Database Management System
Dr. S. Srinath
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 36
Object Oriented Databases

Hello and welcome. We were looking at the ongoing saga of, trying to understand data management in terms of database management systems. We have looked at several different topics in database management. However there is some kind of a common theme or an implicit theme in a database management in whatever topic that we are looking at namely that database or databases are primarily or essentially represented using the relational data model. So what is a relational data model? That is the data pertaining to the UOD or the universal discourse is maintained as a set of tuples or as a set of rows in a table. And the main assumption here is that every possible kind of data can be reduced to a set of tuples.

(Refer Slide Time: 02:34)

In this lecture and in the following lecture, we will kind of generalize on this assumption or we won't accept this assumption and look at other kinds of database management requirements where data cannot be in a sense easily mapped on to the relational model. In this context specifically we shall be looking at the object oriented databases that is shown in the slide here. So let us look at what kind of data that we are talking about when we are looking into object oriented databases but before we begin, in fact object oriented databases have been very popular in the last decade of the twentieth century in the sense that in the 1990's.

But however they were not as widely successful as say the relational model database. Mainly because the object oriented databases do not have a sound theory that is they do not have a nice little mathematical model that describes the complete data model. As opposed to the relational data model where you have the relational algebra or the tuple relational calculus and so on where the entire data model is amenable to a nice theoretical framework. So it's because of one of these reasons which is probably cited as the reason why object oriented databases did not sustain lot of interest. But however the contraries also true to some extent that is object oriented databases have been in use or have been put to use in several different applications mainly CAD application, computer aided design of say electrical circuits or mechanical circuit, mechanical design and so on. And they continue to be used and there are quite a few commercial implementations of object oriented databases.

So in this lecture and the next when we are talking about object oriented databases, we kind of implicitly assume an example application of a CAD that is computer aided design scenario where users would be using computers to perform electronic design or electrical design where an electronic design comprises of several different components, I might have an IC, I might have a capacitor, a resistor, a transistor and so on. Each component having its own characteristics and having its own behavior and so on.

(Refer Slide Time: 05:07)

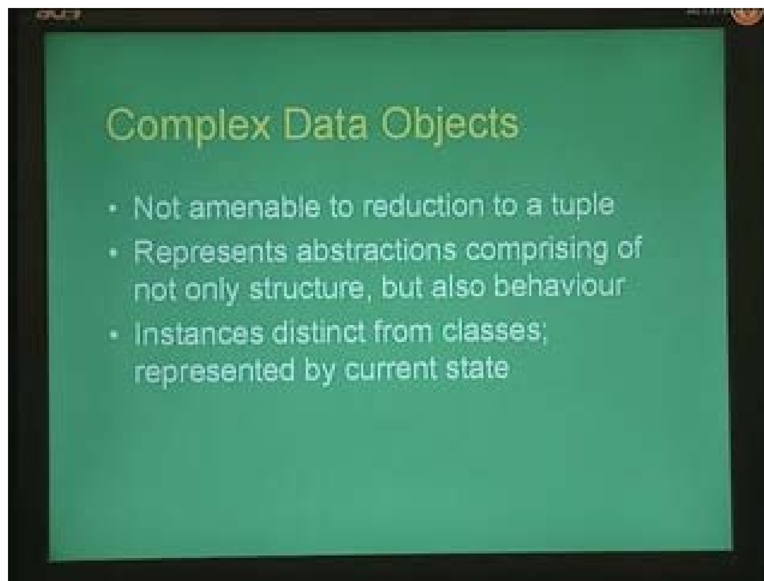
But that's the implicit assumption that we are going to make but that doesn't really necessarily mean that object oriented databases are suitable only for CAD applications, of course there could be several other applications as well. So let us come back and look into what kind of complex data objects that we are talking about when we say that we are going to generalize or we are going to move away from the relational model and look into other kinds of models.

Have a look at the first example here say it's a multimedia databases. What do you understand by the term multimedia databases? Databases that store multimedia objects. What do we mean by multimedia objects? You might have, you would have encountered several kinds of multimedia objects, if you have let us say work with any GUI based operating systems like say windows, windows XP and so on where you encounter objects like menus, scroll bars, drawing areas then something like, when you click something that there is a sound that appears and there is a for specific kinds of events, specific kinds of sound and light so to say, a messages are thrown to the user and so on and there is flash animation and so on.

And look at the second kind of application that we have talked about that is namely the CAD. What would the typical CAD application for electronic design comprise of? A typical CAD application, in a typical CAD application the user should be able to let us say select a PCB a printed circuit board or select a resistor or a capacitor or a transistor or a particular kind of IC and voltage source and control grounds and current sources and so on and so forth.

Now the main theme or the common theme between these two applications is that both of these applications are made up of fundamental objects which form the building blocks of these applications. So multimedia applications are built from several of these different objects, they could be menus or scroll bars and so on.

(Refer Slide Time: 07:31)

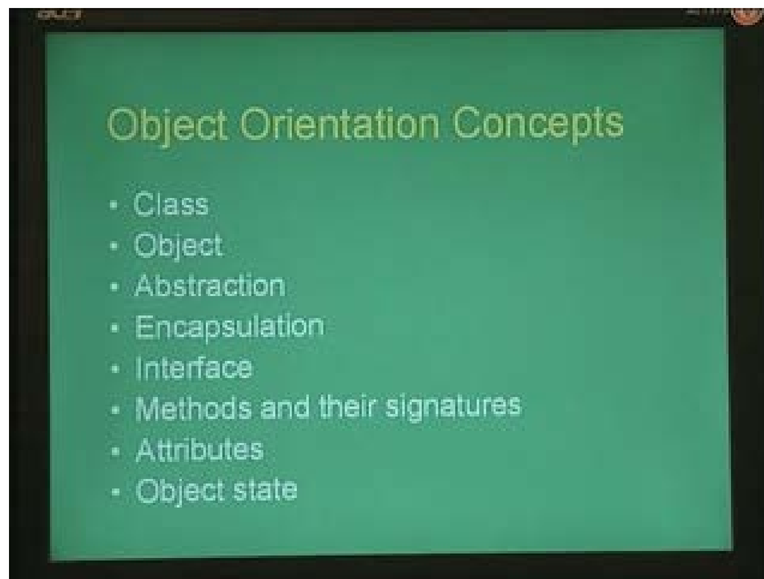


Similarly CAD applications or CAD projects in a sense or multimedia projects and CAD projects are in a sense built using this fundamental objects. CAD projects would have a PCB or IC and so on and so forth. So what are the characteristics of these complex data objects? It is quite apparent that these data objects are not easily amenable to reduction to a tuple. That is we can't really reduce all of this to one set of, one tuple having a set of characteristics because there is much more to an object than a set of different attributes.

So what is this, what comprises this much more essentially the idea of behavior of an object? A transistor for example behaves as is represented not only by a set of attributes saying what kind of a transistor is that or is it PNP or NPN or so on and so forth whatever else goes into describing the attributes of the transistor, in addition the transistor also has a particular kind of behavior. You can apply voltage at one of the pins and measure the voltage at one of the other pins and so on and so forth.

So an object does not simply represent a set of attributes but it also abstract, rather an object is not merely a structural abstraction but it is also a behavioral abstraction. That is when I say transistor, the kind of behavior that the transistor emulates is also abstracted by the object. And in an object there are several instances of an object that can belong to the same class. So I could have several different instances of the same transistor and each of this different instances may have different set of attributes at any given point in time. That is the instance variables of each of these different transistors could be different that means each of the transistors belonging to this particular class has different states at any given point in time.

(Refer Slide Time: 09:56)



So let us briefly take an overview of object orientation concepts. Object orientation concepts, here I am essentially looking at from object oriented programming point of view. The idea of object orientation came from programming and there say several kinds of OOPLs or object oriented programming languages which were started right from the early 70's and so on.

So let us look at object orientation concepts from a programming point of view and then we shall look into how each of these changes when we consider object oriented database systems. Now the fundamental building block in an object oriented system is of course the object but then an object represents an instance and an object belongs to a particular type and here this is called a class.

So an object can belong to a particular class or rather we define specific classes of objects and then we instantiate different objects of specific classes. For example we could define a class of objects called cars and we can instantiate an object of type car which specifically points to one specific car rather than the type of all possible cars. Then the idea of an object is to provide an abstraction to the user. So depending on what the application is when an object of type car is created, it represents an abstraction called car. That is each car is supposed to have a certain properties not just structural properties in terms of what attributes they have but also behavioral properties, what can you do with the car and so on.

For example take something like menu in multimedia database. So menu is an abstraction that is it not only says what are the attributes that make up a menu but what is the behavior of the menu as well. That is the menu should provide a list of items and there is a default selection by the menu and the user should be able to scroll up or scroll down the menu and so on and so forth. So an object provides an abstraction or an object emulates an abstraction of not just structure but also of behavior. And how is this abstraction provided? Through the notion of encapsulation that is an object encapsulates structure and behavior within its fold. That is an object is defined by a set of attributes which define the structure that is described by the object and a set of methods or function calls that operate on these variables which define the behavior that is abstracted by this object. And of course there is the notion of an interface. That is interface is also called the signature of an object that is an object only exposes or the only thing that theoretically at least that is exposed to the outside world is the interface of an object.

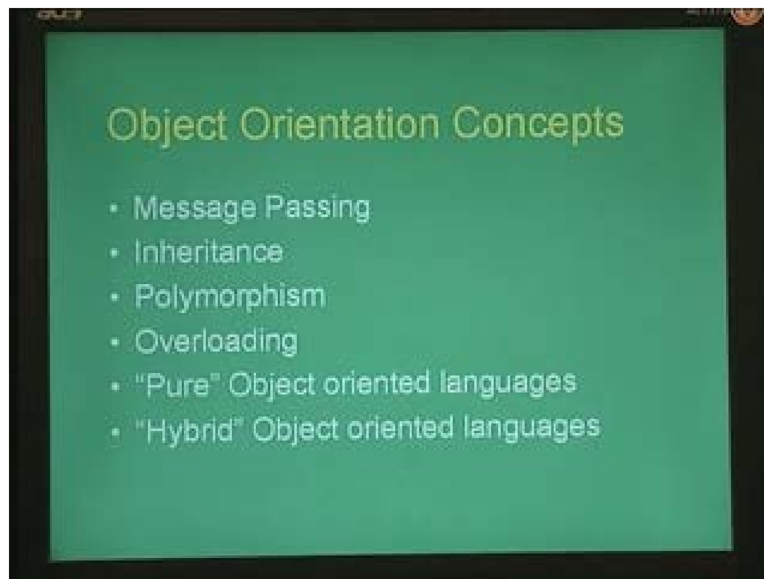
And all external world entities should interact with the object through the interface by calling particular methods and changing attributes and so on. So again just to give an example or just to give an analogy, a car for example gives an interface in the form of a steering wheel and a gear. So you can interact with the car only through the steering wheel and the gear and of course the pedals that is the brake pedal and the clutch pedal and so on. You cannot directly go, when you are driving a car you cannot directly go and manipulate how the engine behaves for example or how the wheels behave and so on. You have to deal with the car through its interface

So interface is the, as far as the user is concerned the interface is the signature of the object. If you want to learn to drive a car, you should know how to handle the steering, how to handle the break and clutch and gear and so on and so forth. So you should be able to know how to handle the interface rather than what lies within the interface. And of course interface, in software interface is made up of methods which are function calls which change the state of the object and methods themselves have particular signatures that is each method requires zero or more input parameters and has 0 or 1 output parameter as well so that forms the signature of a method. And attributes of a class is the set of variables that define the state of the class for example again in a car the attribute would be something like which gear the car is in or what is the speed of the car or what is the acceleration of the car and so on and so forth which basically describe what is the current state of the car.

So any method call would change the or would influence the attributes of this object. And object state of course is a function of what are the values of different, each of these attribute. So the state of an object is something like let us say the state of the car can be defined as say cruising when speed is so and so and the gear is in over drive and so on and so forth. So basically you define set of values and say now if these are the set of values then this is said to be the state of an object. And there are again some more concepts pertaining to object orientation which may also be important. That is some notions of say message passing.

So when an external world entity invokes the method of an object, it is said to have pass the message to the object and the message in turn invoke the object that is invoke method of this object. And some more concepts of object orientation which are particularly useful are the notion of inheritance, polymorphism and over loading. That is when we define a class, we can define a generalization specialization relationship which we also saw in let us say the enhanced ER model where a generalized class represents a general, more general entity than its specialized classes.

(Refer Slide Time: 15:21)

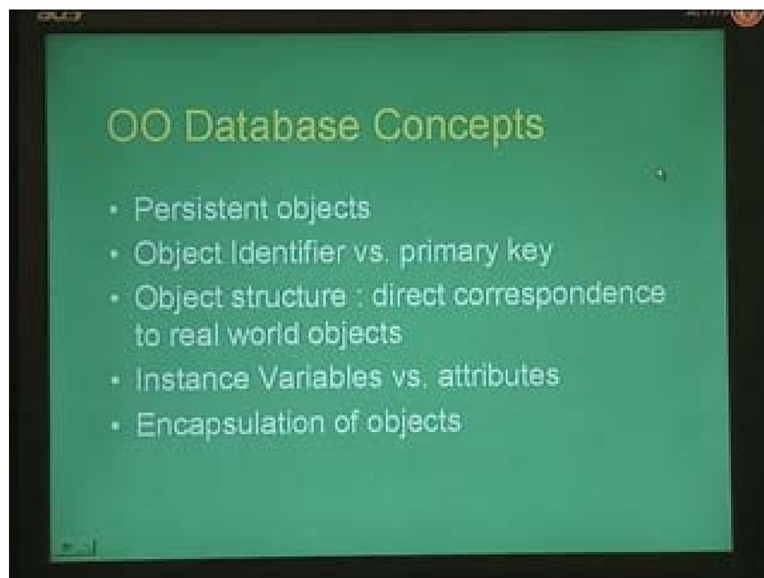


That is wherever an object of a generalized class is required, it should be safe to substitute it within an object of a specialized class. Therefore suppose I have a generalized class called say SUVs or whatever sports utility vehicles and so on. So and there could be different classes of SUVs whatever Qualis and Scorpio and so on and so forth, so several different kinds of SUVs. And what constitutes a correct generalization specialization relationship? Wherever I need an object of the generalized class it should be safe to substitute it with an object of the specialized class. So the specialized class is said to have inherited properties from the generalized class and of course extended on the properties or over ridden on the properties and so on. So a specific property or a specific method for example again coming back to CAD databases or which we said we are going to have a running example.

So suppose I have a CAD database and I have a generalized class called say a transistor and I have a specialized class called a specific kind of transistor of some particular number. So whatever behavior is specified by the generalized class can be actually overridden by the specialized class and in a sense the same method signature seems to be giving different kinds of behaviors depending on which object of the specialized class is substituted. So that brings in the notion of polymorphism that is the same signature, method signature giving rise to different kinds of behaviors that emanate from the system. And there is also the notion of over loading which is a feature that present in many object oriented languages where slight changes in method signatures can be used to perform different classes of the same activity.

For example we can say something like add. Now when I say add to, I can say add int, int where it takes in two integers and gives out an integer. Now the same add could be defined as int, float or float, int or float, float where you can add different combinations of integer and floating point numbers and return back. So at run time the message passing framework is going to determine which kind of add is being called depending on what is the type of the parameters that is passed. And then there are pure object oriented languages where everything is an object there are no what are called as native types. So every single entity like an integer, every integer or every character is an object and there are no native or fundamental data types other than objects. And then there are hybrid object oriented programming languages where which do allow native types and of course what may be termed as semi object oriented programming languages where you can perform both object orientation and procedural programming in the same language.

(Refer Slide Time: 20:05)



Now let us come to object orientation as pertaining to databases. Now what extra features to be required in the concept of object orientation, when we talk about databases? The main concept that is required for databases is the notion of persistence of an object or persistent object. What is the persistent object? A persistent object is something that can

exist persistently or permanently that is the objects can exist even after the program using the object has finished. That means the object exist on some persistence storage like disk and can be recreated or can be reread back from disk whenever required.

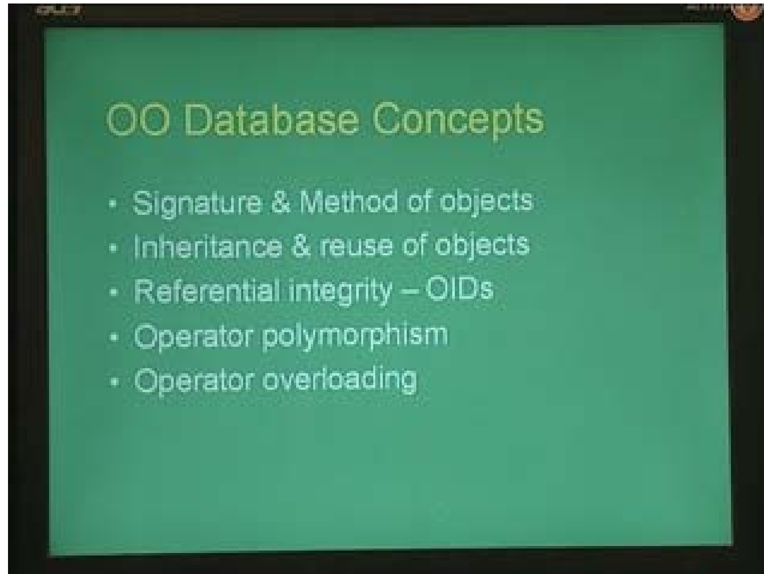
Now for storing persistent objects in object oriented databases, another important requirement is the notion of an object identifier. That is it is important to uniquely identify each persistent object that is stored in the database. You might think of this as a as a primary key as in that we discussed in relational database systems but there are some slight differences between an object identifier or an OID versus a primary key.

An object identifier or an OID is automatically created by the system whenever a new object is added to the system, whether the user specifies it or not. On the other hand it is the user who specifies what forms the primary key in any database relation. And of course in pure relational algebra, each tuple is unique that is a table is a set of tuples not a bag of tuples. Therefore in the worst case the entire tuple form a primary key for the table. However object identifiers are separate attributes that is the entire object cannot form the or cannot uniquely identify given object. This is because two or more objects belonging to the same type can have the same state and hence be indistinguishable as far as their other attributes are concerned. But they still would represent two different objects for example I can always have two different transistors in any given circuit board which have the same input voltage at the same time or same input or output voltage at each of its pins at the same time. However they are still different transistors.

So by default an object database is a bag of objects that is all attributes of an object need not necessarily uniquely identify an object. So we necessarily require an OID or an object identifier. And the way objects are stored in databases are as far as possible should be in direct correspondence to real world objects. So I can store an object like transistor or capacitor or PCB or whatever, so where you have direct correspondence to what one can see tangibly in the real world. And of course there are several different attributes that define an object which alter the state of an object and of course these variables for attributes are defined at a class level whereas when an object is instantiated, these become instance variables. And the instance variables of different objects could be different even though they belong they represent the same attributes. And just like an object oriented programming languages, objects are defined by signatures which are the interfaces of objects and even methods have signatures that are defined for the objects.

And every other notion in an OOPL are also reused here something like the inheritance and reuse of objects that is when you inherit the base, the derived class or the specialized class reuses certain properties of the base class that is because it inherits certain properties of the base class, we can think of it as some kind of reuse.

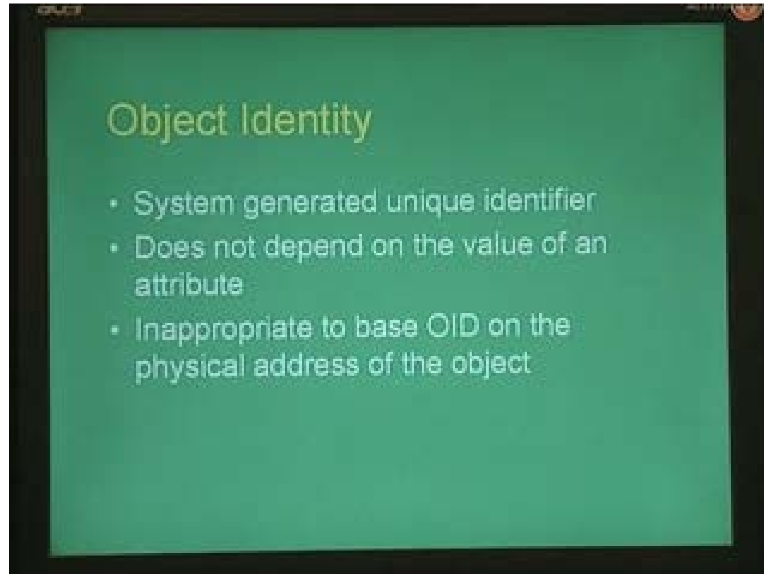
(Refer Slide Time: 24:05)



And there is also a notion of referential integrity in object oriented databases by the use of OIDs. That is every OID, suppose an object a refers to another object b, this reference is captured by putting the OID of object b as an attribute of object a. And referential integrity is enforced by ensuring that at every point in time, the OID that is represented as an attribute in any given object is always a valid OID. And of course there is operator polymorphism and overloading and other concepts that are common in OOPLs.

So let us come back to the object identity aspect like I said before the OID is mandatory in any object oriented database system. And this is usually a system generated unique identifier that is the user need not even be aware that there is an OID that is created for each object. However the system by itself creates unique object identifiers and of course the OIDs have no relationship to the values of attributes. That is the set of all values of attributes of an object need not necessarily, uniquely identify a given object.

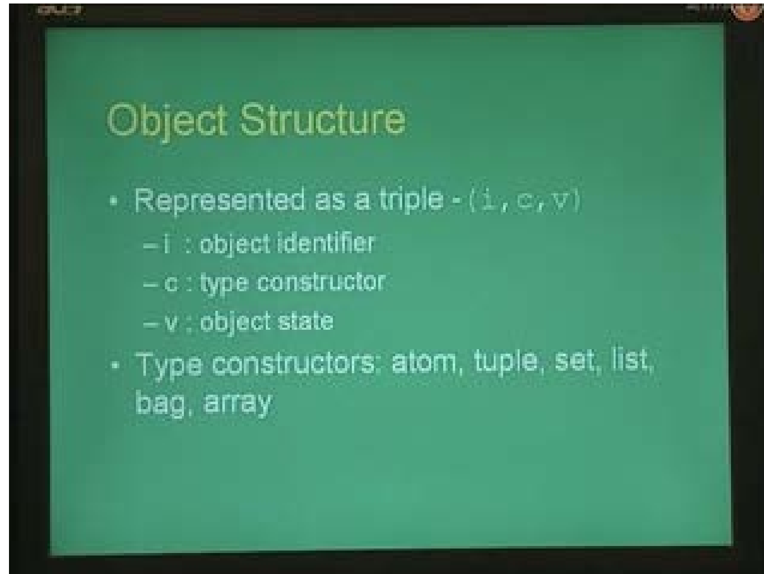
(Refer Slide Time: 25:53)



And usually OID is a logical number and it is not advisable to base OID on the physical address of an object. Suppose I have stored object in a particular directory tree, we should not keep the directory tree as the OID of an object because of several reasons like if the database is migrated to a different system or if the directory tree is changed then the OID changes and the object becomes in accessible.

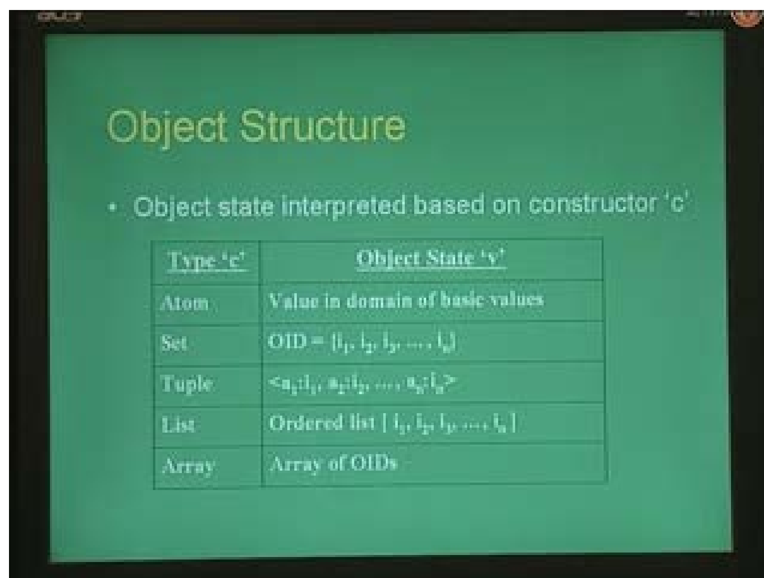
Now like I said before every instance of an object is characterized by a state of an object. Now how do we define the state of an object in terms of the database or in the database parlance? This slide shows (Refer Slide Time: 27:00) a formal model of how the state of an object is represented or in a sense the structure of an object. The structure defines the state space in a sense so the different kinds of state that an object can be. So an object structure is defined by a triple comprising of three values i, c and v where i is the object identifier and c is what is called as the type constructor and v is the object state. So i is the well-known OID that we have been talking about and c can be, c is what is called as the type constructor that says what type of type in a sense or what type of value is this going to be. And usually object databases define different kinds of type constructors like atom and tuple, set, lists, bags, arrays and so on.

(Refer Slide Time: 27:52)



An atom type for example defines a specific atomic value. So I can say atom and then give a value of 5 for the value. So this object represents an atomic entity whose value is 5. On the other hand I can represent a tuple also as an object. So instead of one single value, a tuple represents a list of values, an ordered list of values and list of atomic values essentially. And a set is an unordered set of values, unordered collection of distinct values that we can take up. And list is similar to a tuple except that in a tuple, the size is fixed, the size of a tuple is fixed but in a list different instances may have different sizes for the sequence of values that we can take and bag of course is a multi-set that is a set with a set with repetitions and so on.

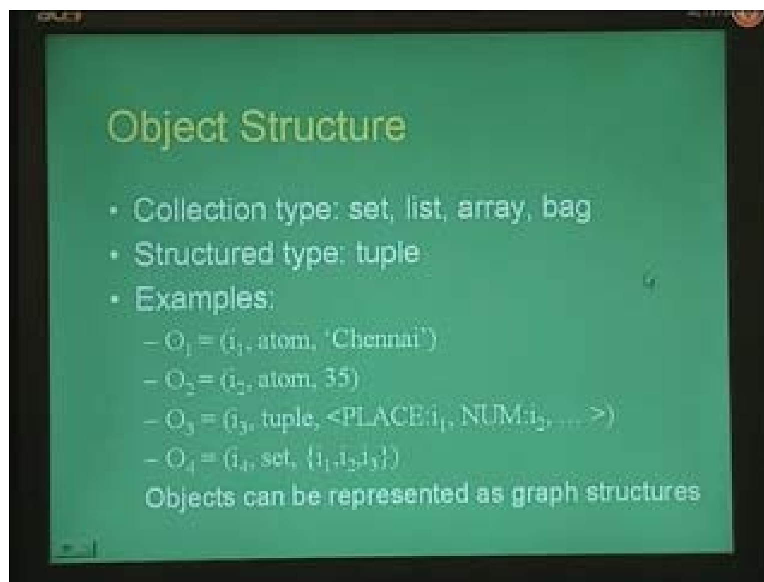
(Refer Slide Time: 29:27)



So this slide shows some examples here where we defined all these things already that is when type c is atom, object state v would be one particular value from a domain of basic values and when it is a set, it is the set of values and so on.

Now this slide shows some examples here. Let us say I define an object O_1 as a triple where i_1 is the OID of the object and the object is of type atom and the value of this object is Chennai. That means this object essentially stores an atomic value or an atomic entity called Chennai as part of this object.

(Refer Slide Time: 29:36)

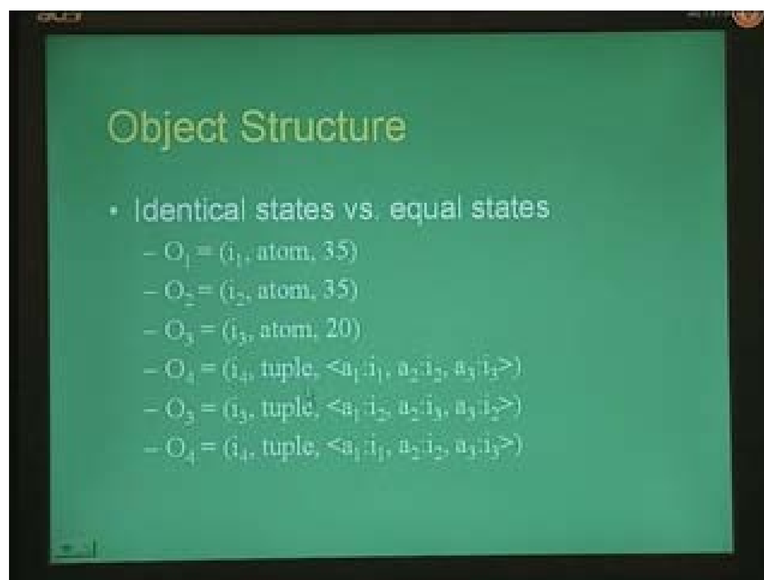


Similarly O_2 has as OID of i_2 and it stores an atomic value called 35 and O_3 is a tuple wherein each element of this tuple is an OID that is i_1 is a OID belonging to the class called PLACE. So O_1 that is i_1, look here that i_1 refers to this i_1 here. So this i_1 is an object, i_1 basically represents an object called O_1 and O_1 belongs to a class called PLACE. And similarly i_2 represents an object called O_2 which belong to a class called num. So this O_3 is a tuple of different OIDs where different, in a sense it's a composition of different objects of different types in the form of a tuple.

Similarly O_4 is a set comprising of 3 different OIDs O_1, i_1, i_2 and i_3. So as you can see here, it is possible not only to represent specific atomic entities it's also possible or rather or even collection of atomic entities, it is also possible to start composing objects, one object inside another. For example O_3 in a sense is a composition that is made up of O_1 and O_2. And similarly O_4 is a set or a collection that contains all three elements that is O_1, O_2 and O_3. So depending on these kinds of associations between objects whether it is a composition association or some kind of a whatever other kind of association that we can define, an object database can actually be represented as a graph structure, so where each object in turn has some kind of an association whether it's a containment or inheritance or some other kind of an association with other objects in the database.

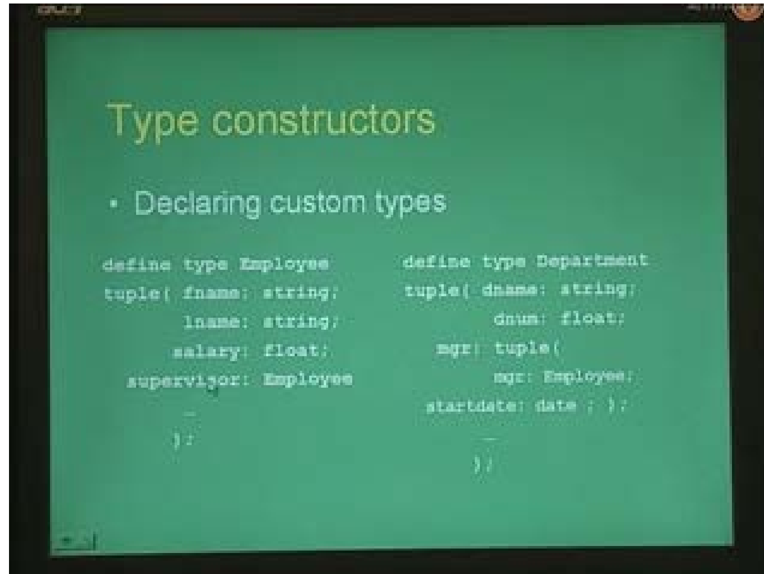
And when we are talking about the states of objects, remember I had mention that two or more objects may have the same state but that doesn't necessarily mean that they are the same object because as long as their OIDs are different, they essentially refer to different objects. So this slide shows such an example (Refer Slide Time: 32:41). So at any instance of time I may have two different objects O_1 and O_2 whose states are the same. That is they represent one atomic value whose value is 35 and there is one more object of the same type called num that we defined in the previous slide which represents a value called 20. However even though both of these have the same state and this has the different state, all three are different objects namely because the OIDs are different i_1 i_2 and i_3.

(Refer Slide Time: 33:17)



Similarly, here these objects i_1 i_2 and i_3 or i_2 i_3 and i_2 or whatever, so these two O_4 and O_4 here have the same state that is a_1 i_1 a_2 i_2 a_3 i_3 and in this case it does represent the same object. Why because i_4 is the same as i_4 here so at the end of it is just this OID which determines whether two objects are the same even when this regardless of what is the state of this objects. And different object oriented database systems provide different mechanisms for defining custom types or custom classes.

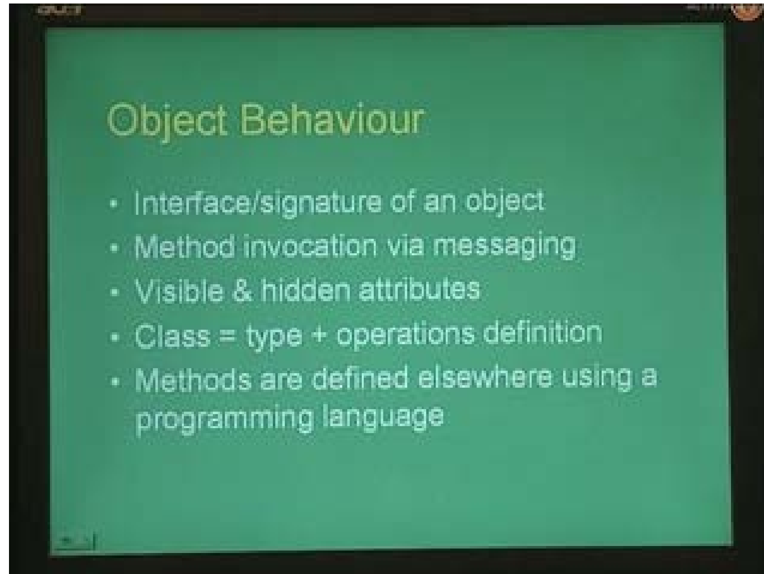
(Refer Slide Time: 34:09)



So here this is some kind of pseudo code for particular kinds of object database systems and later on we will be looking at one particular standard for representing types, namely the OMDG standard. But the idea here is that the user can define his own types. For example the user defines an object of type employee comprising of different attributes that is there is first name, last name, salary, supervisor and so on. And so this forms the tuple of different attributes that represents an object of type employee.

Similarly, there is nested declaration here that is department is a tuple comprising of department name and department number and manager which itself is a tuple comprising of an OID of an object of type employee and start date and so on. So that was, you might have got a question now that what is the difference, what really is the difference between declaring objects or types using what we saw here and with the relational data model itself. That is both seem to be different ways of doing the same thing that is defining a set of attributes.

(Refer Slide Time: 35:37)



However object databases differ from, in one important factor from relational databases namely that of object behavior. So you need not, when you defining a type of an object like say employee and department, it is not just the attributes that you define but also the set of behaviors. So let us look at what is the importance of behavior when it comes to object oriented database systems.

Now object behavior is abstracted by a set of methods and which is visible as the object interface to the external world. Now the interface as I said before is also called the signature of an object that is each object should have a unique interface, each class which uniquely identifies what are the kinds of behavioral abstractions that it provides.

So for example if I have a object of type IC of a particular IC type, let us say some kind of let us say logic gate IC 7404. So this object has particular kinds of behaviors that is you can provide input voltage to a particular pin and you can provide ground to a particular pin. You can provide inputs, logic inputs to particular sets of pins 7404 basically implements AND gates and so you can basically provide logical inputs to certain pins and get logical outputs from certain pins and so on. So probe an input pin or input voltage to a particular pin, all of these are methods that are abstracted by the object.

And of course when we are talking about attributes itself, by default or in pure object orientation, every attribute of an object is actually hidden from the external world. That is the external world can access an object only thorough its interface or only through its method declarations. But in reality though some attributes are visible to the external world, while some attributes are hidden from the external world that is which can be accessed only through method interfaces.

In most object oriented databases, the database management system allows the user to specify the interface of an object along with the attributes like the attributes here (Refer

Slide Time: 38:17) first name, last name, salary and so on. The user can also specify a set of interfaces and the implementation of these methods that is method declarations are provided here and the definition of these methods or the implementation of these methods can actually be returned or can be returned elsewhere using a programming language or method definitions can be done using any standard programming language like C plus plus or java or so on.

So the object database itself does not provide primitives or need not provide primitives to define methods but rather you can actually use an existing object oriented programming language in order to define a method interface. So defining a method interface, now I mean embedding methods in addition to attributes will now enable us to define a class rather than a particular type.

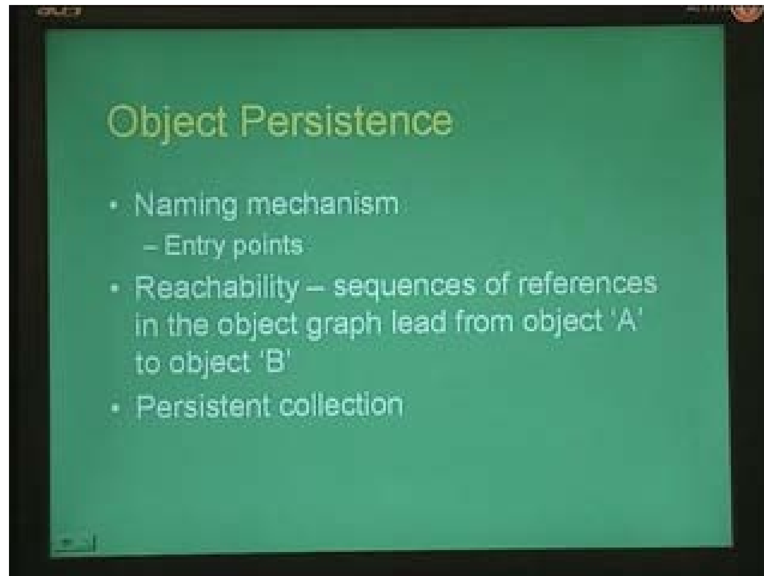
(Refer Slide Time: 39:21)

So one can define a class here for example this slide shows the definition of a class called department where the classes certain attributes like tuple which is a tuple of attributes which basically contains department name, department number, manager which is another tuple in projects and so on. In addition there are certain attributes, certain operations that are also defined like number of employees is, number employees is the name of the method which returns an integer. So when the external world calls this method, an integer is returned which essentially says what is the number of employees in this department.

Similarly create department which is what is also called a constructor method that creates and instantiates an object of type department and put some default values in several, one or more of these attributes and assign employee that is add an employee to the department and so on. So when operations are defined in addition to attributes, we get the definition of a class in contrast to a type. And object persistence, so how are objects themselves persistently stored and referenced uniquely? Of course at the implementation

level, the object database system uses the notion of OIDs that is when we refer to an object and if it is a valid object in the database it is given a unique OID, object id identifier.

(Refer Slide Time: 41:13)



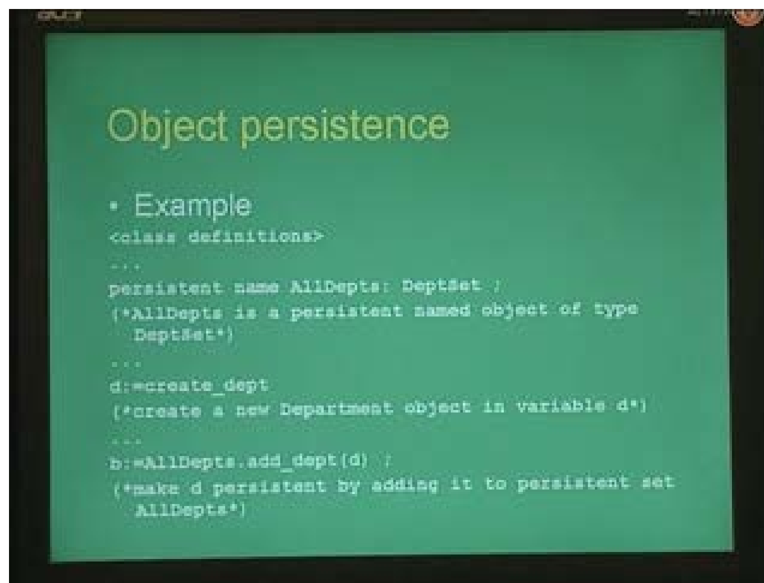
But what is the abstraction? That is the OIDs completely hidden from the user that is the user or application program that is using this let us say the CAD tool does not or need not have to know the OIDs of each object that have been instantiated and stored in the database. Instead the application program refers to each object by different kinds of mechanisms. One well known kind of mechanism is by the use of a naming mechanism that is I can refer to a particular object like say IC 7401 or 7404 number 2 whatever. So each specific object of 7404 that I have created can be given a specific number or this is the first IC or second IC and so on. And like that, using that as a mechanism, the application program can uniquely refer to each object and the each unique name in turn translates internally to each unique OID.

On the other hand there is another kind of mechanism by which objects are referenced in in an object database system by the notion of reachability. That is it may be difficult to give a unique name for every object that is stored in the database system. For example if my let us say I am storing the circuit of a big computer like this, now it has several hundreds of components and this particular circuit is part of larger database of circuits and each different each of this different units or each of the object that are stored in this database has to be given a unique name and which might be impossible I mean it may not be a practical thing to do.

So another way of representing or referencing objects is through the notion of reachability. That is let us say that one particular element in a circuit can be reached only through another particular element. Let us say transistor x can be reached only through the pin number 5 of this IC or whatever. So we don't, in such cases we don't give a

unique name to this transistor and instead we contain with just the name of the IC and any other object that can be uniquely reachable through the IC can be uniquely identified by naming the IC and then following the links. So reachability essentially defines a sequences of references in the object graph that would leave from a well-known or named object A to an unnamed or reachable object B. So this slide shows an example (Refer Slide Time: 44:16) where in some object oriented database systems where some objects can be declared to be persistent that is when we are working with an object oriented database system lets say CAD application, a CAD application is built around an ODBMS. And the way of working with an ODBMS is seamless that is the user would be writing the application program and as part of the application program itself, the user would be interacting with the object database system. So for example there are several, let us say there are several class definitions that make up this application program.

(Refer Slide Time: 44:59)

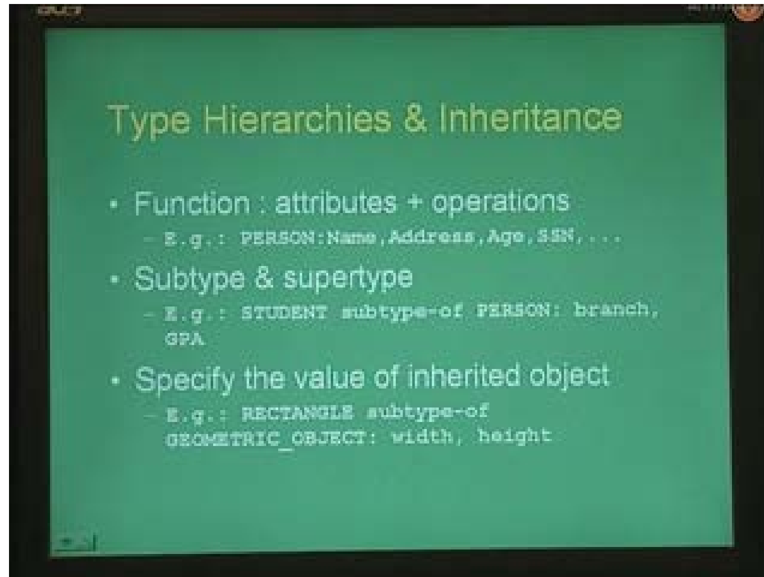


And let us say one of these objects defined by this class should be persistent that is should be persistently stored in the database. So you basically define this objects say all departments which is a persistent named object of type department set which is defined in the class here. So as part of your application definition itself, you define which object should be persistent and which objects can be transient that is they lose their identity or they lose their state when the program finishes execution. And of course this is using the Pascal exsyntax where you can say d equal to create department where create a new department object in this variable called d and then make d persistent by adding it to a persistent set called all department so and then save it to the database system.

So in addition to these different features that are provided by an ODBMS like say type definition, class definition, method definitions and naming conventions and reachability and persistence and so on. There are other kinds of features that are available in an ODBMS of what are called as type hierarchies and inheritances and so on. So the concepts here are more or less analogous to the concepts in OOPLs itself that is whenever

I use a type hierarchy, I am essentially referring to a generalization and specialization relationship.

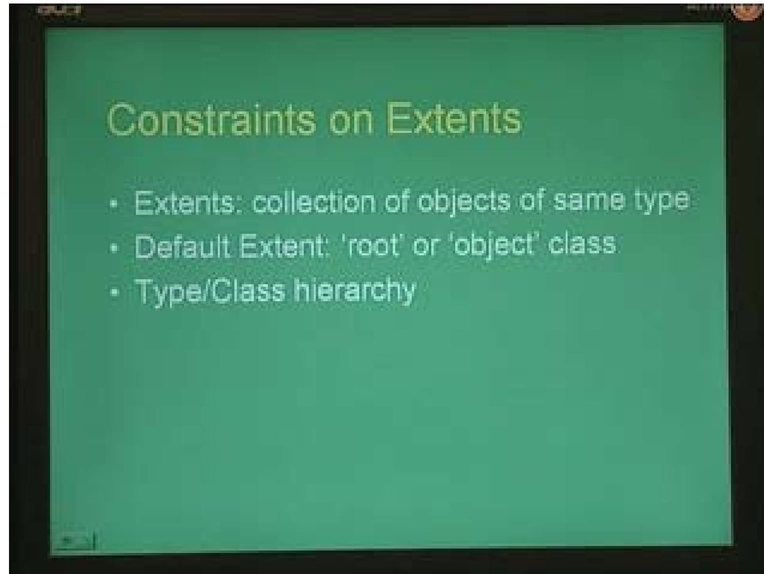
(Refer Slide Time: 46:40)



So a type hierarchy is defined by a subtype and a supertype that is I can define something like a student is a subtype of person and so where I could have defined person as a tuple comprising of name, address, age, social security number and so on. And I can define student as a subtype of person where it contains all attributes that make up a person. In addition there are attributes called branch and GPA which are important for defining a student as well. And similarly I can one can define inheritances that is an object of type rectangle as a subtype of a GEOMETRIC_OBJECT which is not just tuple here but tuple comprising of tuple in addition to certain behaviors.

And then you say a rectangle is defined by width and height in addition to whatever makes up GEOMETRIC_OBJECTS. Then there is the notion of an extent in object oriented database systems. Extents is in some way to give an analogy to ER modeling, in entity relationship modeling we had the idea of entity types and entity sets. An entity type defined a type or a class of entities while an entity set is an entity type coupled with a collection of different instances of this entity type. So the concept that is used here for an entity set is an extent. That is extent is a collection of objects of the same type that is a type definition plus a collection of instances forms an extent.

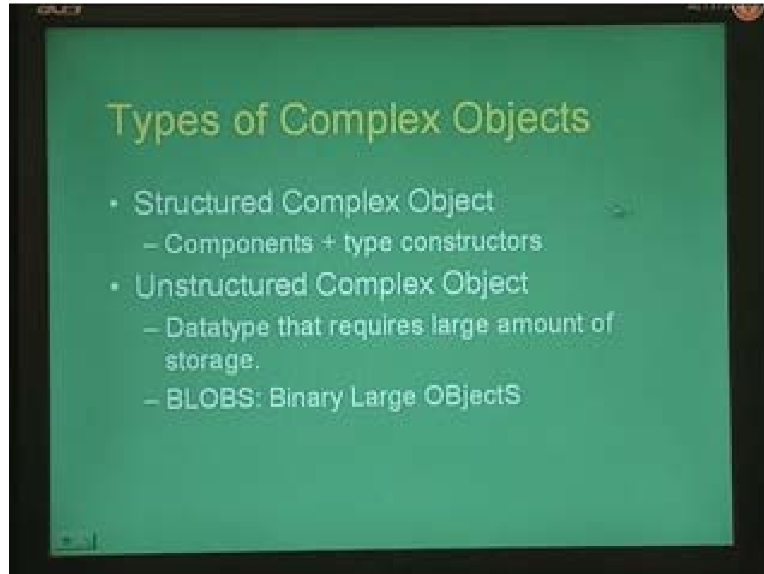
(Refer Slide Time: 48:47)



So the object database system is organized in the form of extents, different extents that is different typed objects are stored in their own extents and then usually because in most object oriented languages there is always a type hierarchy and there's usually a root class or like in java there is what is called as the objects class. There is a default extent that every object belongs to which is the object extent or the root extent. And depending on the class hierarchy or the type hierarchy there can be different sub extents that can be defined on each of these, depending on the class definitions of each of the object that are stored in the database.

And of course the way in which objects are stored can either be structured complex objects, I mean now we are explicitly calling it complex objects that is objects which are not necessarily amenable or data that is not necessarily amenable to storage in a relational database form. So, one can think of structured storage of a complex object or an unstructured storage. Structure storage essentially is some kind of a nested structure, a tuple comprising of other tuples or sets and so on. So, some kind of structuring that is made out of the types that we define or the constructors, type constructors that we define something like atoms and sets and tuples and lists and so on.

(Refer Slide Time: 49:59)

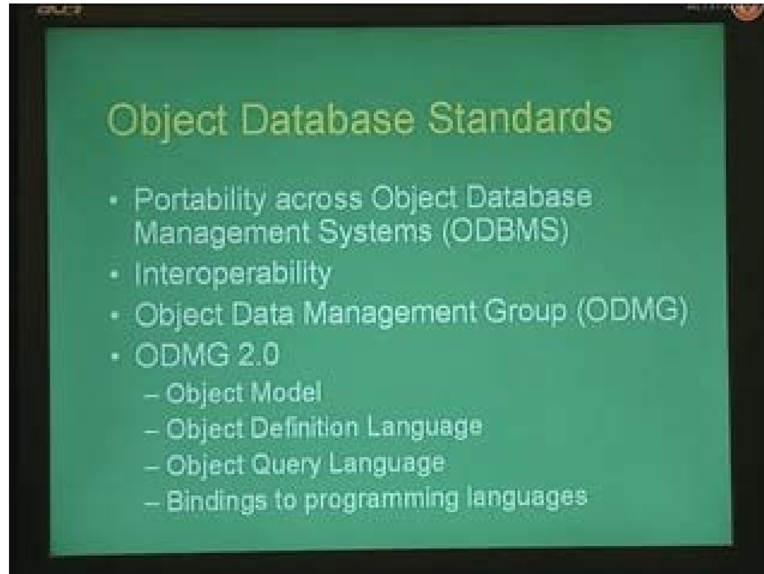


On the other hand there could be unstructured complex objects where especially multimedia objects where I could have a video sequence or an audio sequence and so on where there is no specific structure as such but it's just one heap of data or binary data that makes up this object. And there also called as BLOBS or what expanse to binary large objects. So they are just binary data which are just stored and stored in the database and defined as part of this object.

Now there are several different object database standards that exist and there was several different commercial implementations of object oriented database systems. But many of them have in a sense gone out of business but quite a few of them have still survived and like I mentioned earlier, the main at least as of today the main application area in object oriented database systems is in CAD applications where we need to store objects of a particular, having not only particular properties, a particular structural properties but also behavioral properties. And this behavioral properties or the abstraction of this behavioral properties are extremely important when trying to build let us say an electronic circuit or a mechanical design and so on as part of a CAD application.

Until now we have been mainly talking about object database systems from a pseudo code perspective. That is these are the features that several of these object oriented database systems have or had in a sense but more concretely there have been few standards that define what an object oriented database system should look like.

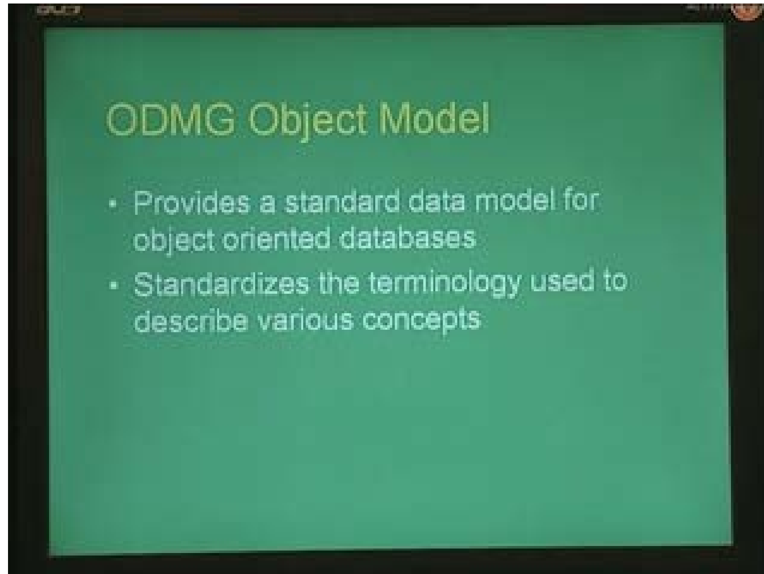
(Refer Slide Time: 51:29)



And among them a well-known standard is the O ODMG standard that is the object data management group standard. And the idea of the standard is to enable certain kinds of features or certain kinds of properties that make up an object oriented database system. So that they can be seamlessly ported across different object database management systems or ODBMS. And the main idea behind the standard is the interoperability between different ODBMS. And ODMG 2.0 defines several different concepts, it defines a basic object model and an object definition and a query language and it also defines different kinds of bindings to programming languages.

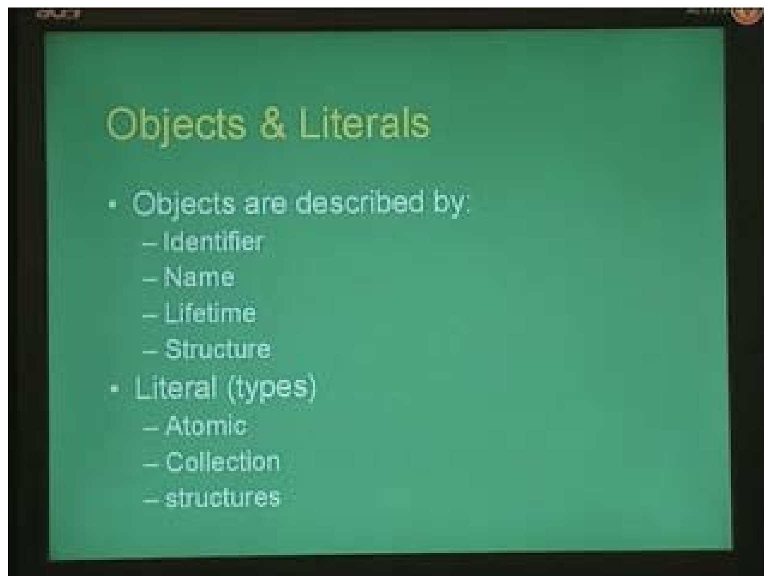
So let us briefly look at what are the main or salient features of the ODMG 2.0 standard and in the interest of time and brevity, we shall not be looking into great, we shall not be looking in great details into the standard but rather look at what are the main features that are provided by the standard.

(Refer Slide Time: 54:18)



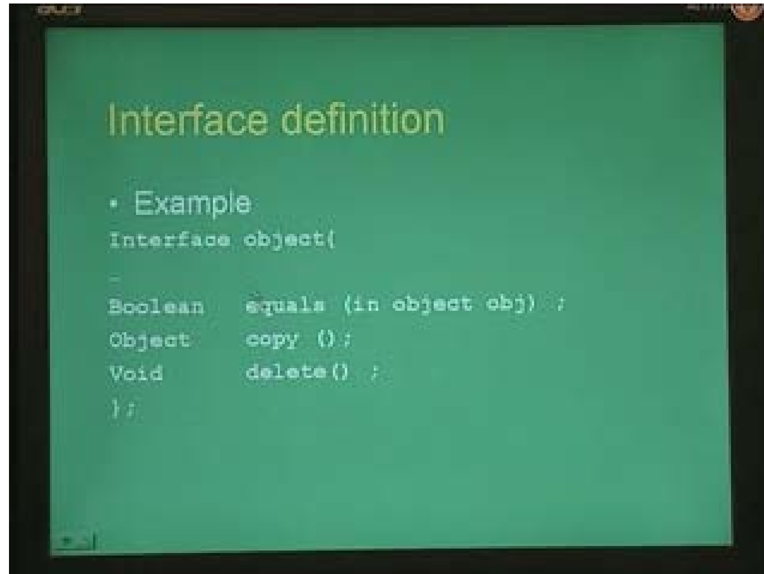
And ODMG standard provides this, basically the idea here is the standardization of terminology.

(Refer Slide Time: 54:26)



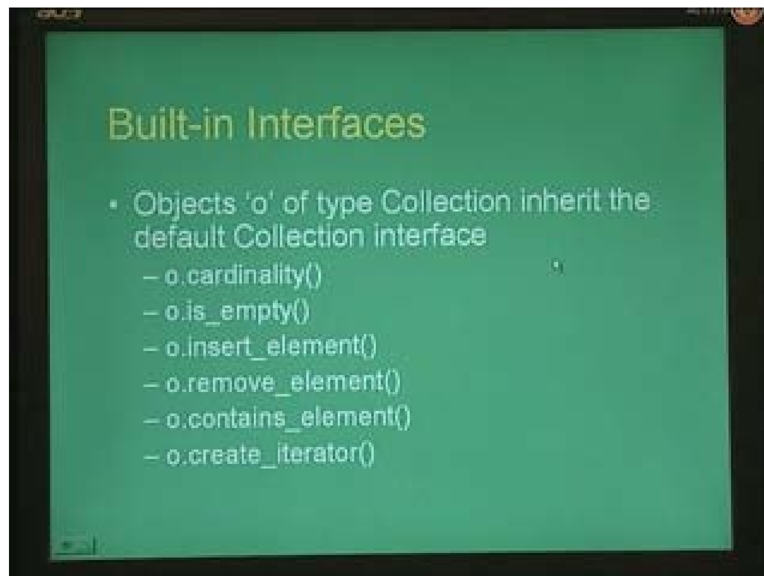
And as shown in this slide, objects in the ODMG standard are defined by these different entities that is name, identifier, life time and so on. And it also defines several types of attributes atomic attribute, collections, structures and so on.

(Refer Slide Time: 54:44)



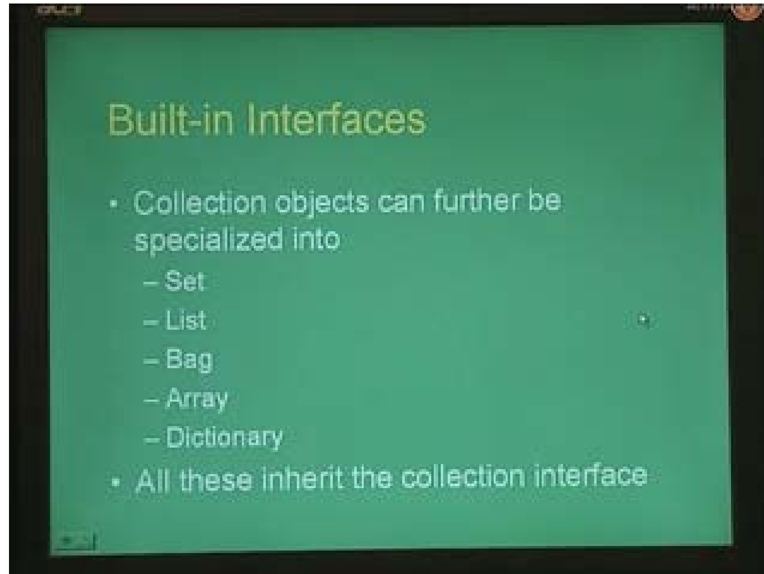
And interface definition is more or less the same that we saw as in the pseudo code and there are several default objects or default classes that are defined by ODMGs standard.

(Refer Slide Time: 55:06)



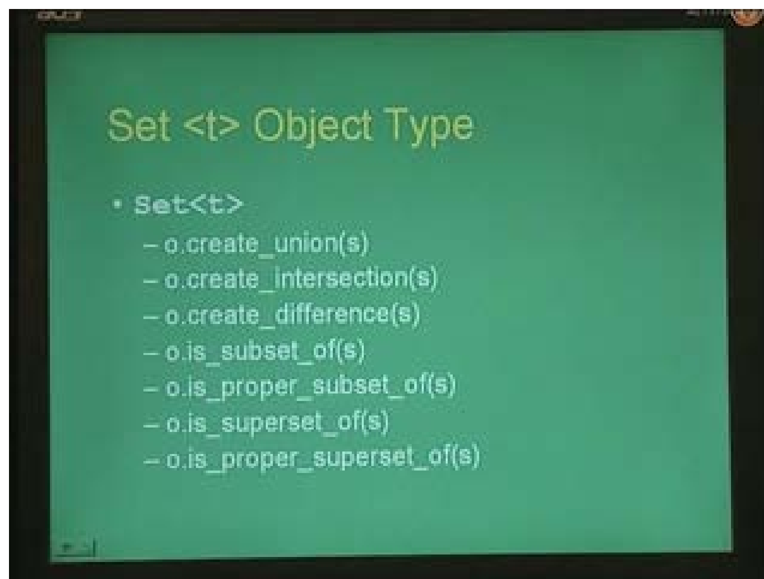
And of these a very interesting default object is the or default class is the collection class which defines a collection of different objects. And this collection class has several methods that are defined like, you can query the cardinality of a collection, you can query whether the collection is empty or you can insert an element into a collection or remove an element from a collection and so on and so forth.

(Refer Slide Time: 55:26)



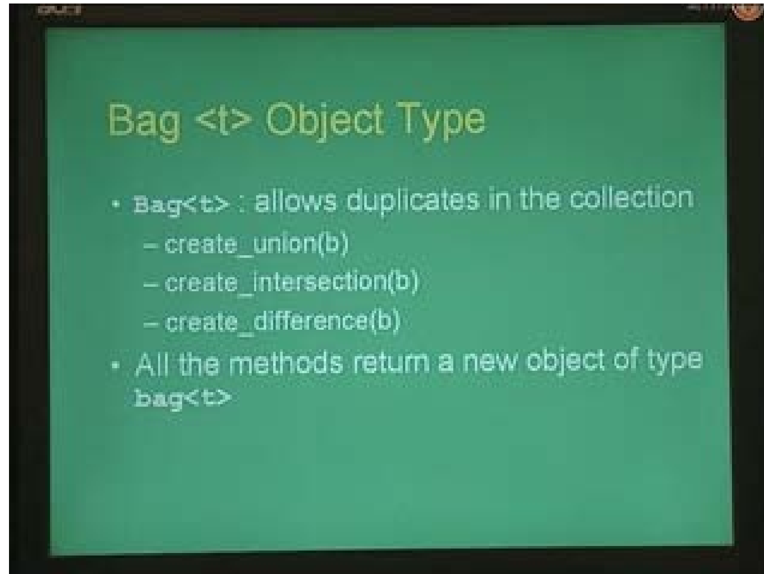
And there is specialized built in, collection objects can further be specialized into different kinds of these, you can specialize a collection as a set or as a list or a bag and so on so. All of these inherit the collection interface.

(Refer Slide Time: 55:41)

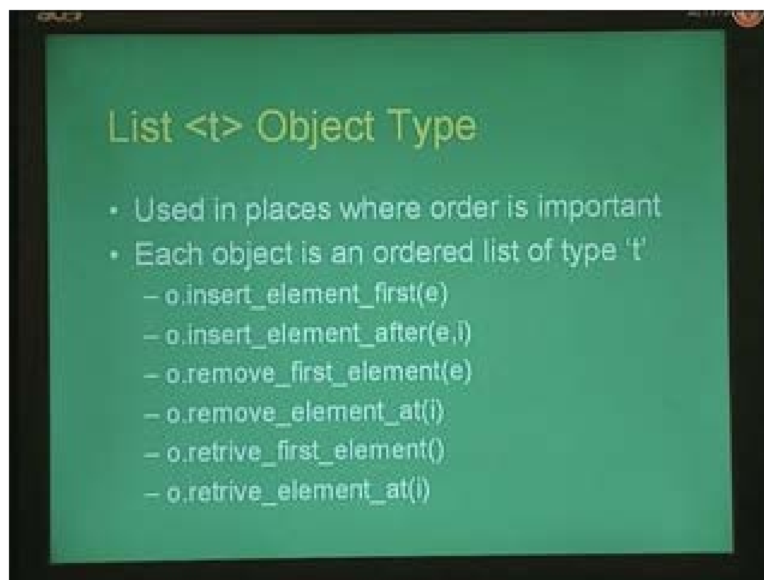


So let us not go into each of these in detail.

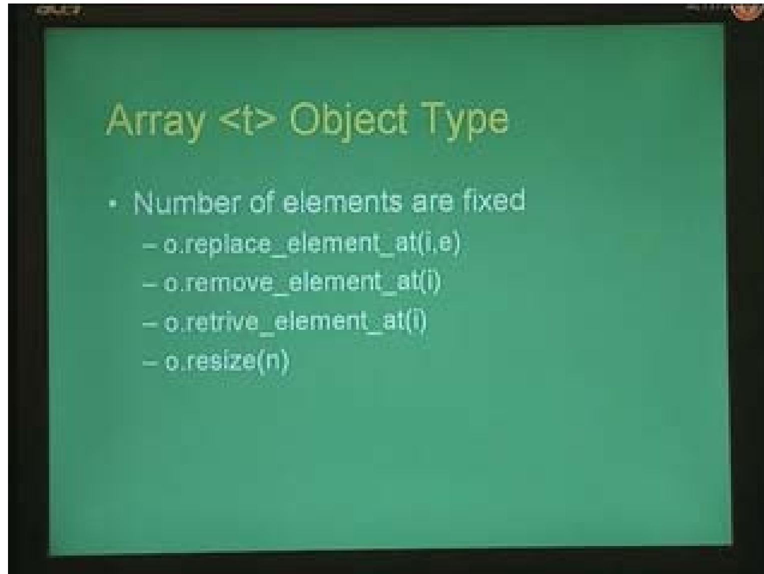
(Refer Slide Time: 55:43)



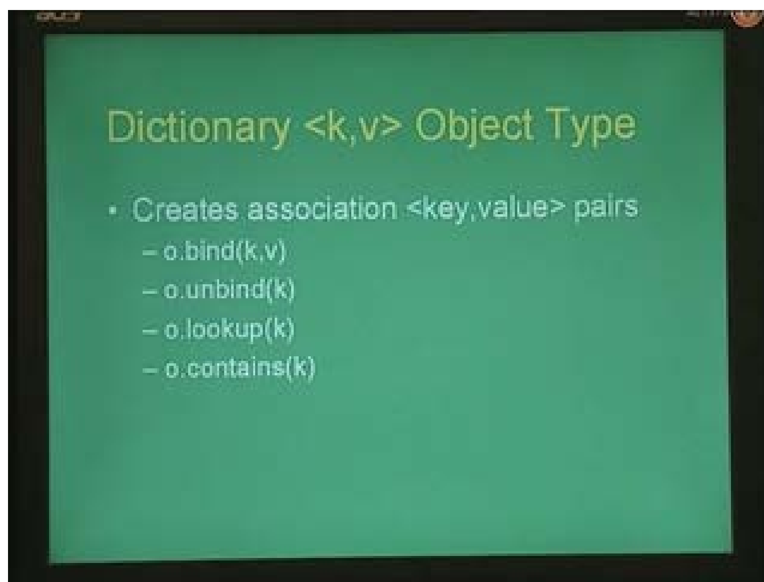
(Refer Slide Time: 55:46)



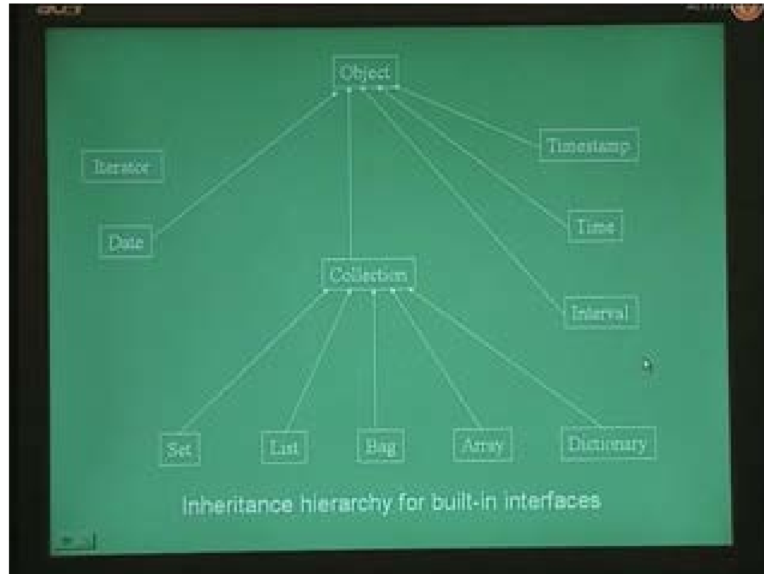
(Refer Slide Time: 55:47)



(Refer Slide Time: 55:48)



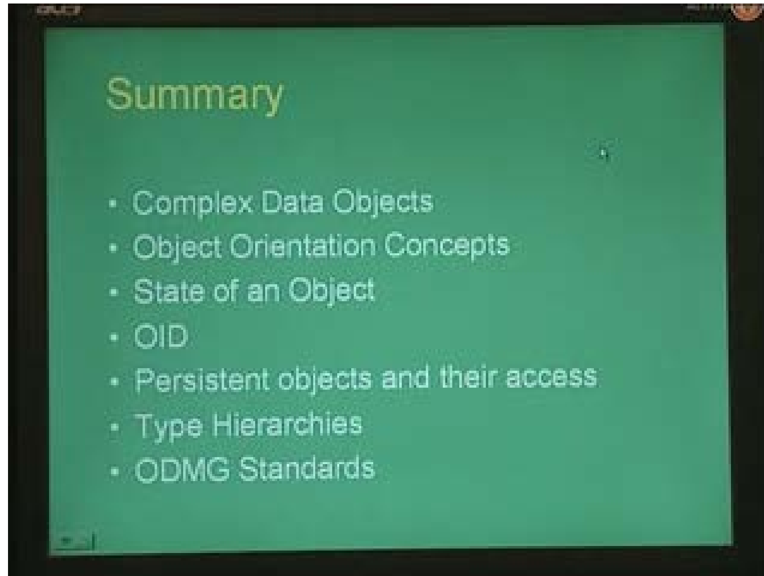
(Refer Slide Time: 55:55)



Let us on the other hand look at some of the type hierarchies, look at the main type hierarchy that is defined by the ODMG standard itself like in java there is a root object in the ODMG standard which is object that is every object belongs to this class called object, root class which is called object. And then collection is a special class of objects which represents a collection of objects and then there are several other kinds of objects like date, timestamp, interval, set, bag, dictionary and so on and so forth. So this is a partial type hierarchy that is defined by the ODMG 2.0 standard.

So let us summarize what we have learnt in this session today. We talked about complex data objects and how they need not be amenable to a relational, reduction to a relational storage. So we looked at object orientation concepts and wherein the notion of a state of an object and the OID of an object become important in order to be able to store objects. Then storage of objects are called persistent objects and how they can be accessed through naming and reachability and so on.

(Refer Slide Time: 56:36)



And then we also looked at the ODMG standard ODMG 2.0 standard which defines its own class hierarchy of different classes. So that brings us to the end of this session. Thank you.