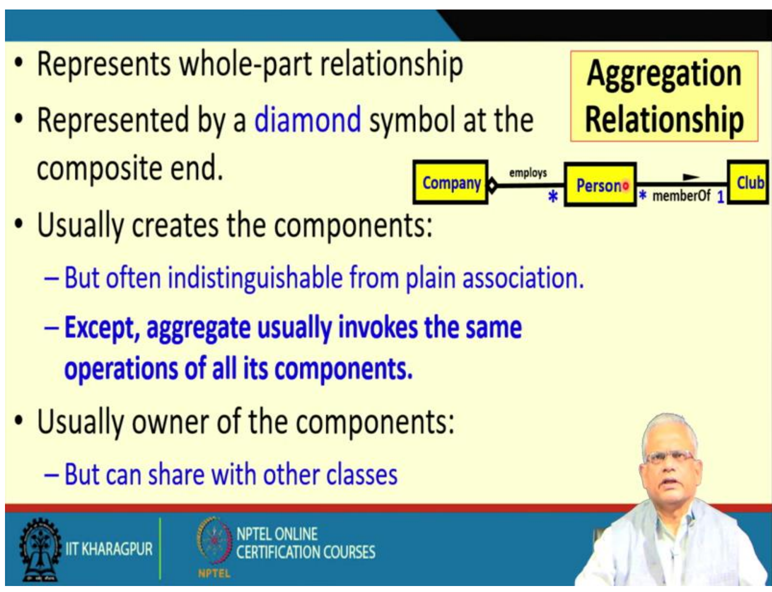
Object – Oriented System Development Using UML, Java and Patterns
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 14
Aggregation and Composition

Welcome to this lecture.

Over the last few lectures, we have been looking at UML and then we looked at use-case modeling, we looked at the class modeling and then also we looked at class relations. In class relations, first we looked at inheritance relationship, which is the easiest to model, as well as to implement in Java. We said we used the 'extends' keyword to implement inheritance relationship. But unfortunately, the other class relations are not that much straightforward to implement.

We looked at Association Relationship and we looked at some aspects of Association Relationship, such as association class, qualified association, association multiplicities and we also looked at implementation of association in Java. And after that, we have just started discussing about the Aggregation Relationship between classes and how to implement in Java. Let's proceed from that point.

[Refer Slide Time: 1:32]



The Aggregation Relationship between classes is modelled with a diamond symbol. And we said that the Aggregation Relation represents the whole-part relationship. A class is a whole class, contains several parts and it models the whole-part relationship. And we use a diamond

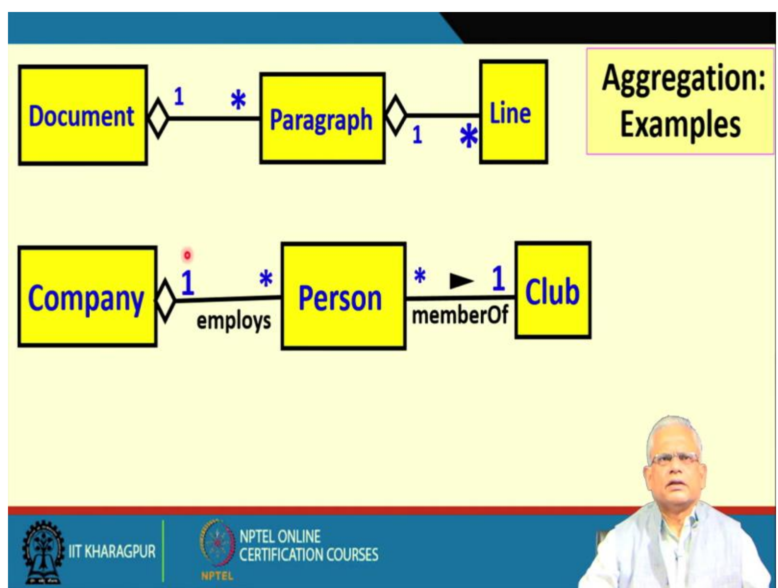
symbol to model the aggregation class relation. The company employs many persons (shown in the above slide). So, here, the company is an aggregate of persons and many persons are members of one club.

You might have a query that why not have the Aggregation Relationship on the club end also, why just it's an association? Company employs many persons, here we have the Aggregation Relationship. But on the other hand, we have an Association Relationship between the club and the person. Even though a club has many members. We will see the subtle difference between the Association Relation and the Aggregation Relation.

The Aggregation Relation is an Association Relation, but has some extra requirements. The requirement is that, the aggregate class not only keeps track of the Ids or the references to the objects here but also, it creates these objects. Like, the company keeps track of many person object references. So, the new is called for the person by the company methods inside the company class. In addition to having the association relation, Company has Ids for the person. But also, it additionally calls the new constructor for the persons and responsible for creation of the persons. On the other hand, the club has Ids of the persons, that is the person object references which is maintained by the club, but it is not responsible for creating the person.

Another difference between an association and aggregation can be that, typically the aggregate object invokes the same method across all the objects it aggregates. For example, print person details. Here it invokes all the person objects to print their details. On the other hand, without aggregation, it possibly calls them selectively. But off-course, it can also call all the object methods. We say that, the aggregate is the owner of the components (the objects that it aggregates). The reason we call it as owner is that, it creates those. In a plain association, it's not required to create, it just maintains the reference Ids. In aggregation, it creates and therefore, we say that it is the owner of the components or the objects. But in association they also maintain Ids to these objects, but they do not create it.

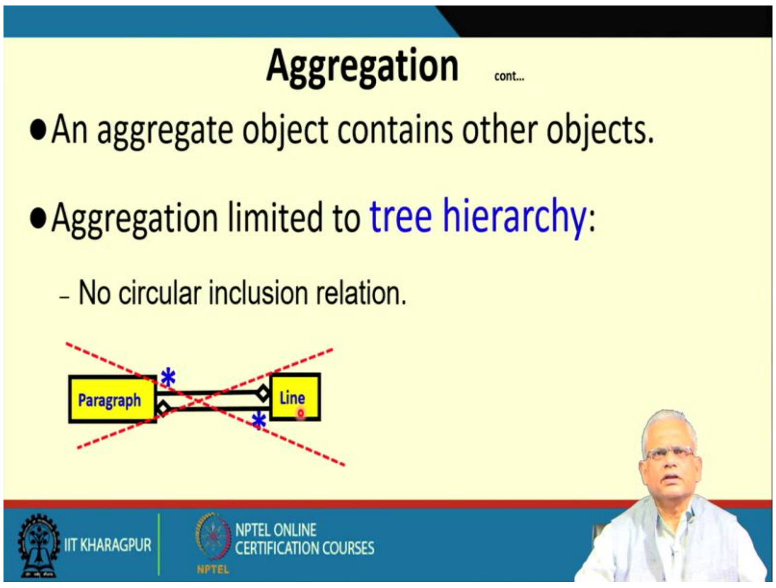
[Refer Slide Time: 6:25]



Now, let us look at more examples of association (in the above slide).

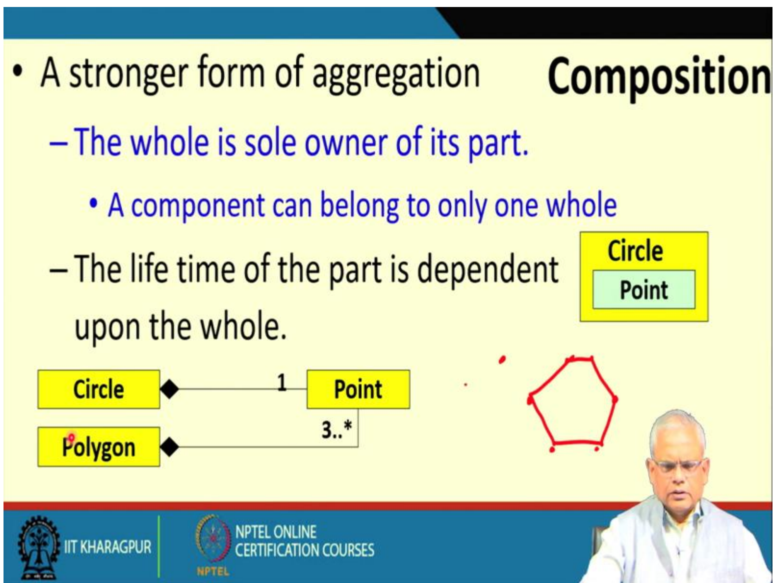
The first example we can see that a document has many paragraphs or a document contains many paragraphs and a paragraph has many lines. And another example, a company employs many persons and many persons are associate with one club. We can write the specific name of the type of the aggregation. In our second example, 'employs' is the aggregation name. In some example, we can prefer to leave it out. In our first example we have not mentioned any aggregation name. In this case, we just read Document contains many paragraphs or a document has many paragraphs. And the multiplicity here we implied is '1'. If we leave it out means if we have not mentioned the multiplicity then also, it means '1'. But then, we can also prefer or wish to write explicitly. But even if we omit it, it is implied '1'. If the multiplicity is anything other than 1, like 2, 3 et cetera, then we need to write it explicitly.

[Refer Slide Time: 8:02]



We say that an aggregate object contains other objects. But then, we must be careful, that the aggregation relation is limited to a tree hierarchy. That is circular inclusion relationship should not be created (as shown in above slide). For example, it is wrong to have this kind of relation, where a line contains many paragraphs and a paragraph contains many lines. This is inconsistent modeling. We should not do this.

[Refer Slide Time: 8:51]



Another special type of Aggregation Relationship is the composition. In composition, we use a filled diamond. In the above slide, one composition example is shown. We read here, that a circle contains one point and the polygon contains three or more points. Alternate representation for composition also shown in above slide: we write the circle class and inside

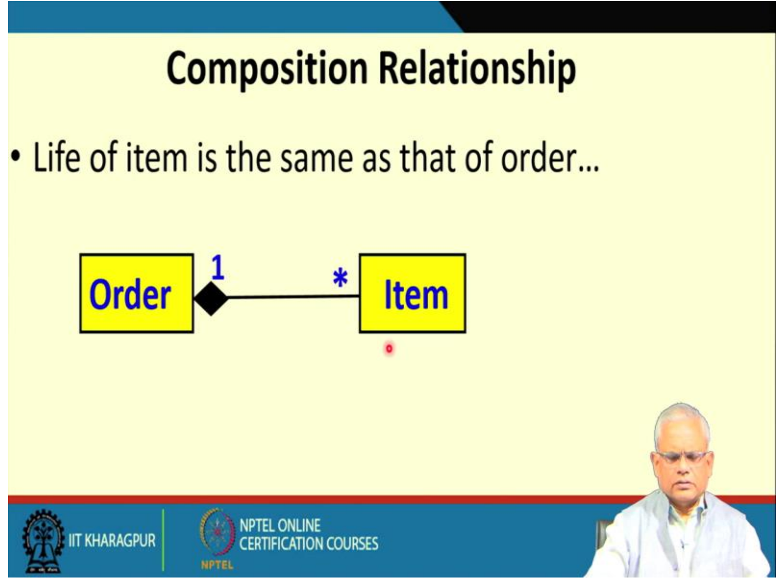
that, we write the point. Or we say that, a circle contains a single point. That is how we represent a composition. Composition is an aggregate relation, but then it is a stronger form of aggregate relation. Here, the aggregate is the whole owner of the component. And a component belongs to only one of aggregate. It cannot belong to the other aggregate. For example, if a point belongs to circle it will not belong to polygon.

There are some other differences between an Aggregation Relationship and a Composite Relationship. The most important is that, in composition the life time of the part is dependent upon the whole. So, this is possibly the most distinguishing relation between Aggregate and Composite relation.

In the Composite relation, once the composite is created then component also created. For example, the circle is created, the points that define it are also created. And once the circle is destroyed, the point which is owned, is also destroyed. So, once the composite is created, the component is also created. And when the composite is destroyed, the component is also destroyed.

Similarly, with a polygon, when the polygon is created, the points that define the polygon are also created. These are the components of the polygon. And when the polygon is destroyed, the components are also destroyed.

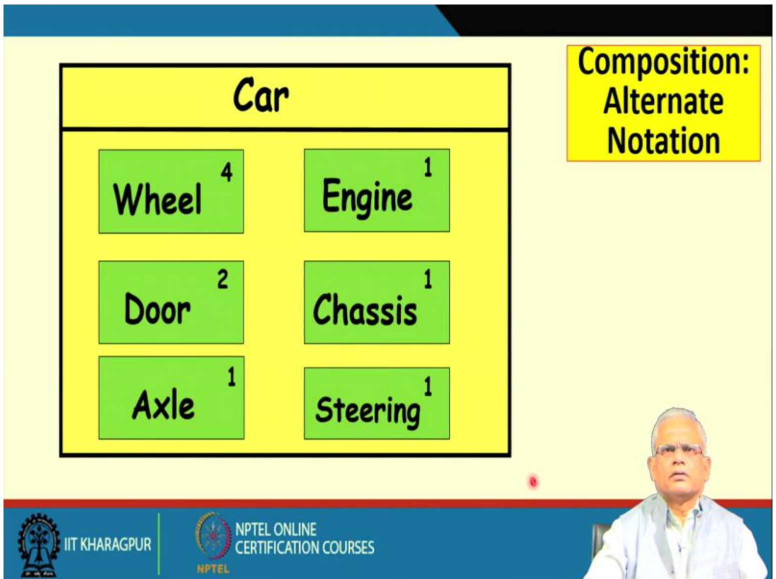
[Refer Slide Time: 12:36]



Now, let us look at the representation of the Composite relation (in the above slide). Here, the diamond symbol is used, but it is a filled diamond. That is the only difference in representation of the Composite relation and the Aggregate relation. Here the cardinality '1' that we write is optional, by default, it is 1. Here, we can have a multiplicity associated on Item end.

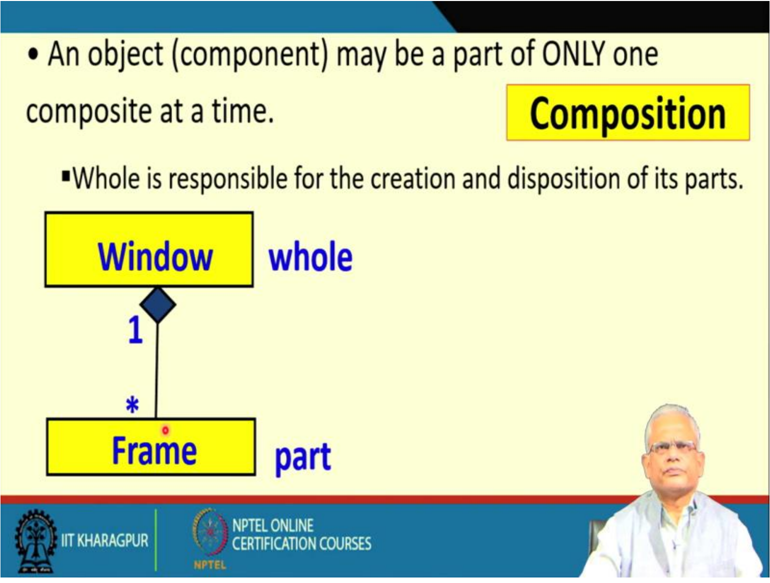
We need to remember that the life time of the component is the same as that of the composite. The composite gets created, along with the components get created. The composite gets destroyed, along with the components also get destroyed.

[Refer Slide Time: 13:31]



This is the alternate notation (in the above slide), for representing composite relation. A car contains 4 wheels. It has 2 doors, 1 axle, 1 engine, 1 chassis and 1 steering. So, this notion is more intuitive and readable. Wherever required, we can use this notation or the other notation, where we use the filled diamond.

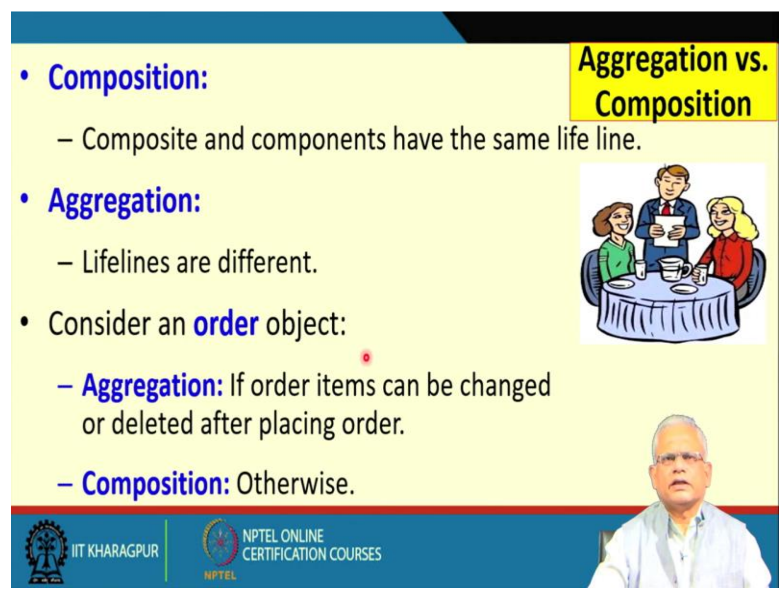
[Refer Slide Time: 14:19]



And we have just mentioned, that a component or an object can be part of only one composite at a time. And the whole or the composite is responsible for creation and destruction of the components. Later, we will see that in the Java code implementation.

This is another example (in the above slide). A graphical window contains many frames.

[Refer Slide Time: 15:01]

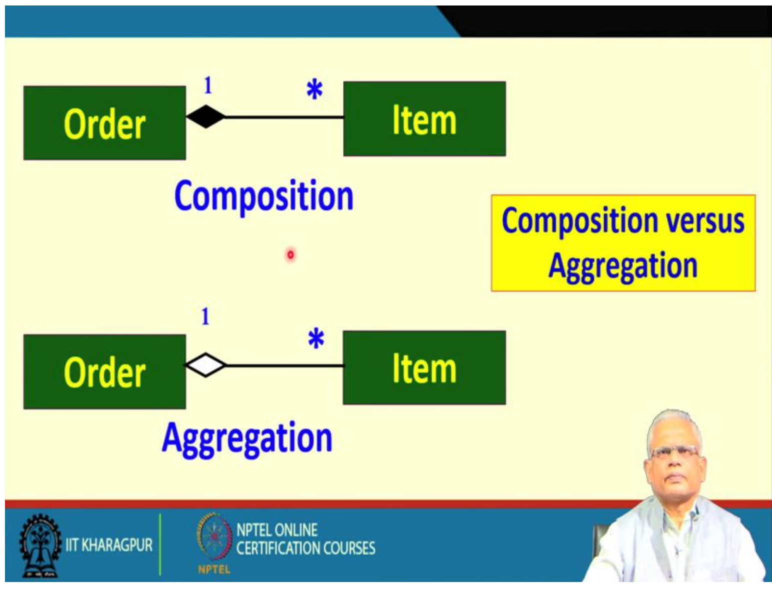


Let me repeat again, that the composite and the components have the same life line. They are created together, destroyed together. We cannot have just some components of a composite destroyed, without destroying the composite. It is not possible. We cannot add also components to a composite. Both composite and its components are created at the same instant. But in Aggregation Relationship, the part whole lifelines are different. We may add new components, we may delete some components, et cetera. Just to give an example of difference between this composite and Aggregation relation, let us consider an example.

Let us say, you went to a restaurant and ordered some items. And the waiter took down the order. Now, let's look at the situation, where you wanted to add a new order. After you given placed the order, you wanted to add a new item or maybe you just wanted to change or delete one item and order another item instead. Let say, in place of salad you wanted soup. But the waiter says, that's not possible: "You cannot change or added or deleted the order. You can only cancel the order and place a new order." Then we model that as a composite relation. A composition where once the order is created, all the items have been created with that. On the other hand, if we model it as an Aggregate relation, then it is possible that we delete some items from the order, add additional items and so on.

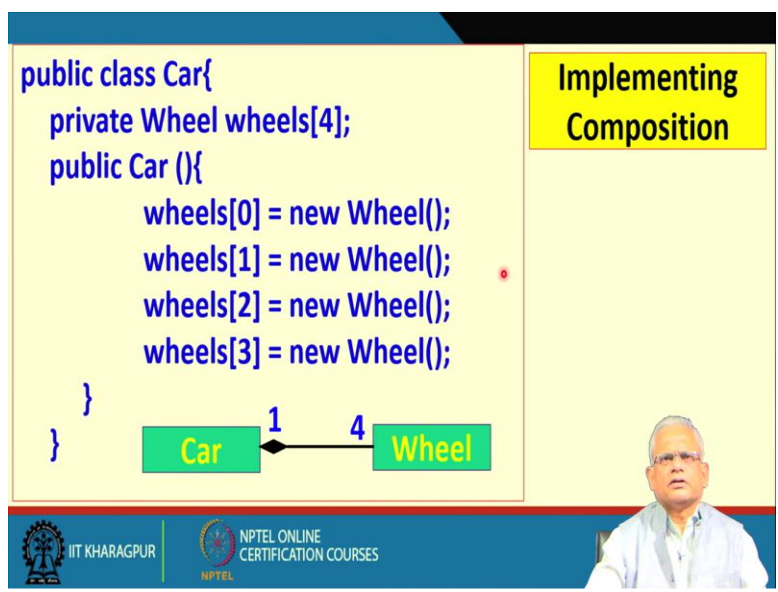
In Aggregation, an order item can be changed or deleted after placing the order. In Composition, we cannot do that. Once the order is placed, we cannot add any further items or delete items.

[Refer Slide Time: 17:58]



In the above slide, we can see composition and aggregation representation for same example. The first one is the composition, where we are not allowed to change any item, add item or delete item. We represent that in the form of a Composition Relationship, using a filled diamond. On the other hand, where we have the flexibility to add items, delete items, modify items and so on, we represent in the form of an Aggregation relation using empty diamond.

[Refer Slide Time: 18:39]

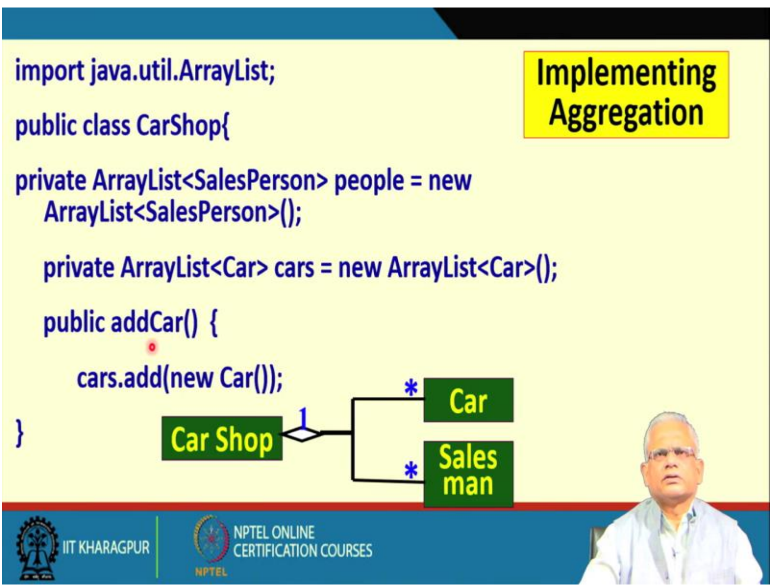


Now let's look at the implementation of the Composition Relationship. A car contains 4 wheels. Here, the car is the composite, wheels are the components. Now here, when a car object is created, the corresponding constructor is called and inside the constructor, the 4 wheels are created (as shown in the above slide). Here wheels are the metal frame, it is not the tyre. It is the metal frame inside the car.

So, when car is created, wheels are also created. Wheels are not the tyre. We can replace tyre, but here wheels of metal frame we cannot replace. So, 4 wheels of the car created in the constructor. As soon as the car object is created, 4 wheels are created. We cannot have a car and take out 1 wheel. We cannot have a 3-wheeler car or we cannot have a 5-wheeler car.

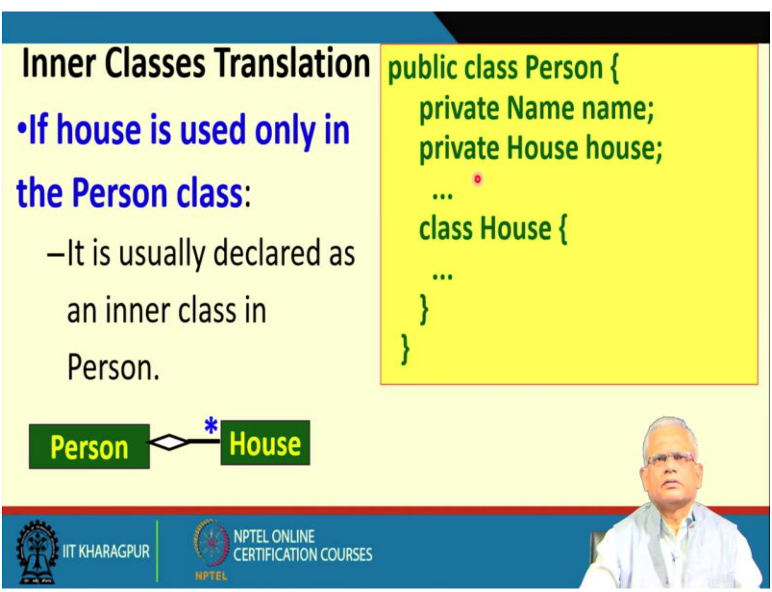
And that is why once the car is created, we have its wheels also created inside the constructor. So, we have seen one possible implementation of the Composition Relationship. All the components are created together and when the car object is destroyed, the components are also destroyed.

[Refer Slide Time: 20:22]



Now let's look at the Aggregation relation (in the above slide). Let say a car shop is an aggregate of many cars and many salesmen. Here, we use an ArrayList, because that's easier to add a number of components here and delete them. We have the SalesPerson is an ArrayList of people. And the cars is an ArrayList of cars. And we can add car, delete car and so on.

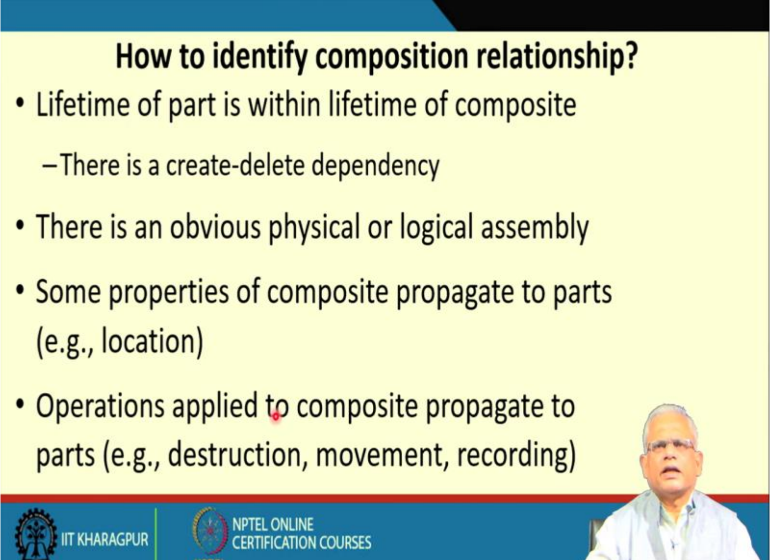
[Refer Slide Time: 21:17]



And we might also mention, that if it is a Composite relation, we might use the Inner Class Translation. That is, if a house is used only inside the Person class, we would make it an inner class of a person, because the house is not used elsewhere. The house is created and used inside the Person class. So, the person has house and the house is used only inside the

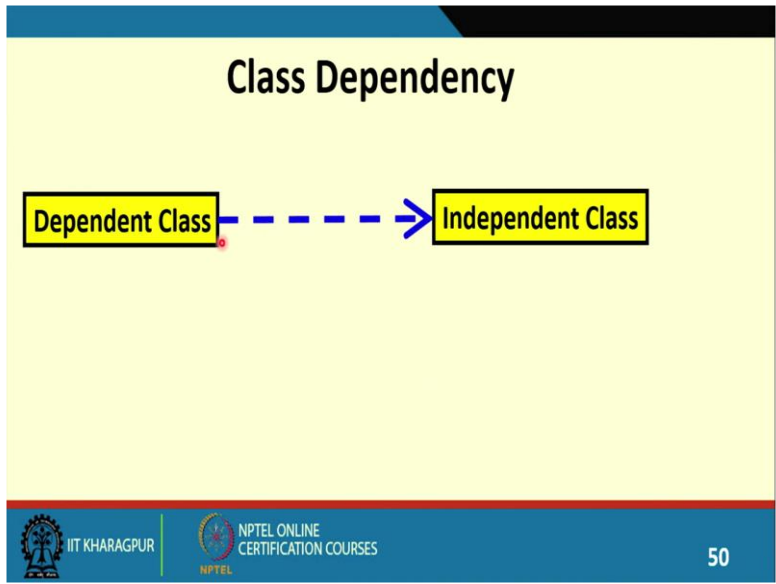
Person class. And it is created, destroyed inside the Person class. So, we might use it as an inner class and create an instance of house inside the person class, because it is not used elsewhere.

[Refer Slide Time: 22:31]



Now, let see that given a text description, how do we identify a composite relation between the classes. The first thing is that, composite and component have the same lifeline. They have the create-delete dependency that is the composite creates the component classes within the constructor and also deletes when the composite is destroyed. Also, there is an obvious physical or logical assembly. We looked at the example of car, which has 4 wheels. We looked at a polygon, or a circle. We create a circle, the point is defined is also created and we delete the circle, the point is also deleted. And also, the properties of the composite propagate to the part. For example, we move the circle, then the point also moves.

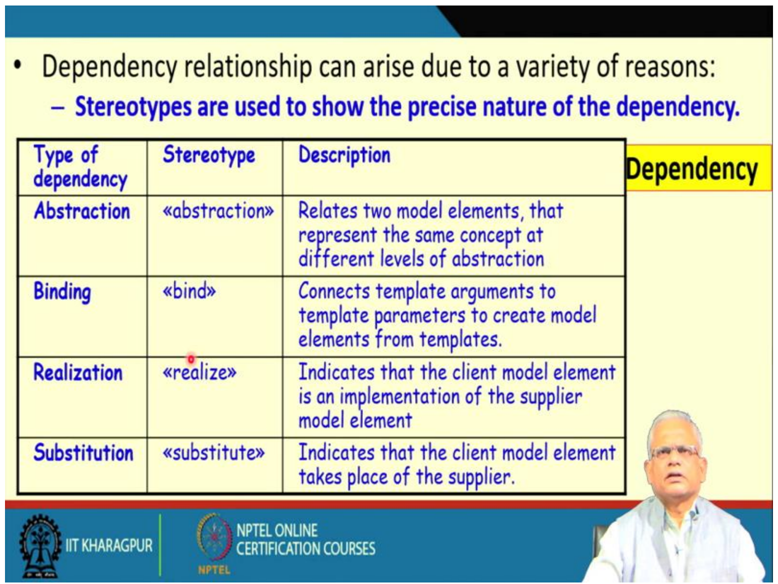
[Refer Slide Time: 24:19]



Now let's look at the fourth relation: The Dependency. Represented by a dotted open arrow (as shown in the above slide), and the arrow is from the dependent class to the independent class. When there is any change made to the code in the dependent class, the independent class not change. But whatever changes you make in the independent class, the dependent class will have to be compiled and also, the code possibly has to be changed. So, that is the meaning of the dependency.

The dependency, the meaning here is that, the independent class is independent of any changes of the dependent class. We might add new methods, we might change parameters inside the dependent class method, et cetera. No change need to the independent class. But if anything changes in the independent class, the dependent class possibly needs to be changed or at least need to recompiled. For example, a parameter to a method changes: there are 3 parameters and we change to 4 parameters. Then we will have to change the dependent class.

[Refer Slide Time: 25:54]



UML identifies a variety of reasons, for which the dependency between classes arises. One is Abstraction: a class is an abstract class and it is, its derived classes depend on the Abstraction, Binding: the template arguments are bound to a concrete class, Realization: interface is realized, Substitution: the client model elements take place of the supplier, so, we have a base class substituted by a derived class.

So far, we looked at two types of relation: One is the aggregation and another one is the composition. We also looked at the implementation of these relations. And in Composition, the additional restriction is that, the components are created and destroyed with the composite. That is there lifeline is the same.

And then we looked at the dependency relation. Two classes are said to be dependent, if the change of the independent class affects the dependent class, in the sense that we need to also change the dependent class and at least have to recompile. And we have also seen that there are many reasons why dependency between classes arise. There are many other reasons, why there can be dependency. We will take some examples in the next session. And see that how the dependency arises and what is the implication. And we will also see, given a dependency relation, what is the possible implementation in Java code.

We are already at the end of the session. We will stop here.

Thank you.