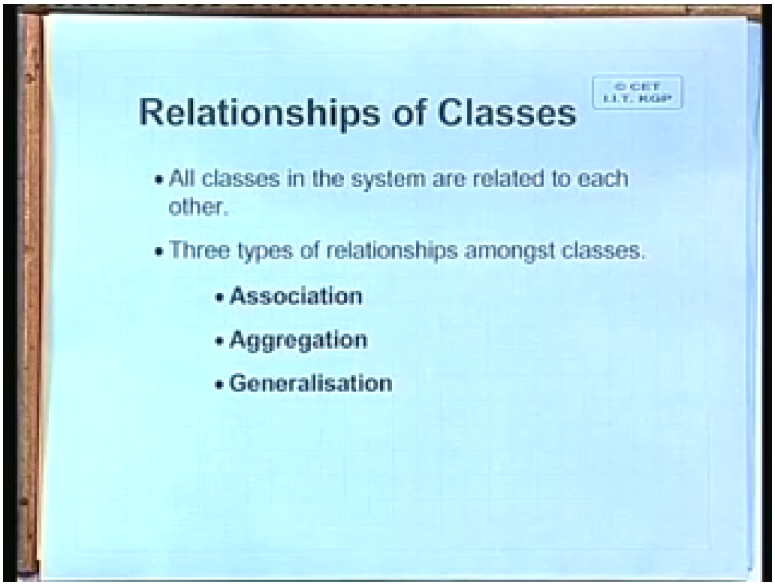
Management Information System
Prof. Biswajit Mahanty
Department of Industrial Engineering & Management
Indian Institute of Technology, Kharagpur

Lecture No. #28
OOAD – II

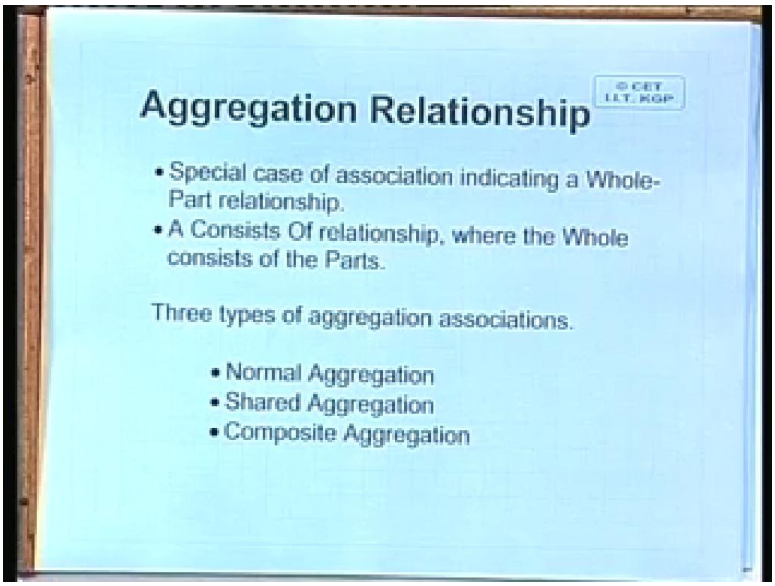
So, we were discussing; we trigger to objective oriented analysis and design the relationship of classes. In particular, we have said that there are three types of relationships amongst classes, particularly the association aggregation and generalization.

(Refer Slide Time: 01:19)



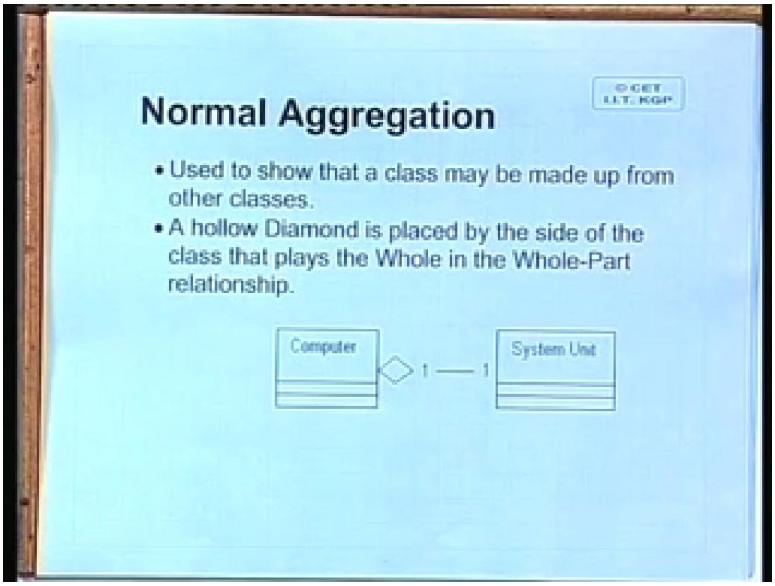
Now, out of these we have discussed the first property that is a association. Now, let us move over and try to see what is shown as the aggregation relationship.

(Refer Slide Time: 01:27)



Now, the aggregation relationship is a special case of association indicating a whole-part relationship. So, a consist of relationship where the whole consists of the parts. So, I have said that there are two basic structures. One is the whole-part and the other one that is the generalization, that is relationship, that is the generalization type. But this one is the aggregation relationship where we have or otherwise called the whole-part relationship. Again, the aggregation associations are basically three types; normal aggregation, shared aggregation, and the composite aggregation. So, the whole-part understanding could be very important when it comes to relationship of classes.

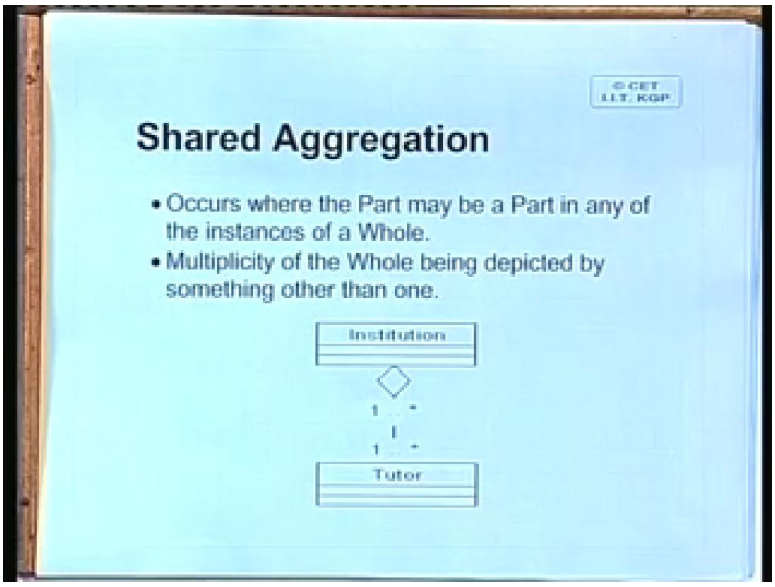
(Refer Slide Time: 02:24)



The first one, that is the normal aggregation. The basic idea is used to show that a class may be made up from other classes. A hollow diamond, this is a hollow diamond symbol is placed by the side of the class that plays the whole in the whole-part relationship. So, here is an example of a computer and it is system units. And usually it is 1 is to 1, a computer has one system unit; a computer is the whole system unit is a part of it. So, it is a kind of whole-part relationship. There is a normal aggregation, the normal aggregation sometimes a simply called aggregation. Most of the time the normal aggregation is called just aggregation.

Can you give another example of normal aggregation? (No Audio Time: 03:29 to 03:35) Another example of a whole-part relationship where we have the part and a whole (( )). No, do not talk about record of a database (( )), because we may not define them as a class. Suppose, you want to have the institute as a class, the institute has got many departments. So, the department could be the part and the institute as a whole. You have to think in that line that both should be ultimately class. So, that could be another example the department is a constituent of the institute. So, department institute both will be part of a normal aggregation.

(Refer Slide Time: 04:35)

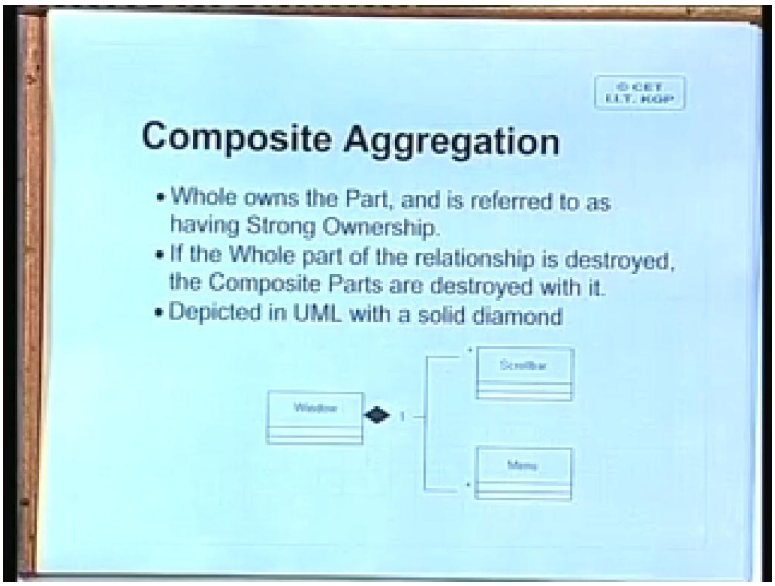


The other type that is known as the shared aggregation is it occurs when the part may be a part in any of the instances of a whole. Something like say, multiplicity of the whole being depicted by something other than 1. See, usually the whole is a something like a

your, you know it is like the department is part of the institute, of the system unit is part of the computer. So, you can think of the multiplicity of the whole is 1, is it not? The ultimately, the whole consists of the parts are basically may be multiple, but the whole should be one, that is the idea. But suppose there are some tutors who are the part of not just one institution, but many institutions.

So, it is like some professional tutors who are actually participating in a number of institutes. So, it is not just an institute and a teacher (( )), but the because the teachers are usually part of a given institute. In this case, the tutor the concept of tutor here is that the tutor will be not only part of one institute, but could be part of another institute as well. So, institution as a class may have a number of objects institute 1, institute 2, institute 3 and a given tutor may be part of not only just institute 1, but also may be institute 2 or institute 3. In that sense it is called a shared aggregation. The difference could be seen in the multiplicity 1 is to star for the institute as well. Is it clear? That is known as the shared aggregation.

(Refer Slide Time: 06:32)



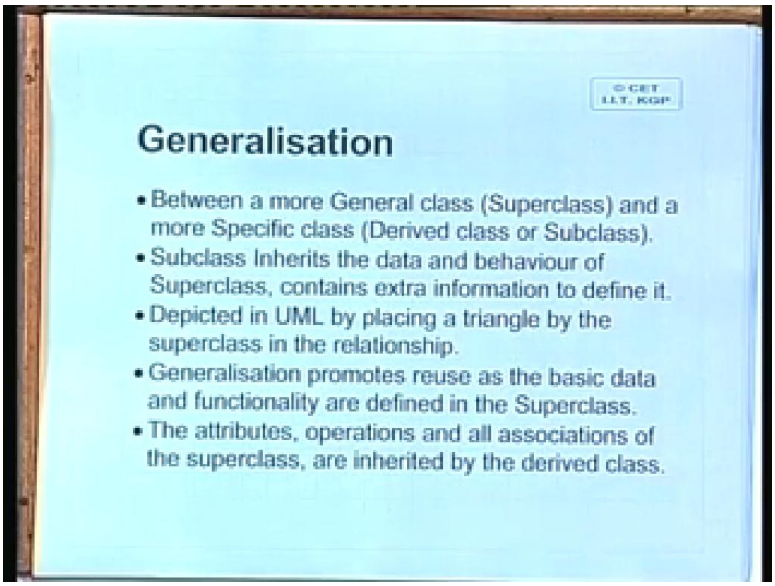
The next one is hold the composite aggregation. The composite aggregation is something like this that you see when we talk about the tutor of the institute or the department and the institute or the system unit and the computer. Even if you the computer classes destroyed, system unit may still be there, is it not? Even if you do not have any record in the institute, you can as well have the department details. The department class may still

exist, but sometimes what happens? The whole and the part has got a very strong relationship. Say this example suppose, you have defined a window, the window has got a menu and a scrollbar.

So, any window that you bring up in computer usually consists of a menu and a scroll bar, by which you can you know scroll from one choice to the other. Now, what happens? If by any chance you delete the window, the window is not there anymore. Then the menu of the scrollbar will also be destroyed along with it. So, relationship is rather strong in the sense that destruction of the whole will also destroyed the menu as well as the scroll bar both. So, it say strong relationship. It say this is strong relationship in that sense. So, whenever you have this kind of strong ownership or relationship you have, what is known as a composite aggregation.

So, whatever it may be the aggregation is a whole part relationship and is an important concept of relationship between classes. Let us move over which is much more important to us and used very much in any object oriented modeling and analysis.

(Refer Slide Time: 08:47)



That is known as generalisation. So, usually whenever we have the we have already same the concept of generalisation in a relationship, while we have discussed the D B M S. Like an employee could be a pilot or employee could be a crew. The employee has got certain details fields, classifications, but pilot will inherit all of them. Not only that it may have certain other special properties. What are those special properties? The pilot

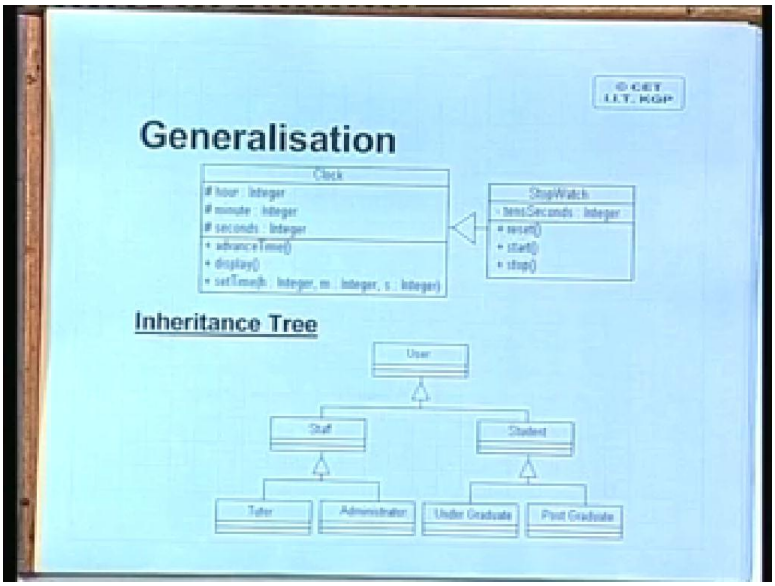
can fly a plane. The crew will be serving people while they are in the plane. So, whereas, another employee may be a clerk, may not be having these properties.

So, a generalisation the basic idea is that you have a class, the superclass. So, called superclass, that is the general. That means, here in the example, I gave the employee is like to superclass and the pilot as well as the crew will be the called derived class or so called subclass. So, the generalisation relationship exist between more general class that is a superclass and a more specific class that is the derived class or so called subclass. The subclass see we are using toward inherit inheritance for the first time what will happen? The subclass will inherit the data and the behavior of the superclass and, but it must contain extra information to define it.

We will give an example to understand what is meant by that. And unlike aggregation here, we are depicting it by placing a triangle by so, that is the diamond symbol and there is a triangle symbol. In this case, we use a triangle symbol not the diamond symbol by placing a triangle by this superclass in the relationship. Generalisation promotes reuse as the basic data and functionality are defined in the superclass. So, what is happening? The major properties of the employee are already given in the employee class. So, when you define pilot you need not keep on giving the same details like employee has got a name, address, salary, show employee, which will show all the details of the employee.

You will simply inherit all those facilities which are already available for the employee. So, you can concentrate only the fly part. So, that is the advantage. Generalisation promotes reuse as the basic data and functionality are defined in the superclass itself. The attributes operations and all associations of the superclass are inherited by the derived class, is it ok? So, all the basically the subclasses own, whatever is there in the superclass as if there is a part of that.

(Refer Slide Time: 12:03)

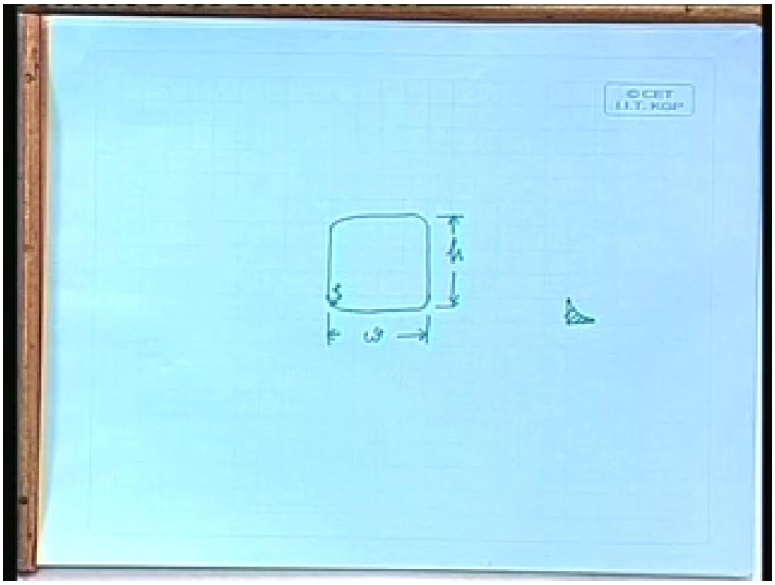


So, here is an example of a generalization. The example basically shows the between the clock and a stopwatch. So, these examples are something like this. That clock has got, let us say three attributes that is the hour, minute, and second. Hour, minute and second three attributes and the clock basically it can had three behavior or basically three what you call the operations defined in the class. Number one advance time you can change the time. Second display you can display the present hour, minute and second in the clock. And you can set time basically to put a particular time at a given point of time. The stopwatch on the other hand is definitely a clock in the sense, we assume here the stopwatch also shows the hour, minute and second at a given point of time.

However, you have three more additional functions that is reset, start and stop. So, these reset, start and stop and we have an additional attribute 10 seconds. That is tenth of a second. So, that means, it how many attributes will stopwatch have? Four. See, it will inherit the that is hour, minute and second. And the fourth one it is own. Similarly, it may have 3 plus 3, six operations define for it. A very peculiar kind of a case could be where you can have the inherited the class can a redefine one of the functions which is in there in the name. It may so happen may be. Suppose, if you define for all basically rectangular type of objects may be a square, a rectangle, a parallelogram those kind of a general structures. The generally, the area is given by height into width. So, if you want to put the area as a superclass function.

So, what will happen? The superclass of the you know the, what you call the shapes, the rectangular type of shapes. We will calculate with the basis of height into width which may be true for inherited class like square, a rectangle, rhombus, parallelogram, each one of them. But assume we are thinking about particular rectangle which is having the edges as chamfered. I have given this example earlier let me just redraw this.

(Refer Slide Time: 15:35)



What I want to say was? See we have a rectangular or a square structure where we have the edges. So, you see not only it has a height and a width, but also it has giving a small radius. So, you see this in a small portion I know this is the radius and this is the thing. So, this much area will be cut off and the area calculation will be slightly more involved. It will be slightly less in fact, the area of this shape will be slightly less. So, basically what you have to do in this situation, you have to redefine the area function once again. Although the area is already defined in the superclass, in the base class or the subclass I am sorry in the subclass or the derived class you have to again redefine area as a function of h w and r. And these will supersede the area which has already being defined in the superclass.

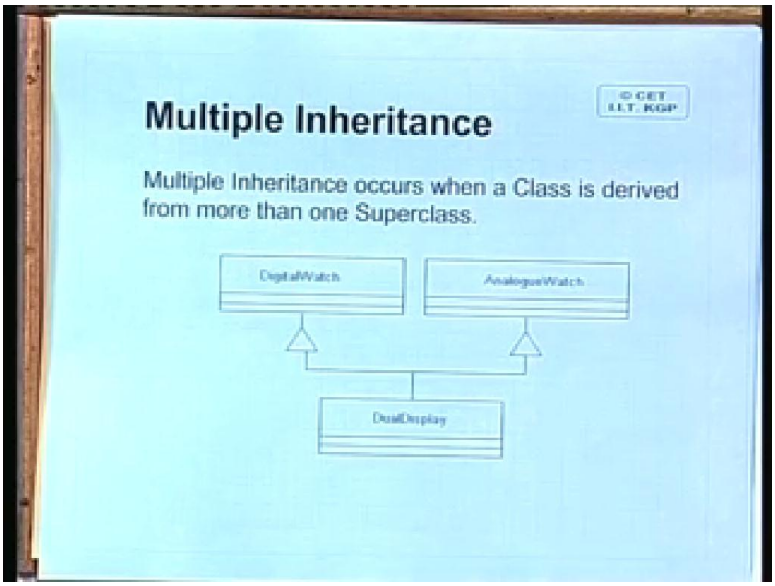
So, this is an advantage in fact, because it may not be all the time possible to have in over the generalization to be perfectly alright. Say otherwise, you have to put the chamfered object to another hierarchy. You cannot put it there, because for most of them the area is having a simple formula. But this one the area formula is slightly revisited.

So, we can do it by a simple calculation. Is it clear? So, this is basically the generalisation therefore, gives you the concept of inheritance.

So, there is further to it what you have? What is known as an inheritance tree. So, you can have a look at an inheritance tree here. See, user could be a staff or a student look at the diamond, the triangle is facing up towards the superclass. The staff could as will be tutor and an administrator. If you there on an administrator, the student could be under graduate, post graduate. So, you can have number of inheritances and then what will happen? The user will have certain properties that will be inherited by staff as well as student. The tutor will inherit the properties of the staff and administrator. Similarly, under graduate and post graduate inherit the properties of the student.

Now, what is then a multiple inheritance? So, subclass in more than. So, now basically, when the inheritance is from more than one superclass then we call it is a multiple inheritance. So, multiple inheritance is the special type of inheritance where the basically, the inheritance comes from more than one superclass.

(Refer Slide Time: 19:09)



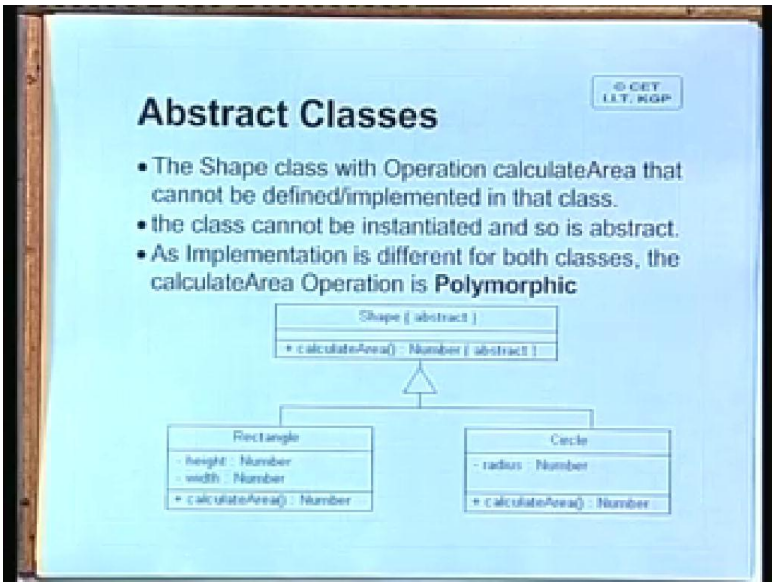
So, one example then you can see here the multiple inheritance occurs when a class is derived from more than one superclass. So, we have the digital watch and the analog watch. Digital watch has got it is own property, shows the time in a digital form. The analog watch on the other hand has hour hands, minute hands like that and dual display. Suppose, if you think of a dual display watch then it will have both digital as well as

analog. So, you can show it as a multiple inheritance. What are the advantages or disadvantages of multiple inheritances? Is it advantageous to have multiple inheritances? reusability increases, but difficulty (( )).

So, there could be ambiguity suppose, the digital watch uses an attribute and analog watch also uses a same attribute. So, which one will it will be inherited by the dual display? You have to specifically look at that. See one big advantage of so called encapsulation, is that once you have defined a class, you need not worry about the how this definition of a particular class comes in the definition of another class. So, you have two classes, the definition of one class should have nothing to do with the definition of another class. So, it is a true encapsulation.

You can independently do it. You can give it to a module developer and let him design that class independently without bothering about another class. But in this case it is not happening because how you define this and how you define this will ultimately come into play while you are defining dual display. Because of these kind of difficulties, multiple inheritance is not preferred or is not available in java language. Whereas, c plus plus defines multiple inheritance. You can have multiple inheritance in so called your c plus plus whereas, not possible in java.

(Refer Slide Time: 21:48)



Now, let us go to another very important concept that is known as the abstract classes. See, here is an example of an abstract class. We have the shape operator this shape class

and there could be two classes rectangle and circle. The rectangle has got height and width and it has a calculate area. And similarly, the circle has a radius and then have it is calculate area. Now, the point is; see the practicality of having a rectangle and a circle define them as separate circles all that thing is fine, but think of suppose, you are basically developing a software for drawing. So, essentially what will you have? You will have basically a drawing, you know icons, different icons are available using those icons you can drop probably rectangle, circle, parallelogram etcetera, different shapes you can basically draw. But how about let us say a simple thing like an area calculation. Will you give a separate area calculation tab or a button for every such shape? Try to see it could be mind-blowing. There could be a 30, 40, 50 kind of shapes and if you have to put an area button. For each one of them it will be difficult. A better idea would be to give a single button, single area button, but to give it you have to have a superclass. But what will be that superclass called and what is it is a basic existence? You see, it may not be possible because then it will be you know it cannot be defined on it is zone for example, shape.

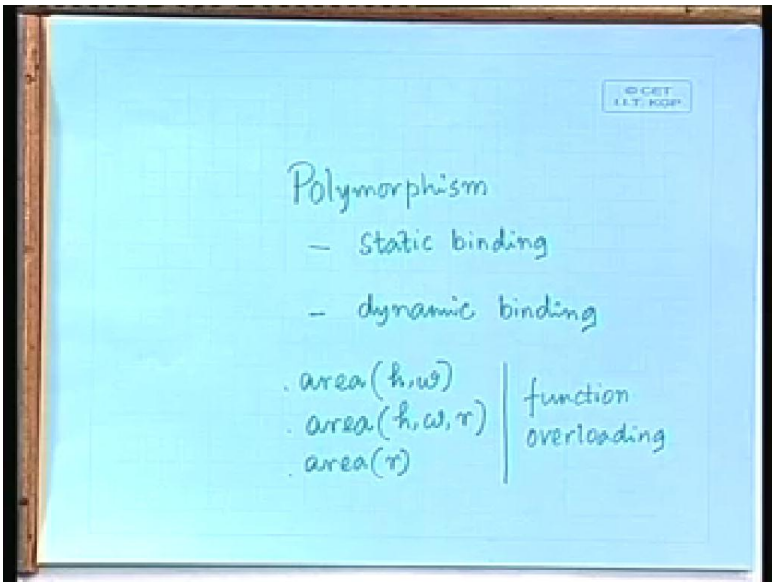
What is a shape? Basically, a shape could be any of these and any of these already defined in the subclasses. In that sense, we have to we can get out of this situation by defining a so called abstract class. But the problem of the abstract class is? A abstract class see the shape class with operation calculate area that cannot be defined or implemented in that class. You if you define calculate area as width into height like I gave, but example like gave there that was for a very specific kind of classes like square, rectangle, rhombus, parallelogram.

I did not include things like circle or other kind of shapes steroids. Those shapes where it is not just width into height. Something even simple like triangles can also have half into width into height is a different. So, the shape it is an abstract class it is defined purely to take advantage of the situation basically, what is known as polymorphism? Basic idea of polymorphism is that the implementation since it is different for both classes, it can only be known at the run time. So, whenever the implementation can possible only at the run time. See, this calculate area button which is the part of the shape, which is the single button.

When you press that calculate area button which calculation or which calculate area function will it use? Will it take for rectangle or circle or triangle or square or what? That

will depend on a given pointer which has been said at that time and pointing to the rectangle circle or the other. So, this is called what is known as dynamic binding. So, essentially the idea of polymorphism is where will have so called static binding and the dynamic binding .

(Refer Slide Time: 26:28)



So, I was explaining to you the dynamic binding. The basic idea of the dynamic binding comes from the concept of abstract classes. The abstract class like shape basically, the calculate area function whenever it is called. Essentially, it determines of which calculate area will it take from the different subclasses or derived classes at the run time. Depending on a pointer arithmetic, but the advantage is when it comes to writing program all you have to do is shape dot calculate area. You see programming is absolutely easy and it is only to be seen at the run time how the calculation could be done.

Whereas, for static binding the particularly when it comes to polymorphism. It is slightly different, it is at the not at the run time, but usually static binding is implemented with the help of what is known as operator overloading. How operator overloading can be implemented? Suppose, we have the area. So, when you give only the h and w, this function could be different from or you see, these three are three examples of operator function overloading basically (( )). See, basic idea is depending on how many parameters you give, a different area function will be called. And suppose, if you simply

give h and w, it will calculate h into w. Suppose you give h w and r, probably the r will be taken as chamfering radius.

And when you give simply r probably it is a circle and it will take a simple circle formula for calculating area.

(( )) will you take (( )).

No, the value you was applying, is it not? When you are calling these.

Because you are calling the first area should not take r at all (( )).

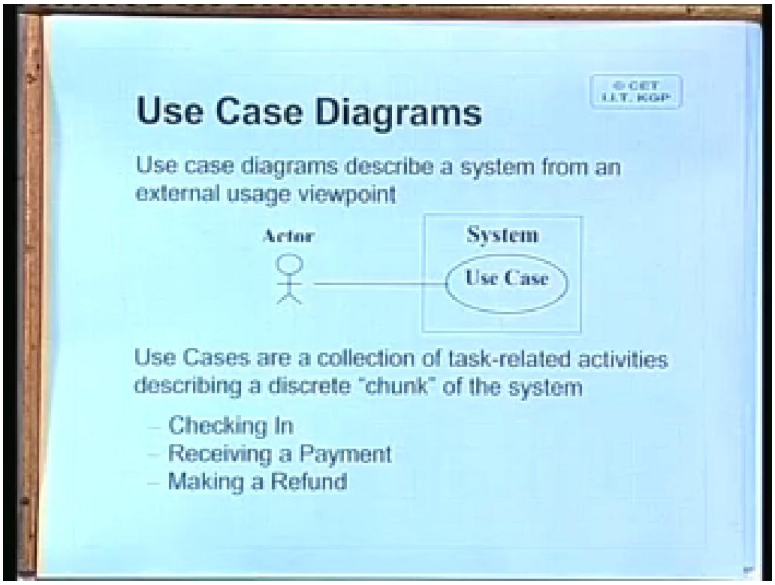
No, it was not take r at all, it is not a question of not take r (( )). You see, these three area although they are called area. But they are actually three different operations and define three, define one after another. See, something like in the same class, the same class has got three different definitions. It is not the same area operation, the same class has got 1, 2, 3 different operations define. Are you getting me? Although they are all called area, but they are not the same functions in that sense. They are three different functions, while your class definition the area operation is defined three times. Depending on how many parameters you are sending either this or this or this will be used. It is not like it is only one function and depending on what you have by logic it is (( )). No, it is not like that, is it? So, that is the idea. So, that is about the classes.

Now, let us so far we are talking more about may be the programming aspects of object oriented analysis. And basically, a getting a broad idea of what could be classes, but essentially three very important concepts we could get one of encapsulation, inheritance and polymorphism. They are very basic object concepts and these concepts cannot be really very well understood, if you take out the programming aspect of the thing. But you must remember our course is mode of analysis and design the other than programming. So, with a broad understanding of what is the object orientation? What is a class? What could be the associations of the class?

Let us go back to the modeling aspect of object oriented analysis and let us start write at the beginning. That means, suppose you have a business situation then how you actually begin? We basically, begin the object oriented analysis by what is known as use case

analysis. So, the use case analysis one of the very primary thing about use case analysis is what is known as the use case diagrams.

(Refer Slide Time: 32:00)



The use case diagrams basically, the describe a system from an external usage view point. See, when you are drawing an use case diagram essentially, what you are doing is how the user looks at it. Do not compare use case diagrams with let us say that dataflow diagrams. The dataflow diagram is again a developer's point of view. It is a developer's point of view. The processes which you are defining in a dataflow diagram in the such a system analysis and design. Essentially, those processes are from a developer's point of view, how he would like do processes to be implemented. Because, we have seen later on the processes themselves becomes later on.

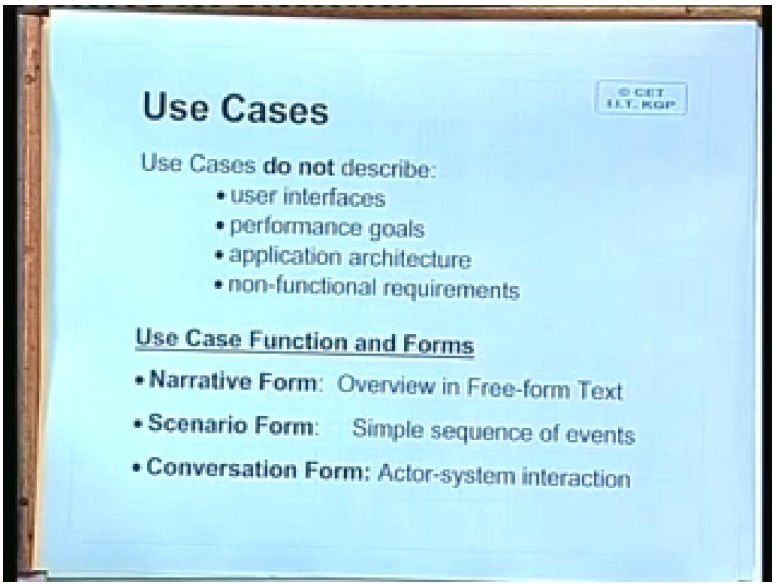
But common in that sense, the use case diagrams are do not help much when it comes to design. A basically one can say, the use case diagrams basically help in putting the requirement analysis or the functional requirements in perspective. So, whenever you are you know having the requirement analysis done and those requirements you want to put it in perspective one of the very basic tool use cases. You see, do not get confuse by use cases and use case diagrams. The use case actually is the elliptical shape here basically, which is a which has to be discussed in further. Suppose, we have a given use case suppose, an actor could be a student and a use case could be register for courses.

A very good example you can find in object oriented analysis and design book. Books for example, by there is a book. In that book he has given in great detail, the post the example of a post. The post is actually point of sale terminal. What is a point of sale terminal is basically a terminal where a sales clerk is sitting and the whenever the customer comes with a list of items then you know the sales clerk will a note the items. And he will give a receipt and he will accept pay that. So, may be if we take that check the items second could be the receipt payment and it possible. Suppose, it is a return kind of a situation, then it could be refund.

There could be more use cases, but to begin with we can think of three use cases for point of sale terminal. One is the check items, the second could be the receipt payment, third could be your refund. How we have defined or the obtained these use cases? Basically, from a clear thinking that not from the developer's or you know programmer's point of view, but purely from an analyst's point of view. What the user actually doing? What are the broad categories of work that the user or the clerk in that case is actually be doing is the point of sale terminal.

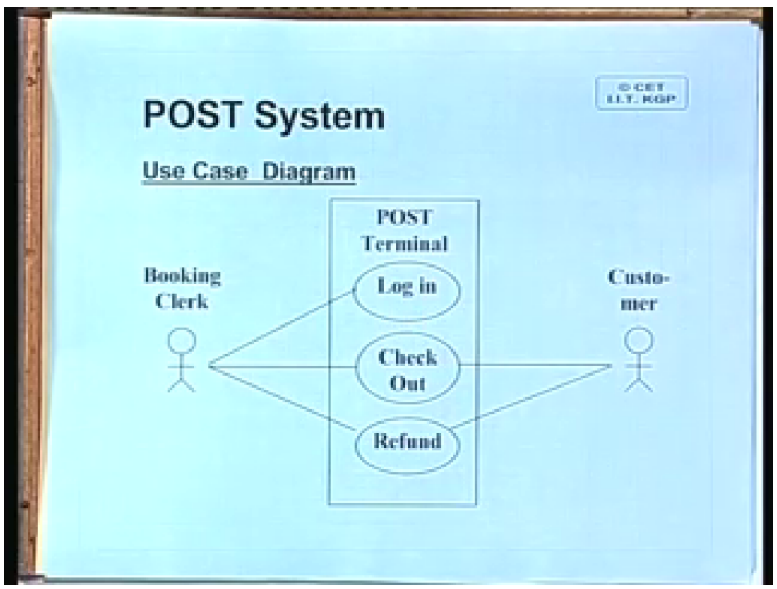
But sometimes we can gets and we may say that switching on the terminal. Then you know then opening the screen, then entering the password, bringing the system up we like the we can list so many use cases. But that is not right. We should try to look at it from your user activity point of view. What is user actually by all these processes. What the user is doing is actually logging in. So, we may say log in, log in could be a use case is it alright? So, that is why you must; you see use cases are a collection of task-related activities describing a discrete chunk of the system. It should be a discrete chunk of the system and not just one small thing in a sense that should be a business activity. So, one use case must represent the business activity.

(Refer Slide Time: 37:07)



Now, before we move further we must also see what use cases do not describe? Before, we see what use cases actually describe let us understand what the use cases do not describe. Do not describe the user interfaces, the performance goals the application architecture or the non-functional requirements. So, use cases do not describe user interfaces, performance goals, application architecture, non-functional requirements. So, only the functional requirements it basically shows. It does not shows the implementation. Implementation is not shown. So, I think let me show a diagram before we move for that.

(Refer Slide Time: 38:12)



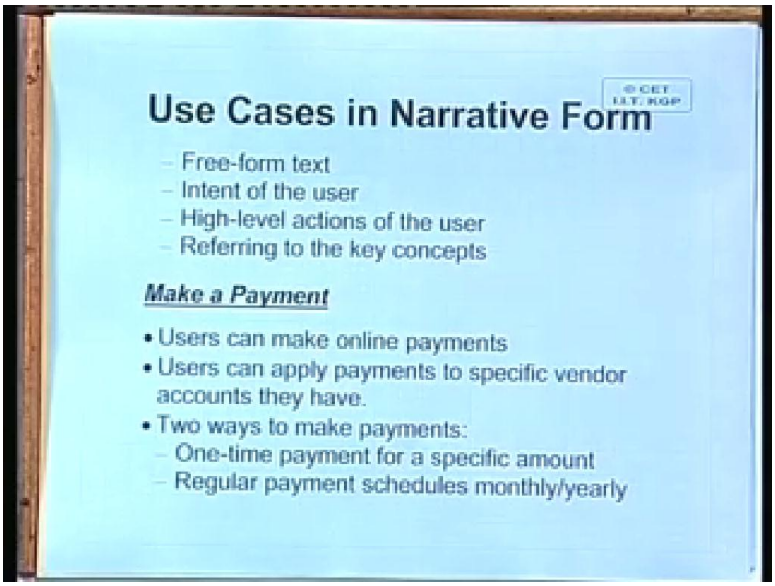
You see, this is what I was talking about. About a post system, the point of sale terminal an use case diagram will have the booking clerk. The booking clerk and on the other hand will have the customer which is shown by the stickman symbol. The stickman symbol for booking clerk and the customer, these are called actors. The actors are basically the people who are involved in this. They are not entities like we have define external entities earlier, but they are actually who are doing the job. And whenever there is a connection for example, when you are logging in, the customer has nothing to do with it.

So, it is only the booking clerk who is involved in logging in. The check out on the other hand you know is something for both booking clerk and the customers both are involved. Whereas, the refund we have again booking clerk and customer both involved. So, this is a very simple example of an use case diagram and one can see that we are identified. See broad use cases and out of these three broad use cases, we are put it inside a rectangle. This is basically the system and the system in which they use cases are implemented, that is the post terminal point of sale terminal. So, we will come back to these examples later.

Let us try to understand further, what the use cases are? Basically, you see it does not end here simply drawing an use case diagram. Basically, what the diagram is going to say? A diagram will only say who are the actors and what is the use case, that is all. So, that is the only thing that it tells, but it does not end there. What you must add in a any use case is description of the use case. It must be described in detail. So, what kind of descriptions write the description in three forms, the first one is known as the narrative form where you give the overview in free-form text. Or you can write it in a scenario form where you write the simple sequence of events. Or you can write the conversation form by actor-system interaction.

So, you have the actor on one side and system functions are other side, and what the actor will do? What the system will do? I will give you some one or two examples for the first two.

(Refer Slide Time: 41:03)



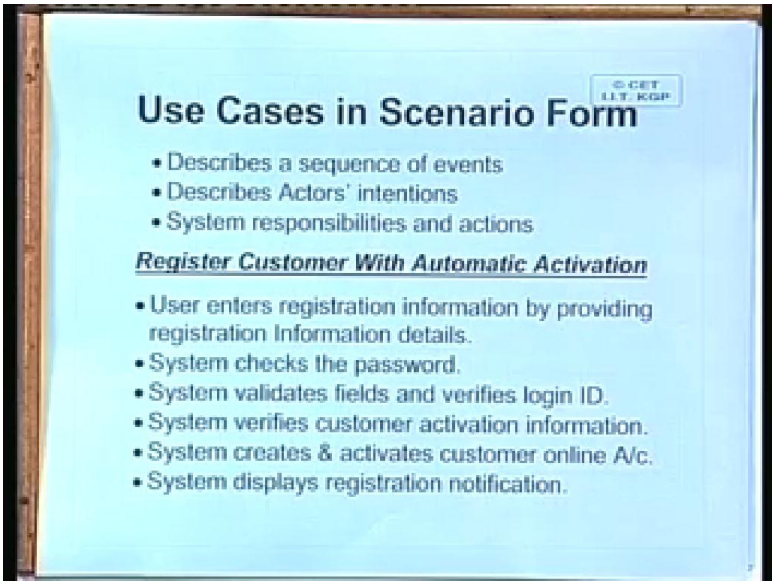
So, let us see what is happening in the so, called narrative form. In a narrative form we have what is known as a free-form text. It shows the high-level actions of the user is intent and while a referring to the key concepts. So, an example supposes we have an use case make a payment. So, what are the broad activities that is going on? Users can make online payments. So, users can apply payments to specific vendor accounts they have. And two ways to make this payments, one-time payment for a specific amount or regular payment schedules monthly or yearly. So, this is a may be called some kind of an use case description.

So, you can refer to may be the u m l user guide I think bihar the text book to understand object oriented analysis or book is not bad you can also have a look at book. The post case study has been given in great detail in book. There is another book by (( )). So, you can also see that and in all these books you can see that apart from the use case diagrams, you can also see how to write the use case descriptions. So, this is one example of an use case description in a narrative form. Now, let us move over and see how the next one? That is how use case could be written in the scenario form. So, what basically and why do I write use case description? What is the use of these?

You see the use case description is like stating the requirements. You have done the requirement analysis and whatever way that requirement analysis you are coding it in a nice form. That is the basic purpose the use cases (( )). Let us move over to what is

known as the scenario form. The scenario form basically a describes a sequence of events sequence of events.

(Refer Slide Time: 43:35)



Describes the a intention of the actor into a actor's intentions and the system responsibilities and action. So, what the system is supposed to do? What is the actor intends to do? And what are the sequences of events. Let us look an example, register customer with automatic activation. So, what is the being done? First, user enters the registration information by providing registration information details. So, may be whenever you want to register as a customer say, to some business say for example, you want to be a customer to a consulting organization. So, you want to register.

So, naturally you have to first keep a set of information. Set of information may be your name, address in what purpose you want to have, this particular facility may be a confirmatory password. So, all these information you have to enter. Then system would check the password and system would validate the fields. Particularly, and verifies the login id and the password. So, system you see the system also have got certain things to do, then system will verify the customer activation, information whether the information is true.

So, may be some cross referencing thing will be given and from the cross referencing you can do this. Then system creates and activates customer online account and the system displays the registration notification; that means, such an such customer has been

registered. So, basically you see what are these suppose; you have simply drawn a register customer with automatic activation, this is like an use case. What are the, who are the actors? Yes, customer and the system. You see the actor need not be a person all the time. The actor can as will be a interface or a system (( )). See, any entity which has something to do with it, our next slide is an actor's we will discuss that.

So, that is the simple diagram that you might have drawn. You have drawn an use case register customer with automatic activation, one actor as customer, another actor as system. And they have both to do something with this use case. That is our very simple use case diagram (( )). See, no see users are the so called customer. Customer is entering the all these information. So, it is alright. Where system is also doing things, the system is checking the password, system is validating the fields, verifying password. So, system has got lot of actions to do.

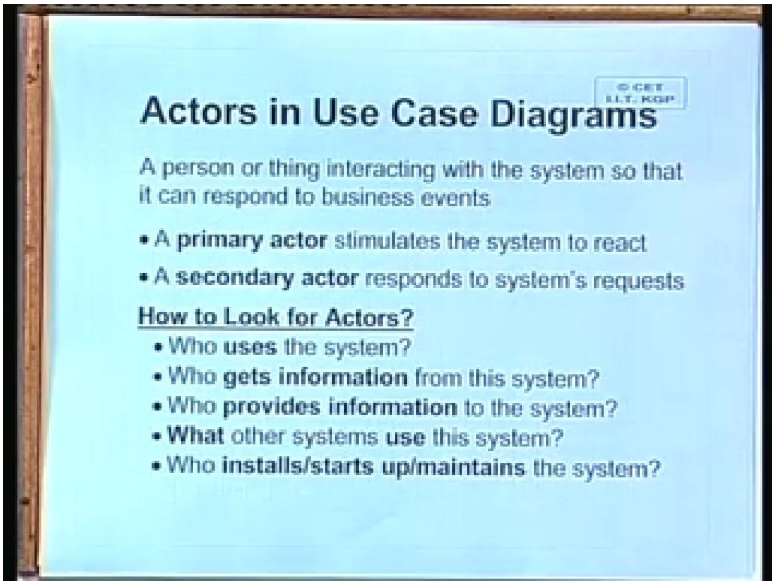
You see please understand this, do not look at it I say that in the beginning, do not look at it from a programmer's point of view, from a developer's point of view. Look at it from a functional point of view, from a the user's point of view. The use case diagramming method is a purely user orientation. Any requirement analysis diagram or requirement analysis document when you prepare (( )). It is a pure block box that is why the requirement analysis is called the block box view of the system. It does not talk about implementation, it does not answer any why question, it only answers the what questions. (( )).

Post terminal was not the actor post terminal was no, post terminal is the terminal where the clerk is working is the system. But if you think that login requires also the system intervention then you can call system also you could have put as an actor. For example, whenever you are let us a logging in, there has to be a password authentication. So, you may like to put system or the also as an use case a not as use case is an actor. We will give more examples I think at that time, system ok working is working with the data, but look at it how the user would look at it you see you have identified certain requirements.

What are the requirements? This is what is here, some of the requirements that we have identified like checking the passwords, like validating the field verifying login id. So, look at it from user's point of view, who is doing it? some system is doing it. So, you put it as an actor that is the idea. I think if I give more examples you ask this question once

again after we discuss more about the use cases may be when we give one or two examples maybe we can come back to that.

(Refer Slide Time: 49:12)



Then about the actors. So, actor a person or a thing interacting with the system. So, that it can respond to the business events. In fact, we can also answer to that question in this way that how you look at it a look at a business situation may be differing from one person to another. So, one modular who has defines something as an actor, may be slightly different to another. It may not be all the time exactly matching so little bit of subjectivity could be involved there. So, you can have a primary actor who stimulates the system to react and a secondary actor would responds to this so called system's requests. So, we can have what is known as a primary actor or a secondary actor. So, how to look for actors? We can have an actor is who uses the system, who gets inform.

Now, these answers may be some of your question who uses the system, who gets information from this system, who provides information to the system. Then, this who part is over. What other systems use this system? I will give one example, who installs starts up or maintains the system. So, all these will be part of what is known as actors. See, sometimes what happens let me give an example (( )). Suppose, you have then old system running and old payroll system running in your organization. And now, you want to develop an accounting system which was previously not there financial accounting system out of which the payroll is a part.

Now, you have started developing now, everybody knows the financial accounting and payroll are related to one another in a big way. So, whatever you do in a financial accounting package, the payroll will have lot of interfaces. Now, what will you do? The old payroll you cannot discontinue suddenly. So, you have to you are redeveloping the payroll system may be, but you do you cannot put the new payroll system in use immediately, because old payroll system is still running. So, you may call the old payroll system as the legacy system. As a legacy system you allow the new system, new payroll system will not be activated.

The legacy system will be still running; the other financial accounting systems will be operated. And whenever there will be any payroll kind of a thing then the legacy, you have to interact with the legacy system. In the process of time, when your new payroll is fully developed and you have full belief on the new payroll system, you can discontinue the old and continue the new one. Till such time the old payroll system or the legacy system should be shown as an actor because, you are interacting with it. So, this is what it is, that who uses the system, who gets information from this system, who provides information to the system and what other systems use this system, and who installs starts up or maintains the system?

So, we leave it here today in our next class, we will come to more about the modeling with use case, I will show some examples of use case. And then move over to more important things like conceptual diagram, sequence diagram, you know and collaboration diagram, class diagram and so on. So, thank you very much.