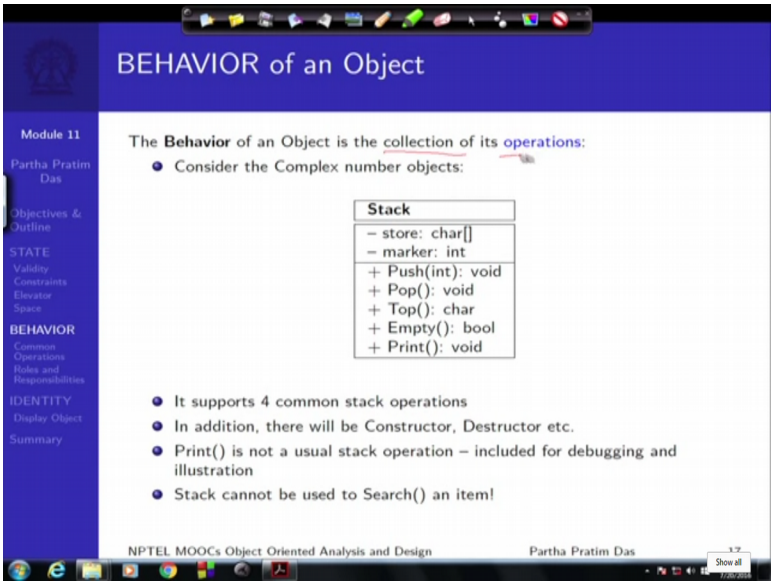
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 19
Nature of an object: State, Behavior and Identity (Contd.)

Welcome back to module 11 of object-oriented analysis and design. We have been discussing about nature of objects, we have noted that an object must exist. It must interact and it must be distinguishable and taking from there, we have noted that every object will have state, behavior and identity. We have talked about state already, talked state charts, how certain parts of the property could be static and certain part of the properties could have dynamic values. Now we move on to discuss the behavior of an object.

(Refer Slide Time: 01:05)

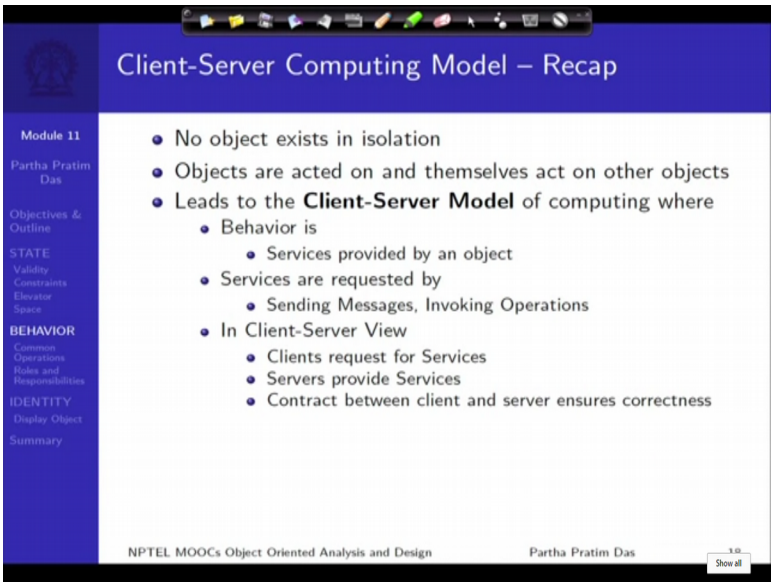


The behavior of an object is a collection of its operations. So, I have a stack object, it has some properties of course to store the items in stack, to mark up what is the top of the stack. But what we have more interested now is a fact that it has different operations. These operations together define the behavior of the stack. So, we can if we just look at specifically then these 2 behaviors, these 2 operations define the LIFO behavior of the stack, Last in first out.

These 4 together these 4 together basically the set of operations that particular role is expected to go for. But in addition, I have also provided another operation to extend the behavior. this operation of print is primarily for the purpose of debugging and for the purpose of illustration. For example, if I am

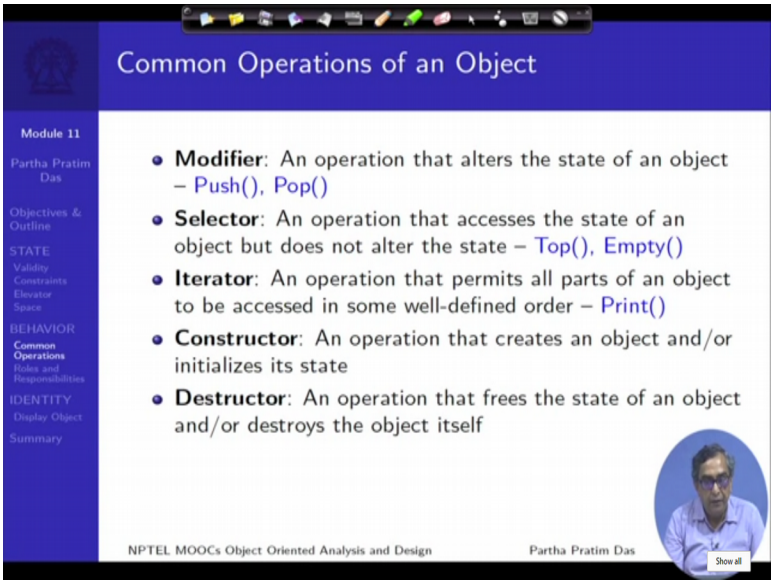
creating the stack object implementing the stack object, and then I need to understand at a stage if my stack operations are working correctly and I might need some help to print the state of the stack at any point of time and therefore add a print operation.

So, behavior will talk about all of these together as a collection and I know that search is not a part of this behavior therefore I cannot search an item in the stack. That's the basic notion of the behavior.
(Refer Slide Time: 02:57)



At this point I would like to dry your attention to the client-server model of object based interaction and computation that we introduced earlier that is the fact that no object can exist in isolation. An object in isolation has no interesting aspect to talk about. So, objects are acted on and they themselves act on other objects. So, if I have object o1, I have object o2, so o1 acts on o2. So, this is acted on and o1 acts on o2. So, either you are acted on or you act on other objects.

And this is what we have seen leads to the basic client server model behavior is a set of services operations provided by the object. Services are requested by these are just variance of the terminology, services are requested by either sending messages or invoking operations and in summary the client server view, clients request for services, the servers naturally provide services. Client request for service, the server provides the service and what carries it through is the message which has to be done according to contract, according to a protocol which is agreed between the client and the server. So, this set of messages or operation basically define the behavior of an object.
(Refer Slide Time: 04:50)



Now of course the objects could have variety of operations in its behavior. but several years of experience of different people in designing objects, in modeling systems have laid to the observation that there are 5 basic types of common operations that usually happen for an object. So why we are designing the object, it will be good to understand what kind of operation we are adding to the whole behavior set. So, these behaviors include the first is all the modifiers.

A modifier is an operation that will change the state of the object, now we will know what is the state of the object. So, if I talk about the stack, then push is a modifier because push what will happen if you push, naturally a new item will get into the store and the marker will move to the top position. Both these properties that the state have will change their values therefore the state of the object will change. Similarly, if I do pop again some value will be removed even if I have shown that removal is not a physical rewrite of the store.

Certainly, the marker has to move to mark that the top position has changed, so the state of the object has changed. So, these are modifier kind of operations. Then I could have the selector kind of operations which in which accesses the state but does not change. When I checked the top element or I check if a stack is empty or not, am not changing the store of the object but am just checking out something based on their values.

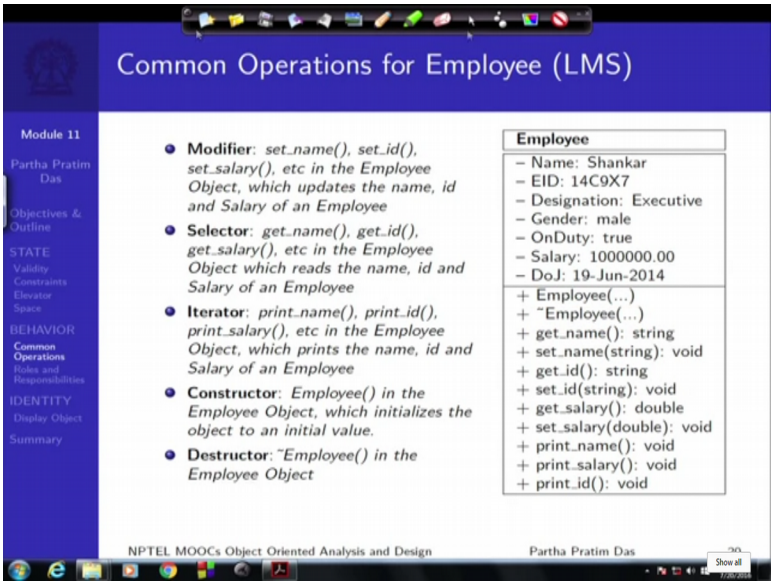
A third class of operations are called iterators and these are very powerful type of operations which are basically goes over all parts of an object and does something, accesses, performs something in a well-behaved manner, in a well-defined manner. So, if you just look at consider the print function in the

stack object, we will find that it is a iterator because what does it do? It iterates, goes over each and every element of the store that the stack has and prints it with the output.

So different kinds of prints and different kinds of ways to go over the different parts of the object in an organized manner is the behavior of the common iterator class of operations. So, these are basically what is the what comes from the functionality of an object certainly what needs to be there in addition to these, or the constructor operation which actually creates the object and the destructor operation by which an object annihilates itself.

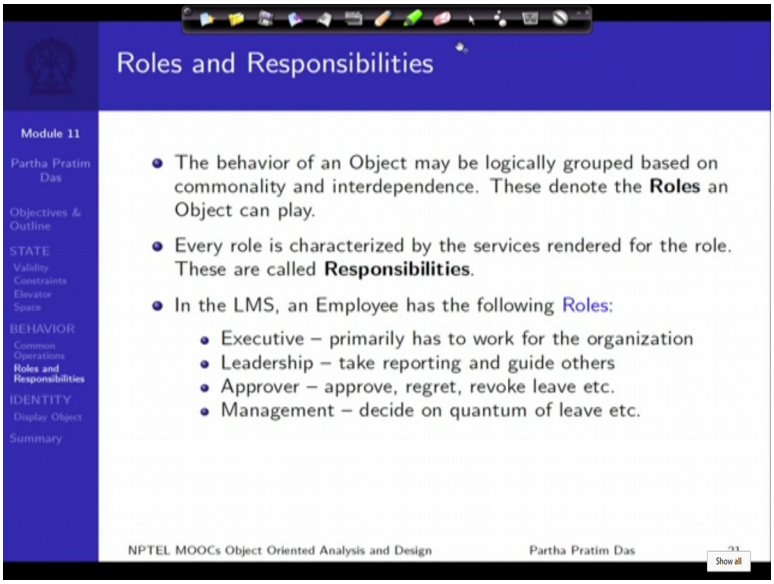
So, though I have not explicitly shown a stack will need a constructor to create a stack and it will need a destructor so that it can destroy itself when the job of the stack is over. So, whenever you design an object and you define its behavior try to be clear in terms of which category your operation falls in. it is usually a good practice not to write very complex operations which could do multiple of these activities in the same operations that is it is modifies as well as iterates and while it is doing a construction. So, it is better to avoid these kind of complex operations for the ease and clarity of design.

(Refer Slide Time: 09:02)



So, this is ah another example for primarily for your self-study I have taken an object instance from the lms example, an employee and I have shown that for that employee all that different kind of some of the operations that you can have, how those operations are classified according to being a modifier, being a selector, being an iterator or being a constructor or being a destructor. We can see the constructor is here, the destructor is here and so on.

(Refer Slide Time: 09:43)



we have roles and responsibilities of behavior. an object which is not very simple will certainly be having a large number of operations and as it happens that often these operations are kind of groupable. I can put similar operations interrelated operations together. So, in a total behavior if I group the operations according to their semantic commonality according to their interdependence, then I will get the different roles that the object play as a part of the behavior and it is possible that roles may overlap that these 2 different roles may have certain things which are common.

So just to take an example is if we if we look at the different activities that the employees do in our leave management system we have seen that there are some employees perform the role of being an executive. There is their main task is to work, primarily is to work for the organization but some employees whom the document says the lead employees certainly take reporting of other employees, executives and guide them, providing leadership. So those who work as executive do not have the role of leadership.

Those who are le lead or manager, the lms specification say can actually approve the leave for others, can revoke an approved leave, can regret an applied leave and so on. so, they have yet another role which is what I am calling here is an approval role. Then managers are certain employees who have management roles, for example they can decide how much leave an employee should get and so and so forth. So, this is a context of having roles and behavior it usually becomes a cleaner design

if we have a clear understanding about the different roles that the behavior of an object will have. And every role will then be associated with the responsibilities or the set of operations that the particular

role is expected to perform.

(Refer Slide Time: 12:27)

So, this is again for your self-understanding. These are 3 objects been shown from the lms system and the left most one is for an executive so to illustrate that I have introduced a proper called designation. So, the left most one is for an executive, next is for a lead, next is for a manger and here you can see what are the roles that they these objects will play. Now if we look into the operations, we will find that this role of an executive is responsible for these operations

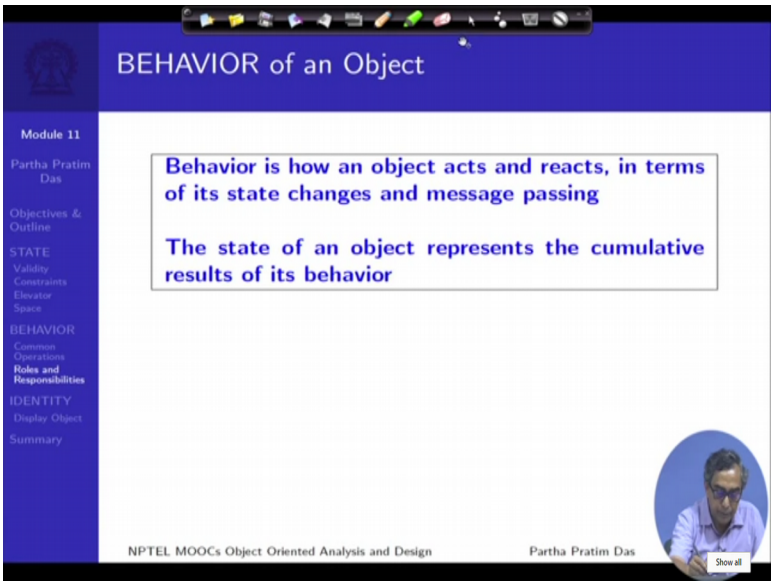
that is recording his or her own attendance, applying for leave, cancelling a leave, availing a leave that has been approved and so on. so, the executive has say if the role is of an executive, then these are the roles, these are the responsibilities. Now if we look into an employee object which is for a lead, then certainly that employee has an executive role and associated responsibilities but the employee also has other roles like leadership and approval and associated responsibilities of approving leave, revoking leave, taking reporting and so on.

if we look into the manager, we have yet one more role and the associated responsibility for that role. So, as we can see here the different objects all are of the all our employee objects, their behavior has different roles and the roles in this case are overlapping, for example a lead employee actually performs 2 roles, the role of an executive and the collective role of leadership and approver, a manager performs all the 4 roles and correspondingly they have overlapping responsibilities to perform.

So, it will usually of course it is not mandatory that you will have to define or you will have to demark

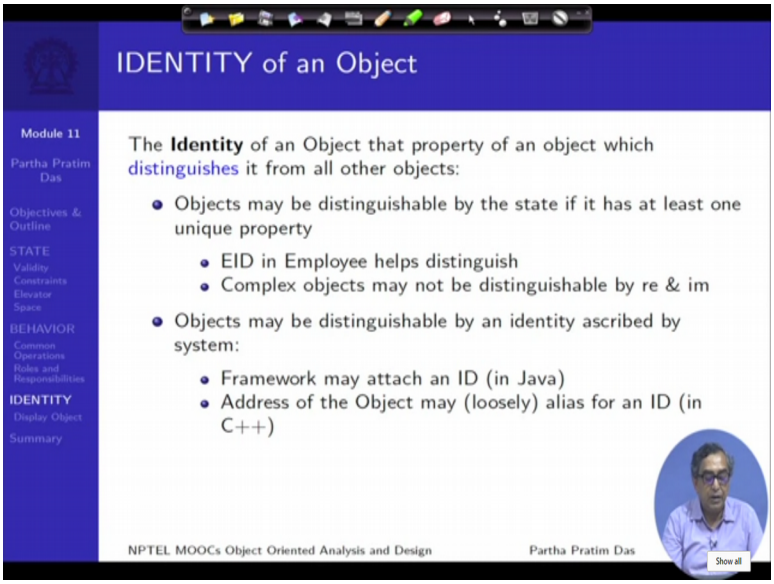
it what are the different roles of a behavior and the corresponding responsibility but as it turns out it usually gives you a better design if you are clear in terms of a different roles in the behavior of an object that you define and you are able to associate the operations as responsibilities of different roles.

(Refer Slide Time: 15:19)



So, to summarize the behavior of an object is how an object acts, how an object acts and reacts. Act means proactively, react means in response to what somebody else has done in terms of state changes and message passing. So, if the state changes naturally the object will have and if the message have received the object will react. The state of an object represents the humiditive result of its behavior of any point of time all these operations has performed we have seen whether particularly as we have seen that any kind of modifier operation then the state will keep on changing and that is the result of the behavior that we will get to use.

(Refer Slide Time: 16:12)



Next, we have talked about 2 aspects of state and behavior. next we will talk about the identity of an object. The identity of an object is as we started saying every object must be distinguishable. So that's the simple thing, the identity of an object is a that property of an object which distinguishes it from all other objects. So that's something very critical for the objects to exist and behave, without identity, we will not be able to actually do any reasoning in the system.

Now how do I put the identity, how do an object get that identity, it could be grossly, there are grossly 2 major ways the identity can be thought of. 1 is objects may be distinguishable by the state, that they can say that if I know the state of the object, I know which object it is. How do I do that? It's very difficult to do that. But I will be able to do that if the object has at least one unique property. I mean those of you who are familiar with some bit of database systems particularly of the relational kind will identify this unique property as a key.

Now it may be a single property, it may be a collection of properties that uniqueness here mean that a property is unique or a key if or any 2 objects that they can have if that particular property will always be taking different values. So, consider that the employees in the leave management system the employee object has an employee id naturally there cannot be 2employee objects with the same id so therefore this is a distinguishable property

and the object no matter what else happen just by the state of object which comprise of various different properties like the name, the date of joining and whether the employees on duty and so on will also have the eid and we will be able to distinguish the employee object by the eid part. So that's one

way of distinguishing objects but you can just extend that if we are talking about complex objects then we may not be possible to distinguish to complex objects by their state

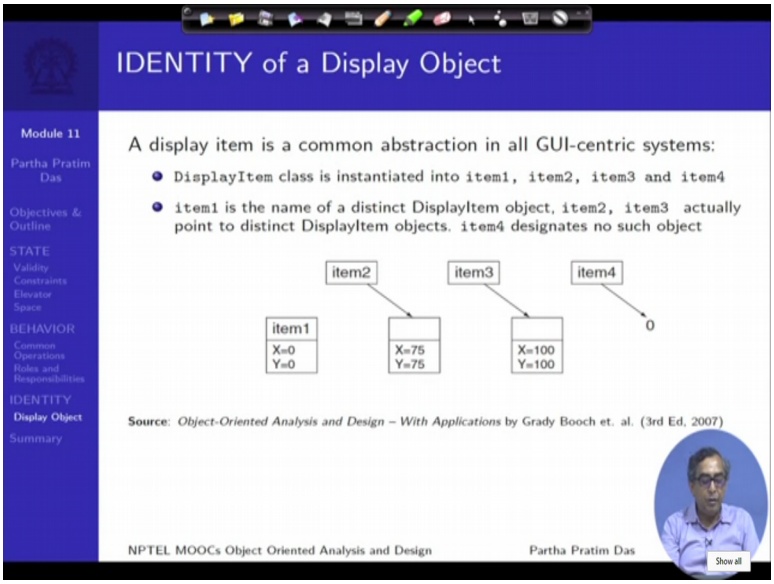
or by the value of re and im, I can have 2 complex objects which have the same real part and the same imaginary part and they are just 2 different objects. So, if so it is not necessary that all objects will have a unique distinguishable property so the objects may be distinguished by the identity that is described on an object by the system. That is a system says that well to I distinguish an object based on or to identify an object or to setup identity for an object based on a distinguishable property kind of I open up the object,

look into the state and say well this is the object but in stack the system that manages these objects could put an id on top of that object and this is what is done typically in a lot of programming systems for example those programming systems which are based on framework I would just remind you that we have talked about the fact that in the evolution of programming languages we talked about that all a major part of the recent programming systems is based on framework

whereas there is a certain runtime that runs like jvm for java or dot net for c sharp and so on which manage the objects that exist at the time of execution so if there is the framework that goes with the programming system then the framework may attach an id to the object? So, it does not need to look into what is inside state of the object but based on the id that it attaches separately you can figure out what is the identity of an object but systems which do not ha or the programming system or the programming languages which do not use a framework that is do not use a separate runtime system would not be able to do this.

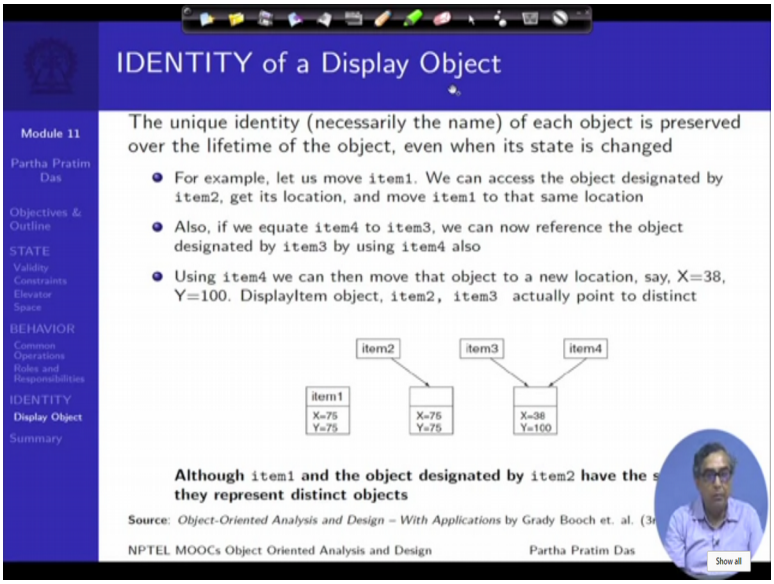
So, for them it's typically the address of the object gives loosely or works loosely as an alias for the id. So, you say what is the identity of the object is wherever it exists. Now this kind of an identity is somewhat difficult to work with because without changing the object in anyway if I just relocate the object from one memory address to another then potentially I would be impacting the identity of an object so you will have to be careful in those systems in terms of dealing with the identity of the object.

(Refer Slide Time: 21:41)



So, to look little bit more into this let us just take consider a simple example, this whole example is very well developed in the book in the OOAD book is just a hint from there, it talks about display item is a common abstraction in gui centric system. So, the idea is you have a kind of a canvas you have kind of a canvas the gui canvas and you have different objects on that canvas. And where they occur in the canvas is defined by a pair of values x and y coordinates of the center of that display object. So here in this instance it shows the 4 different 4 objects this is item 1 is 1 display item I display object and item 2, 3 and 4 are pointers to display items. of that item 2 points to this particular display item which is unnamed but you can still identify it by the pointer, item 3 points to another and item 4 is kind of a dangling reference which currently does not point to any display item. So, you can see that in in this case we have different identities, some are maintained directly by the object as an item 1 or indirectly to the pointer if there is no direct identity of the object.

(Refer Slide Time: 23:25)

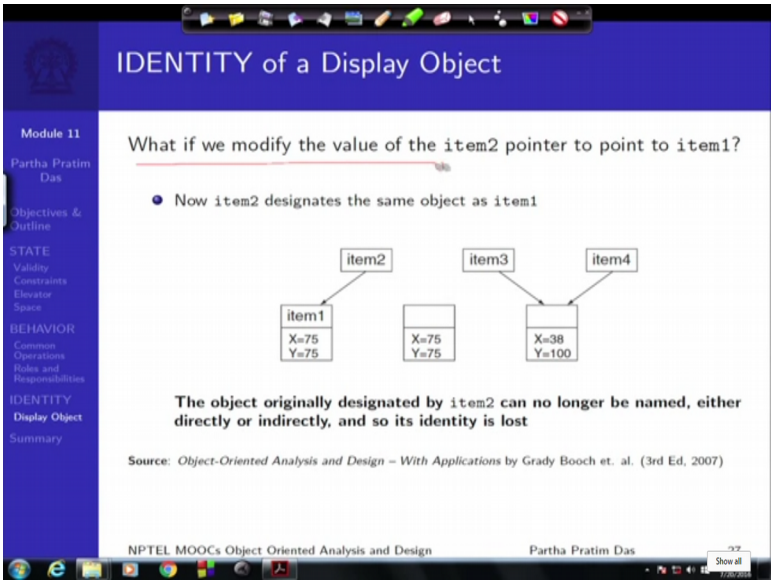


Now let us see what happens when some operations start happening with this that the unique property that each object will preserve through the over the lifetime is the identity even when its state will be changing so let's say on that on that calibers on that gui display system suppose item 1 has been moved item 1has been moved so earlier it was in 0, 0 and now it has been moved to a location 7 5 7 5, 75, 75. How did you move that?

We used item 2 got the location of this unnamed object and we have moved item1also to that same location. Now suppose we equate item3 and item 4, item4 you remember, item4 earlier was pointing to not, it was not pointing anywhere. Now we have equated through m4 so both refer to the same object. The 2 basically 2 different references but the same object and using item4 we have actually edited the x coordinate that is we have actually moved that object therefore if you try to move that object also we will get it in a new location.

So, what we can see that item1 and item2 have the same state, both are located in the same 75, 75 locations but they necessarily represent distinct object this is the cruces of the identity that a state may not actually define it. Here where is the identity coming from? The identity here is coming from the identity that is written on the item1 object and the identity of the pointer which holds this are named object which holds this unidentified object.

(Refer Slide Time: 25:54)

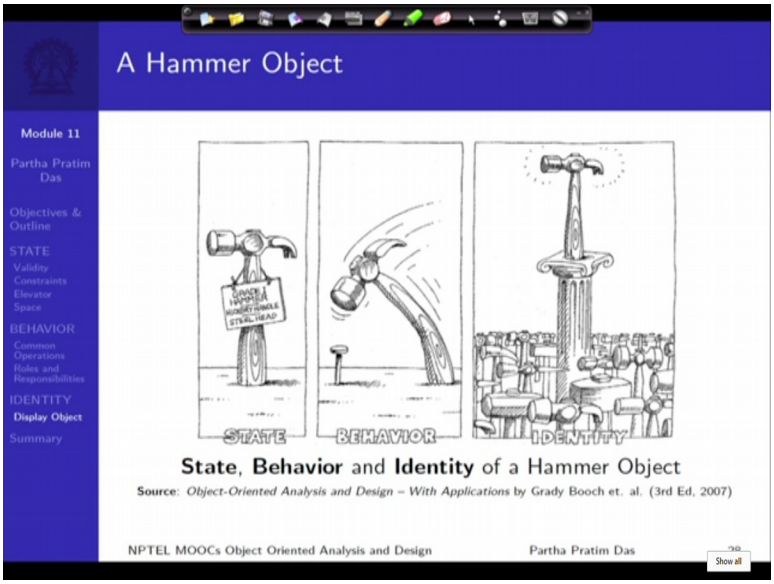


Let us go further let us see what if we modify so we are just asking this question what if we modify the value of the item2pointer to point to item1? So earlier it was pointing here, we change and we make it point to item1. So now what will happen item1 and item2 basically both of these actually mean the same object, the 2 identities but actually they mean the same object and the worst part is this particular object has lost its identity?

It did not have it was not named, it was not designated anywhere, it was identified by the pointer which was holding this address of this object, now that has moved on this side, the pointer has move on and therefore this has become kind of lost its identity and therefore it cannot accessed it cannot be managed directly or indirectly. So, when you can see that identity of an object is not only, it is it is critical in the sense of abstraction because unless objects have identities how do they interact with each other,

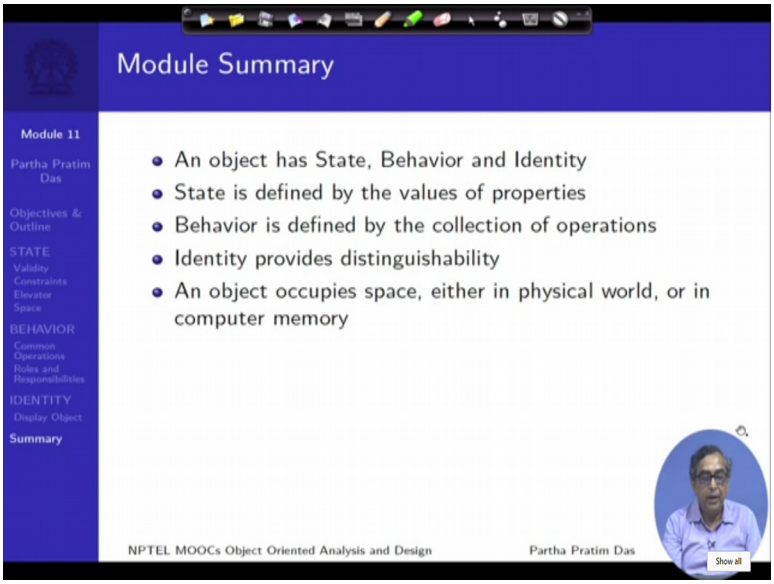
how do they act and react which the behavior need but we note that their identity may not be always derivable from the state variant or certainly the kind of behavior that they demonstrate but identity is third dimension to an object where you will need to somehow be able to distinguish objects and 2 objects having the same state can still have different identities and then object who has lost its identity is lost for the system and so in the design all these aspects of identity will need to be born

(Refer Slide Time: 28:06)



And now we come to the final conclusion so this I would like to conclude with this nice kind of cartoon from the OOAD book it says what is a state and what is a behavior, the state basically says all these grade 1 kind of handle, this steel head and so on to the to those of the values, the properties who defines the state of a hammer, this is certainly the behavior the behavior of the hammer is to hammer in the nail and the identity is this is this my hammer which stand still amongst the pool of different hammers. A symbolic diagram a representation but very clearly thinks the core point of the nature of an object.

(Refer Slide Time: 28:59)



To summarize on the module, we have seen that the object has state, behavior and identity. A state is defined by the values of its properties and we have seen different variations on that in terms of static property and property that have dynamic values and so on. the behavior is defined by the collection of

operations, we have seen what are the different common operations that a behavior should include and we have also talked about how the role and rules of a behavior should be properly identified to designate the responsibilities of the associated operations that every role must have

And finally we have seen that identity provides a distinguishability and it could be inherent to an object if there is one or more distinguishable properties but in many cases but in absence of such key properties, an object may need to carry an identity which is either provide by a dynamic framework supporting the whole object based implementation or merely by the location of the object or the address of the object and there are potential risks in terms of those wham of which we have already shown.

So finally, the objects occupy space either in the physical world or the computer memory and it is very critical to keep in mind that an object must exist, must interact and must be distinguishable.