roommate who has made the payment, and the actual amount that has been paid. Of course, when recording an expense, we need to remember that so-and-so roommate actually made the payment.

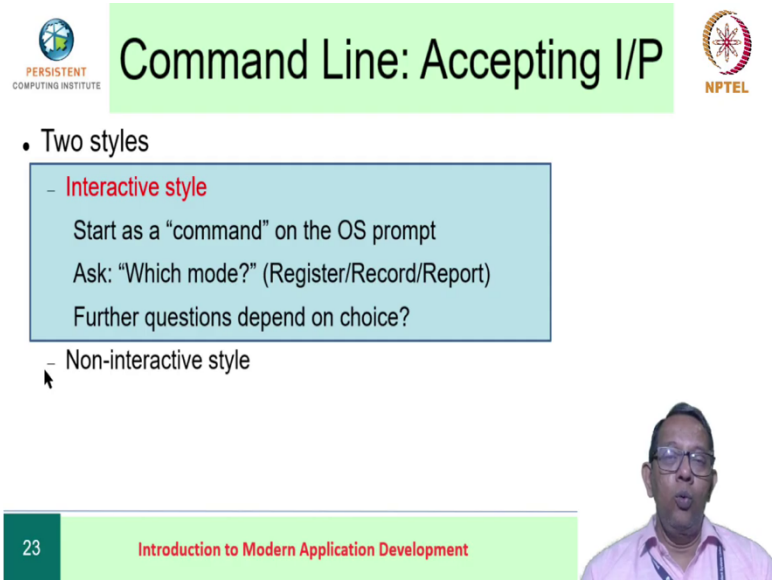
In the third phase of generating the report, the report actually is just a table of all the expenses that have been done so far by the group. Since we have chosen to generate the report at the end of the month, we are going to say that this table contains all the events for a given month. However, there is still one more design decision to be done. Do we generate report at the end of the month for all the members of the group of roommates, or do we generate report for individuals?

If we generate report for all of them then the single report is just copied out multiple times for each roommate and he or she gets to see the entire expenses who paid who for what event and how much. In contrast suppose the report is generated for an individual, then the individual only sees his or her final contribution required. Either they may get paid because they have been paying so far or they may have to pay because they have not paid for many events.

Let us arbitrarily decide to report on an individual basis. It is a simple exercise to extend this such that your application can be made to either generate a complete report or generate an individual report.

This is again a good time to pause and review what has been talked about.

(Refer Slide Time: 23:52)



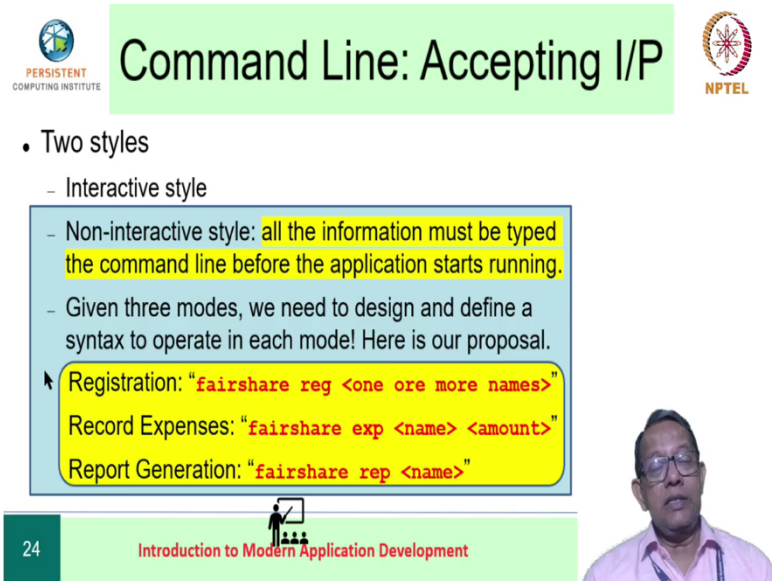
For a command-line application we have chosen to use the keyboard. However, keyboard is just the source of input there are two styles in which inputs can be taken up from input devices. The first is interactive style; in an interactive style you start the command as a prompt, just the command nothing more. Your application which is now a command of your OS starts running and now it starts asking question, for example, which mode do you want to operate in? The possible modes are of course register, record or report.

And depending on whichever is the particular choice the subsequent questions will be asked:

- If the mode chosen is *'registering'* then the next question could be how many roommates are there. After updating the number of roommates, the subsequent questions will be name of each roommate.
- If the mode chosen is *'record'*, then the command line program will ask what event next, who paid and finally how much was paid.
- If the mode chosen is 'report' then the next question would be whose report is required, at which point the user types his or her name and gets the corresponding report.

We see that the questions that will be asked by the command in application will depend on the choice of the question which mode. All of this is happening after the command has started running.

(Refer Slide Time: 26:21)



The other style is a non-interactive style of accepting inputs. ***In a non-interacting style, all the information that is required for input must be first typed on the command line before the application starts,*** this is the key difference between an interactive application style and a non interactive style. In a non-interactive style, all information that is required must be given before the program starts executing. In an interactive style you typically collect the information after the program starts running.

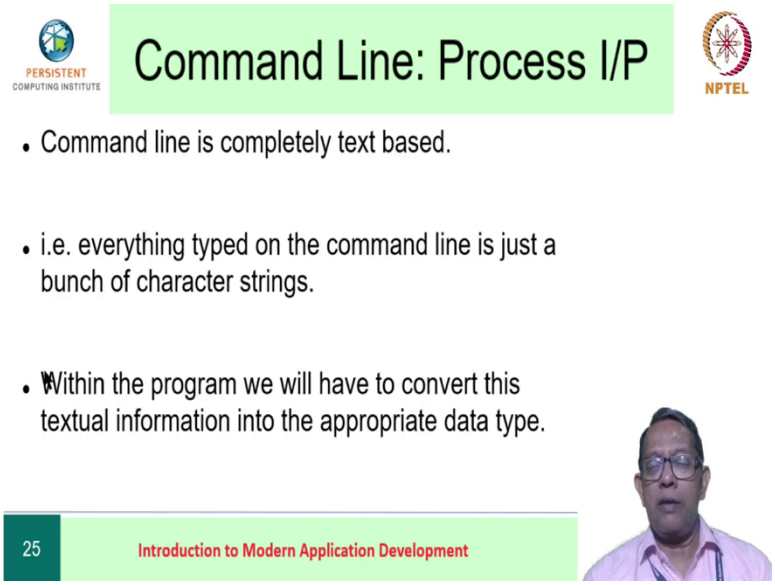
Given that in a non-interactive style we need to supply all the information before the program runs, it means that we must now think about designing the syntax for the entire command line. For an interactive system only the name of the command was sufficient, nothing more was required on the rest of the command line. For a non interactive style because we need to supply information before the program runs, we must now put all the required information on the command line itself. This means we need to design and define a syntax for the command line.

Moreover, the syntax should be defined for each mode. Here is our proposal for registration, let us add a keyword **"reg",** which is a short for register, and the command line application would now look like: `fairShare reg <one or more names>`

In order to record the expenses, let us create a keyword "exp" for a short for expenses, and then the command would look like: `fairShare exp <name> <amount>`

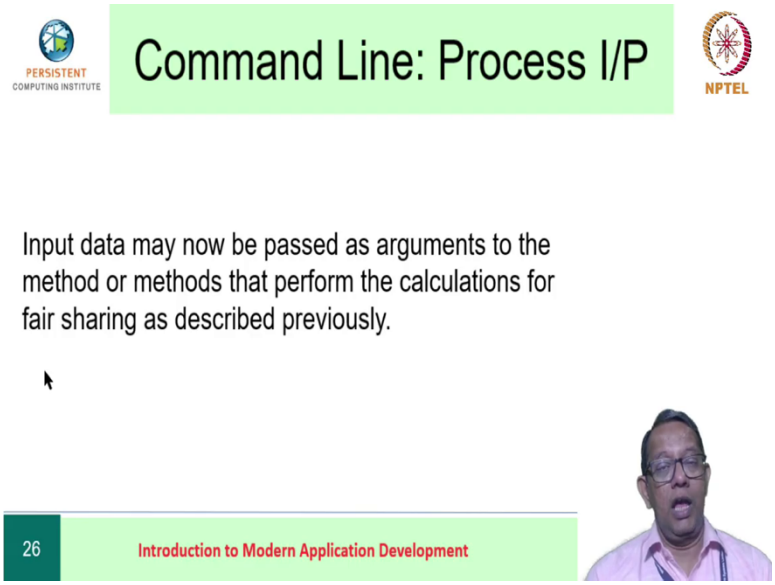
For report generation let us use the word “rep” which is a short for report, the command would then look like: `fairShare rep <name of roommate>`

(Refer Slide Time: 29:32)



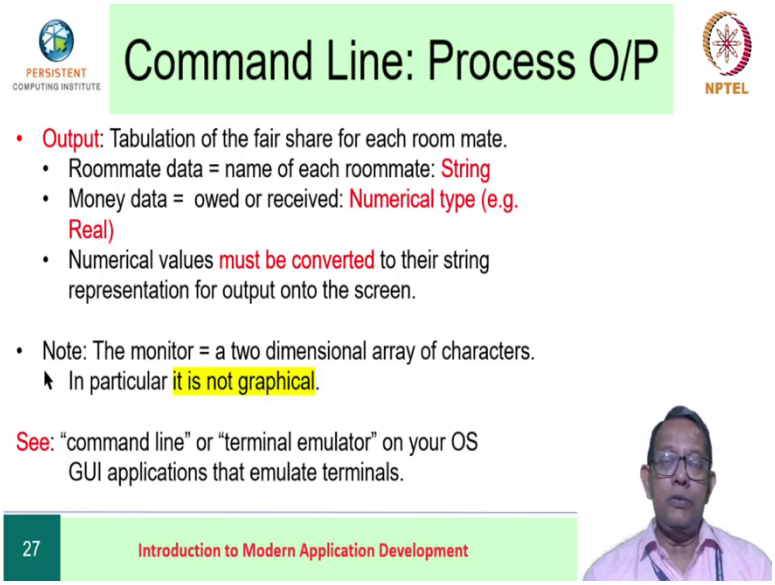
Observe that the command line is completely text-based, that is everything is typed on the command line and whatever is typed is just a bunch of characters, a set of strings, really nothing more. But that means that within the program we must convert these strings that appear on the command line from the textual form to whatever is the appropriate data type that we need.

(Refer Slide Time: 30:06)



For example, whatever amount you type on the expenses mode of operation that is just a set of characters which need to be converted to the numerical value before we actually use it. However, after one such conversion is done then the input data can be passed as arguments to any number of methods that perform the required calculations, but you cannot just pass the strings.

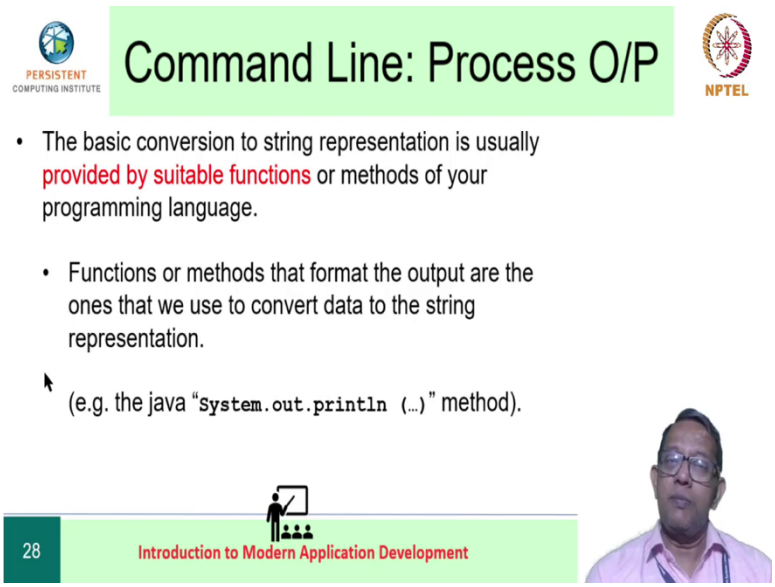
(Refer Slide Time: 30:55)



How would output processing now look like? Recall that the output is just the tabulation of the fair share for each roommate. The roommate data that is the name of each roommate is just a string, the money data whatever it is owed or received that is a numerical type, typically a real number which in languages like C, C++ or Java would call it as a floating point. Note that numerical values must be converted to their string form before the output on the screen.

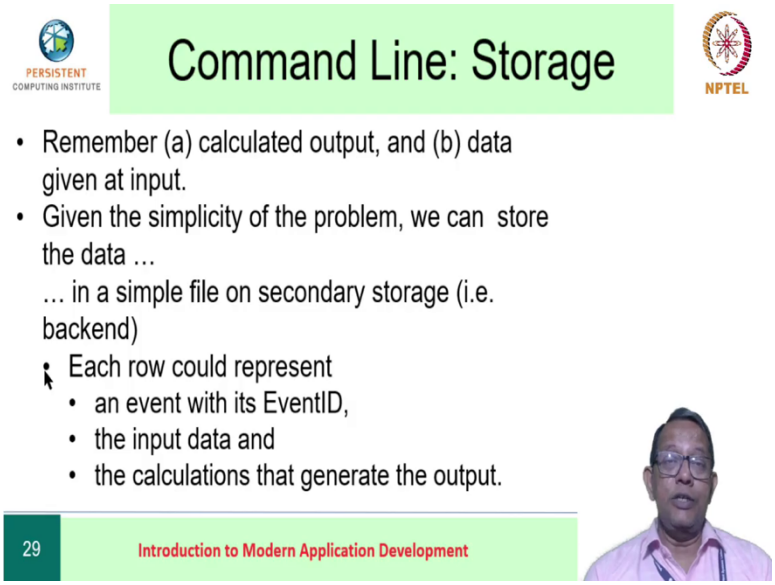
In our case when we are thinking of a terminal, standard input and standard output, we should be thinking of the monitor as a 2-dimensional area of characters, in particular it is not graphical. To really see the difference, see command line or terminal emulator on your OS, these are GUI applications that emulate terminals.

(Refer Slide Time: 32:23)



The basic conversion to string representation is usually provided by suitable functions or methods of your programming language. Functions, for example, that format the output are the ones that we used to convert the data to the string representation. For example, in Java you might be using `system.out.println()` or any such suitable method, but these methods that typically convert to the string representations are often provided by the language itself. *Again, this is a good point to stop and review what has been done.*

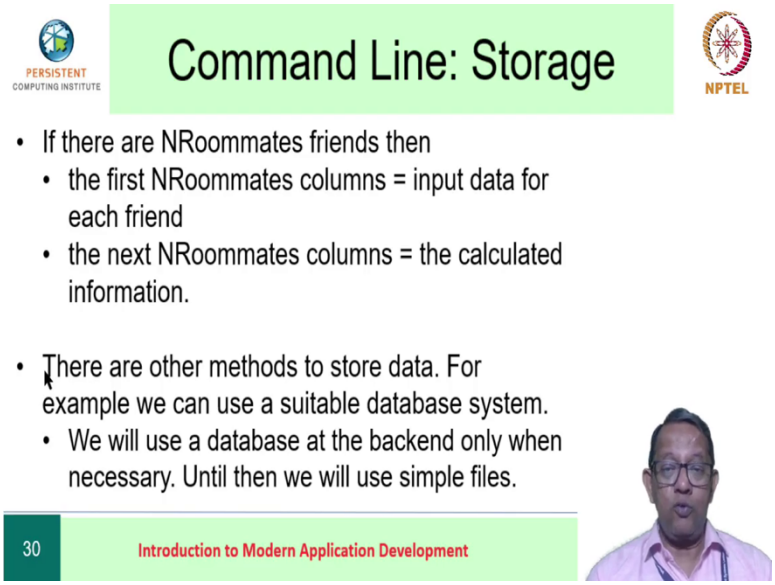
(Refer Slide Time: 33:16)



Let us now consider the issues of storage; this was not something that we had really thought about so far. But the idea is that we need to remember the calculated results and of course also the input data. Given that we are really using a simplified problem, we can store this information in a simple file on secondary storage, that is the backend. And we could store the file in a manner similar to what we saw in the spreadsheet.

Each row would be one line in the file, each different row will be a different line in the file, and the values that are stored are shown on a line the spreadsheet when we put them into the file, and one of the simplest ways of storing them is by separating the values by commas. This format is known as comma separated values or CSV. In a CSV, each row could represent an event with its event ID, the input data that is the expenses by whom for that event, and finally, the calculations of the per share data that are generated as the output

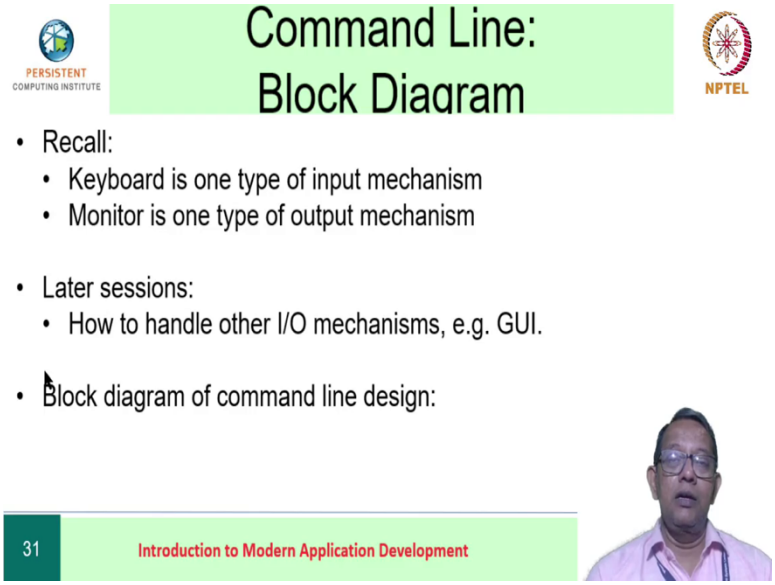
(Refer Slide Time: 35:06)



If there are `NRoommates` friends then the first `NRoommates` columns are the input data for each friend, and the next `NRoommates` columns are the calculated information; this is very similar to the spreadsheet. There are of course other methods to store data for example we could use a suitable database system and we may have to eventually use one.

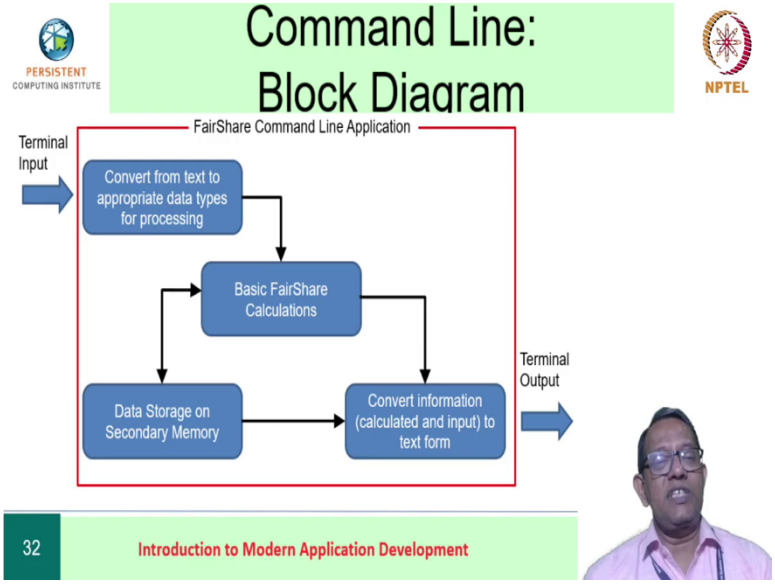
But we will use a database at the back end only when necessary until then let us use simple files.

(Refer Slide Time: 35:45)



Let us now bring all of these things together and try to pictorially depict out the functionality of a command-line program. Just a quick recap, the keyboard is the in one of the input mechanisms and we have chosen that as our input mechanism, and monitor is one kind of output mechanism and we have chosen that. In later sessions we will look at how to handle other input and output mechanisms like a graphical user interface.

(Refer Slide Time: 36:32)



But in the meanwhile, here is a block diagram of the command line design, from the left we see a terminal input, the red outline is the boundary of the command-line application, and on the right, we see a terminal output. Within the red box, the outlined box, we see that from the input, whatever is the data that comes, we need to convert it to appropriate data type for processing. It is then passed to the actual fair share calculator, the method that actually computes the fair share of each roommate, and the results are then converted from the calculated form to the output to the text form so that it goes on the terminal.

There is also a block shown for data storage on secondary memory. Note that this is a bi-directional block, the read from secondary memory is coming in into the basic calculations function, and whatever is the result of calculations or the input that has been so far acquired that may be written as the output to the backend storage.

This is how the command line application would look from a fairly high-level view. There are of course many more details that one needs to take care of, but this would form your second assignment eventually next week.

With that we come to the end of this session. Thank you, see you in the next session.