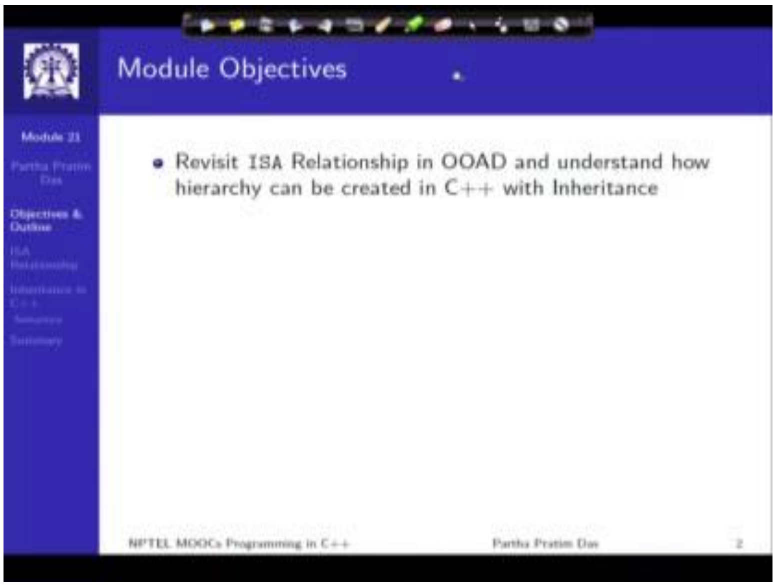
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 36
Inheritance: Part I

Welcome to Module 21 of Programming in C++. From this module continuing over the next number of modules, we are going to start the discussion on most important aspect of an object oriented programming language that is, the dealing of inheritance amongst classes and amongst objects. We have so far learnt how classes can be defined in C++, how objects can be instantiated. We have learnt about the feasibility of different data members and member functions of a class. We have learnt about the construction and destruction process and different lifetime issues of objects. We have also talked about other features that are related to variety of extension or exceptions to encapsulation and access by functions in terms of friend functions, in terms of static functions and so on, and we have seen how overloading of different member functions and global functions can be done.

Now, inheritance is one topic which kind of will combine all these understanding into building the core backbone of design of object based systems. Therefore, before we start of studying this in depth I would urge all of you to revise and be very thorough about the different features of C++ that we have discussed so far because we would be referring to all of them in a very frequent regularity now together.

(Refer Slide Time: 02:42)



Module Objectives

Module 21

Partha Pratim Das

Objectives & Outline

ISA Relationship

Inheritance in C++

Semantics

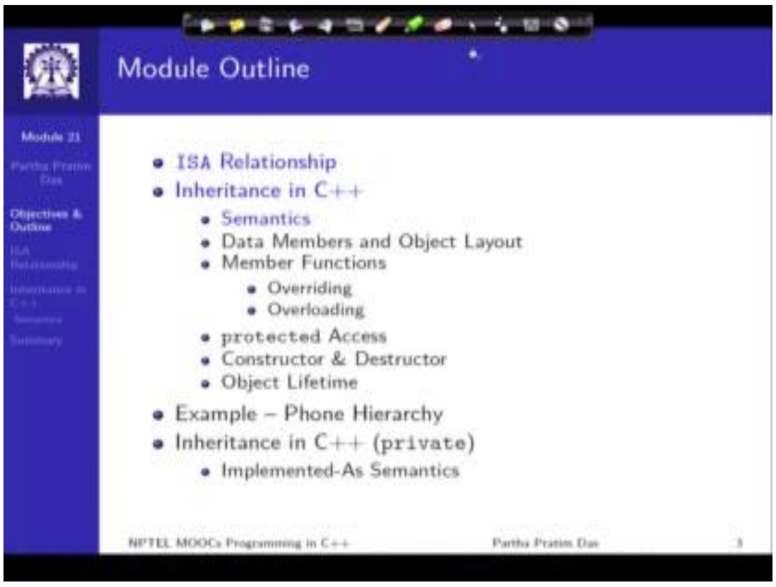
Summary

- Revisit ISA Relationship in OOAD and understand how hierarchy can be created in C++ with Inheritance

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

Now, for discussing inheritance as I have already mentioned this will span over a number of modules. In this particular module, we would try to revisit the ISA Relationship or hierarchy of object oriented analysis and design and see how that can be created in terms of C++ inheritance.

(Refer Slide Time: 03:04)



Module Outline

Module 21

Partha Pratim Das

Objectives & Outline

ISA Relationship

Inheritance in C++

Semantics

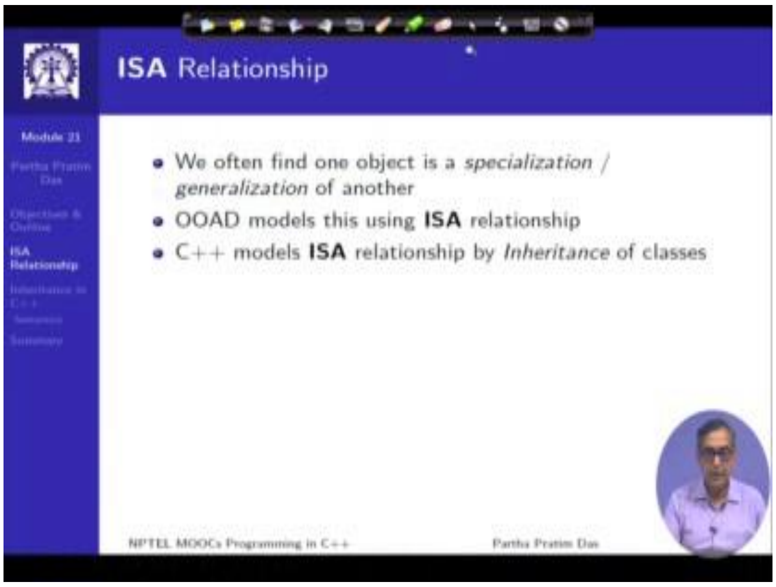
Summary

- ISA Relationship
- Inheritance in C++
  - Semantics
  - Data Members and Object Layout
  - Member Functions
    - Overriding
    - Overloading
  - protected Access
  - Constructor & Destructor
  - Object Lifetime
- Example – Phone Hierarchy
- Inheritance in C++ (private)
  - Implemented-As Semantics

NPTEL MOOCs Programming in C++ Partha Pratim Das 3

The outline that I present here is organized little bit differently. This is not just the outline of this current module. This is the outline of what I intend to discuss at the first level of inheritance in C++. There will be some subsequent sequel to this which we will talk about inheritance in a dynamic scenario, but this is the basic foundational aspects of inheritance. What we will do, as we move from one module to the other I will highlight the specific topics that the particular module we will discuss in terms of blue color. So, if you look into this of this whole context, this is what we intend to discuss in the module 21.

(Refer Slide Time: 04:01)



ISA Relationship

- We often find one object is a *specialization / generalization* of another
- OOAD models this using **ISA** relationship
- C++ models **ISA** relationship by *Inheritance* of classes

Module 21

Partha Pratim Das

Objectives & Outline

ISA Relationship

Inheritance in C++

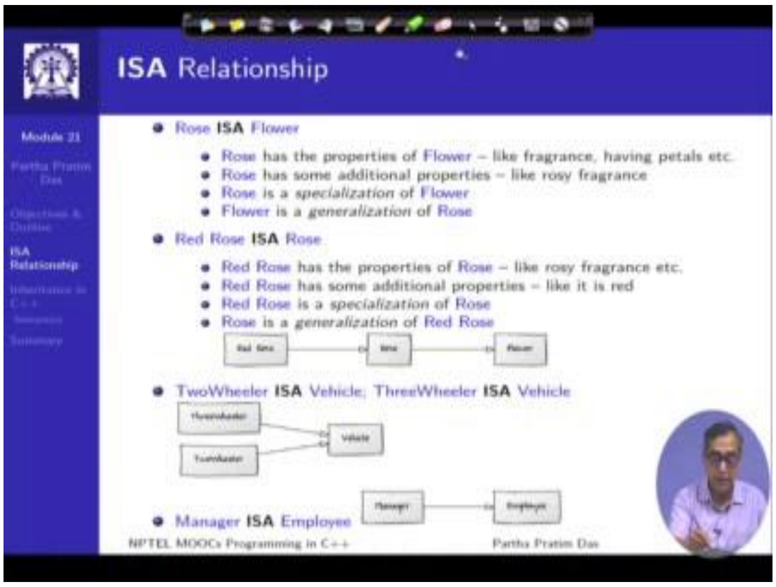
Semantics

Summary

NPTEL MOOCs Programming in C++ Partha Pratim Das

Having said that, let us get started now. So, we are aware of ISA relationship, we have often not talked about it in this course and you may be familiar with this otherwise also, that in the real world we often find that one object is a specialization or generalization of another object. Specialization and generalization these are the key. So, one object is a most specialized or generalized form of the other and this is known as ISA relationship and object oriented analysis and design treats this in depth and C++ model this ISA relationship by inheritance of classes.

(Refer Slide Time: 04:57)



ISA Relationship

Module 21

Partha Pratim Das

Objectives & Outline

ISA Relationship

Inheritance in C++

Semantics

Summary

- Rose **ISA** Flower
  - Rose has the properties of Flower – like fragrance, having petals etc.
  - Rose has some additional properties – like rosy fragrance
  - Rose is a *specialization* of Flower
  - Flower is a *generalization* of Rose
- Red Rose **ISA** Rose
  - Red Rose has the properties of Rose – like rosy fragrance etc.
  - Red Rose has some additional properties – like it is red
  - Red Rose is a *specialization* of Rose
  - Rose is a *generalization* of Red Rose
- TwoWheeler **ISA** Vehicle; ThreeWheeler **ISA** Vehicle
- Manager **ISA** Employee

NPTEL MOOCs Programming in C++

Partha Pratim Das

So, before getting into the inheritance logic of C++, let us quickly take a look into the ISA relationship. Let say what do you say is a specialization generalization? Suppose, we say that rose is a flower. So, what do you mean by saying rose is a flower. We say mean that rose, this is the special side, this is the general side that rose is a specialization of flower that is it has all the properties that flower has, like whenever we say flower; certain concepts come to our mind, like it will have fragrance, it will have bright color, it will have petals and so on. So, rose has all the properties that a flower is suppose to have. But in addition, rose has some additional properties that is why you would like to specialize that. For example, a rose has very specific rosy fragrance which is not similar to the fragrance of many of the other flowers.

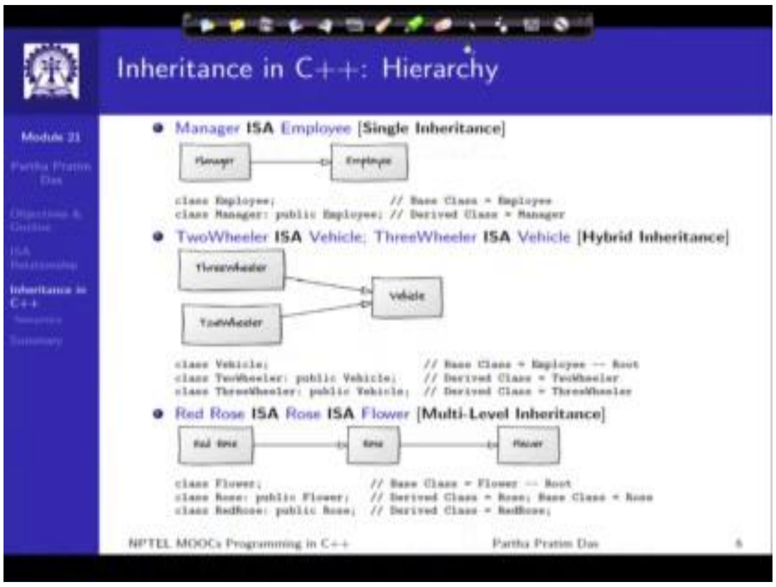
So, when this is done we say that rose is a specialization of flower and we can say the same thing in a reverse way. We can say that flower is a generalization of rose. So, if we have again a red rose and a rose, we can say a red rose is a rose in the same way red rose has all the properties of a rose, but it has an additional property like it is color is red and this is the generalization specialization that will exist between rose and red rose. So, we designate that in terms of what I have drawn here, this is known as many of you may know this is known as UML diagrams. Unified Modeling Language is a strong language to describe systems in the object oriented way. So, these designate the classes and this arrow which ends with open triangle at the end means specialization.

So, this is to be read; rose is a flower that is the direction of the arrow, red rose is a rose and so on. So, diagrammatically we can we will often depict the inheritance or specialization generalization in terms of this diagrams. Similarly, we can have; we can say that two wheeler is a vehicle, a vehicle which we use to move around. So, we can move around in 2 wheelers, so we can say that 2 wheeler is a vehicle. Similarly, 3 wheelers is a vehicle, but we can designate that in terms of vehicle being the generalization and 3 wheelers and 2 wheelers are two different specialization of the vehicle.

So, the generalization and specialization could relate in multiple different ways. So, this is a one kind where there is one generalized concept and there are multiple specialized concepts for that. We usually refer to this as a base and these, we refer to as derived. Different languages will use different nomenclature, for example, those of you who are familiar with Java, we will identify this base or the generalization as super class and derived or specialization as subclass. In some cases, the particular class which is the most generalized is often referred to as a root.

Yet another example of ISA relationship could be from the employee domain. We say manager is a employee which means that manager can perform all functions of an employee, but an employee can perform some; manager can perform some functions that the employee cannot do, possibly the manager can hire employees which employees themselves cannot do. So, this is the basic concept of ISA relationship which we want to bring in in the context of C++ programming.

(Refer Slide Time: 09:20)



Inheritance in C++: Hierarchy

Module 21

Partha Pratim Das

Objectives & Outline

ISA Relationship

Inheritance in C++

Semantics

Summary

- Manager **ISA** Employee [**Single Inheritance**]

```
class Employee;                 // Base Class = Employee
class Manager: public Employee; // Derived Class = Manager
```

- TwoWheeler **ISA** Vehicle; ThreeWheeler **ISA** Vehicle [**Hybrid Inheritance**]

```
class Vehicle;                       // Base Class = Employee -- Root
class TwoWheeler: public Vehicle;    // Derived Class = TwoWheeler
class ThreeWheeler: public Vehicle;  // Derived Class = ThreeWheeler
```

- Red Rose **ISA** Rose **ISA** Flower [**Multi-Level Inheritance**]

```
class Flower;                 // Base Class = Flower -- Root
class Rose: public Flower;    // Derived Class = Rose; Base Class = Rose
class RedRose: public Rose;   // Derived Class = RedRose;
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 6

So, in this, let me illustrate that if I want to encode this information of generalization specialization in C++, how do we go about doing that? So, the simplest of this cases is the manager is an employee, where we have just two classes one is the specialization of the other. So, we normally say refer to this as a single inheritance because you have just one relationship here. So, we write a class, say the representation for employees, class employee. Here, I have just used a kind of you know incomplete definition of a class. This does not have the definition, this is just saying that there is a class called employee because we will look into the data member and member function details later on.

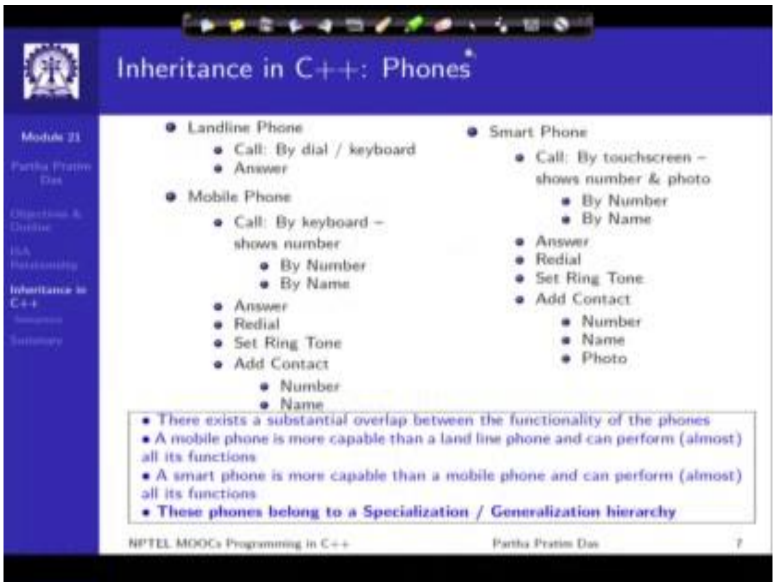
We are just interested to define the relationship between classes. So, given that, now we say that the manager is an employee through this form. So, here you can see that the critical addition that are making is after the class manager, there is a separator colon and then I am putting a new word public and then the name of the employee class and this is to be read as in the OOAD terms, this will be read as manager is an employee or manager derives from the employee. So, employee is the base class and the manager is a derived class.

In the two wheeler-three wheeler vehicle example; given this diagram we are actually dealing with hybrid inheritance. Hybrid inheritance is a context where one base class has more than one specialization and we can easily encode that in C++ in terms of a root class which is the vehicle. Then we say 2 wheelers is a vehicle by saying two wheeler colon public vehicles. 3 wheelers is a vehicle by two wheeler colon public vehicle. So, there is one root class and there one base class and there are two derived classes in this.

Finally, coming to the rose example, we have what is known as a multilevel inheritance because this is one level, this is another level. So, together it is a multilevel inheritance red rose is a rose, is a flower. So, here this is the base class flower is the base class and the derived class is rose derived class is rose, but if we look into the next pair of relationship then rose turns out to be the base class and red rose turns away to be the derived class. So, whether a particular class is a base class or a derived class cannot be decided absolutely just based on that class, it depends on where does the class lies on the hierarchy.

The classes which are leaf, at the leaf that is, which is not you know specialized by any other class are necessarily derived class. A class which does not have any super class or parent is necessarily the base class and the root class, but classes like who exist in the middle could be derived class for one part of the inheritance and base class for the other part of the inheritance, and we could really have variety of different complex and hybrid kind of inheritance structure that we would like to encode and understand in C++.

(Refer Slide Time: 13:02)



Inheritance in C++: Phones

- Landline Phone
  - Call: By dial / keyboard
  - Answer
- Mobile Phone
  - Call: By keyboard – shows number
    - By Number
    - By Name
  - Answer
  - Redial
  - Set Ring Tone
  - Add Contact
    - Number
    - Name
- Smart Phone
  - Call: By touchscreen – shows number & photo
    - By Number
    - By Name
  - Answer
  - Redial
  - Set Ring Tone
  - Add Contact
    - Number
    - Name
    - Photo

- There exists a substantial overlap between the functionality of the phones
- A mobile phone is more capable than a land line phone and can perform (almost) all its functions
- A smart phone is more capable than a mobile phone and can perform (almost) all its functions
- **These phones belong to a Specialization / Generalization hierarchy**

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

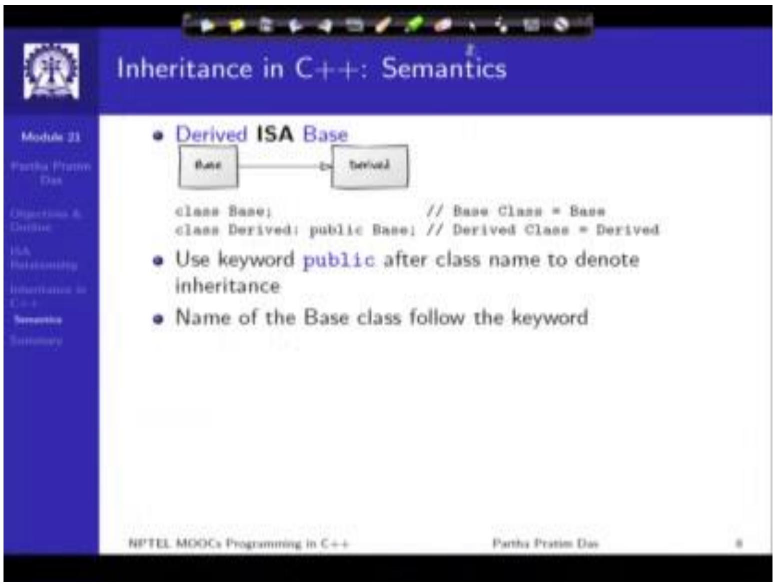
So, here I just take to you a little different kind of an example. We all are familiar with phones, you all use variety of phones. So, I am saying that, if we consider just 3 common types of phones that we use today; the landline phone, the mobile phone and the smart phone, then we can see that there is; and all that I am trying to do here is kind of associate the functionality that we mentally associate with these phones. So, say landline phone, what will be the functionality that you will associate; that you can make a call, you can answer a call.

For a mobile phone you can obviously, do all of that so, but you can do so many other things, possibly in a mobile phone you will be able to redial the last number, you will be able to set a ring tone, you will possibly have an address book where you can put your contacts by number and name and so on so forth, there would be several others. So, I can see that the basic functionality of a landline phone is also satisfied by the mobile phone, this already exist. So, I can say that a mobile phone is a landline phone; this ISA does not mean that it is same as, but this means that the mobile phone can satisfy all the functionality that the landline phone can satisfy.

Similarly, if I come to a smart phone, then I will have again these functionalities of call and answer, redial all this, but I may have some additional functionality. Typically that I can associate a photograph with my contact and if I associate a photograph with my contact then possibly smart phone at the time of call or at the time of answering, receiving the call, at the time of redialing will be able to show that photograph. So, there is commonality between; there is a substantial commonality between these different kinds of phone, but as we go from one kind of phone to the other, landline to mobile we find that there are some additional functionality that come in. As we go from mobile to smart phone we find yet some more functionalities coming in and in that way the phones form a very nice small specialization generalization hierarchy.

So, this was just to sensitize you, just to make you aware of the different kinds of gadgets and their hierarchies that are possible. At a later module we would like to take up the phones and actually do a complete design of the inheritance structure but now, let me move on to the concrete semantics of what the C++ inheritance would mean.

(Refer Slide Time: 15:56)



Inheritance in C++: Semantics

Module 21

Partha Pratim Das

Objectives & Outline

ISA Relationship

Inheritance in C++

Semantics

Summary

- Derived **ISA** Base

```
class Base;                    // Base Class = Base
class Derived: public Base;    // Derived Class = Derived
```

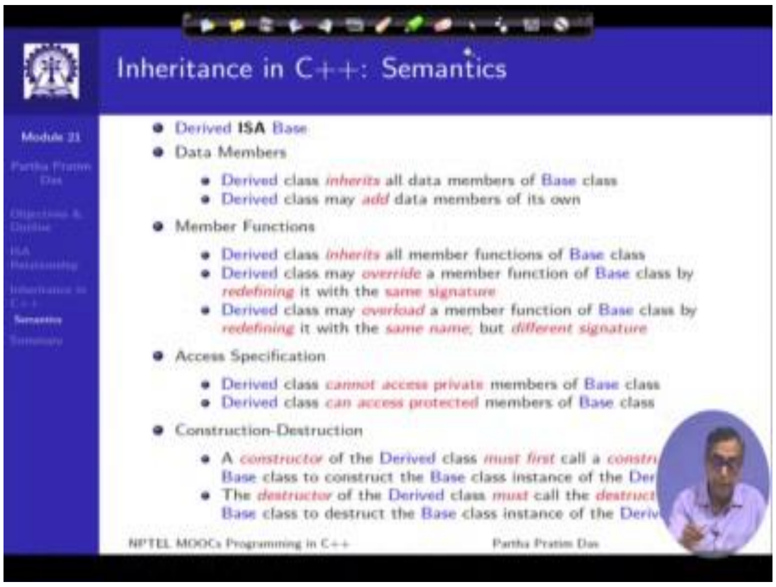
- Use keyword `public` after class name to denote inheritance
- Name of the Base class follow the keyword

NPTEL MOOCs Programming in C++ Partha Pratim Das 8

So, in general we will talk about two classes; the base class and the derived class. So, the ISA model is derived; ISA base. So, the name of the derived class is derived and the name of the base class is base and certainly we have already seen that this is how this is to be represented in C++. So, the new introduction is this public keyword as a keyword it already exist because we have used it for access specification, but here we are using it again for some special purpose of inheritance, and we will see what is the significance of this particular keyword and we will talk about their alternates, but for now just take it kind of as a prescription that if I want to say that the derived is a base then this is a way to say that, and after this keyword you can put the class name which is the generalized or the base class name that you need to specify.

Now, having said that certainly tokens up a whole lot of a question because as we will expect that the base will have different data members. The derived will also have different data members, base will have different methods, derived will also have different methods. Base will need to be constructed, derived will need to be constructed and so on.

(Refer Slide Time: 17:49)



Module 21

Partha Pratim Das

Objectives & Outline

ISA Relationship

Inheritance in C++

Semantics

Summary

Inheritance in C++: Semantics

- Derived **ISA** Base
- Data Members
  - Derived class *inherits* all data members of Base class
  - Derived class may *add* data members of its own
- Member Functions
  - Derived class *inherits* all member functions of Base class
  - Derived class may *override* a member function of Base class by *redefining* it with the *same signature*
  - Derived class may *overload* a member function of Base class by *redefining* it with the *same name*, but *different signature*
- Access Specification
  - Derived class *cannot access private* members of Base class
  - Derived class *can access protected* members of Base class
- Construction-Destruction
  - A *constructor* of the Derived class *must first* call a *constr* Base class to construct the Base class instance of the Der
  - The *destructor* of the Derived class *must* call the *destruct* Base class to destruct the Base class instance of the Deriv

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, what we need to very carefully specify and understand is; what is the semantics of, how this base derived relationship work? So, I would just like to outline at the very beginning and then we will take up each one of these and try to do a more detailed discussion. I would first like to outline that when you talk about semantics of inheritance in C++, these are the first level items that you need to be very careful about. The first thing is, I am sorry; the first thing is data members. So, we will say that the derived class inherits all data members of the base class. So, even if you do not have any data member in the derived class, it will still have a lot of data members which are all the data members of the base class, but it can further add more data members.

I will come to examples later on for each one of these, but I am trying to tell you the basic principle. The basic principle of inheritance is I have some concepts which is this and I am trying to specialize to provide a more special concept which need to satisfy everything that the general concept satisfies. So, the base, the derived will have to satisfy everything that the base satisfies. So, it needs to have all the data members, but it can add its own to further refine the whole concept.

In terms of member functions again, we will see that there are very key ideas that come in. The first part is the same that a derived class inherits all the member functions of the base class, but then there is a major difference that can; as you inherit a member function you have the choice of re-implementing it redefining it without actually changing it is signature. That is you want to again define the same function with a different kind of algorithm and if you do that then you say that you are overriding the member function in the base class; be very cautious because this concept sounds extremely close to the concept of overloading and therefore, there is a good possibility that you will start confusing it with the overloading and what makes things worst is you can actually also have overloading in the context of inheritance.

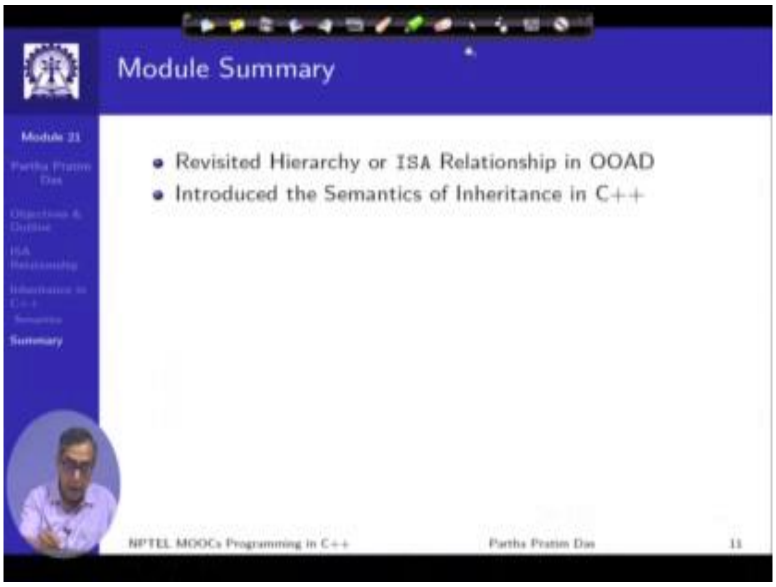
The difference is when you redefine a function with the same signature it is overriding. When you redefine a function with the different signature it is called overloading. So, these are the different semantics that we will need to understand in terms of the member functions behavior in under inheritance.

The next aspect that we need to look into is in terms of access specification. Now, naturally in terms of access specification we know that there is private access and there is public access. Private access is for the members of the class, public access is for everybody. So, a derived class of a base class is certainly not a part of that base class. So, a derived class cannot access the private members of the base class. It can access only the public members, but then we will see that that will lead to a lot of difficulties because the semantics of specialization that we are trying to put together particularly with reference to overriding will become extremely difficult to code, if the derived class do not have any access to the internals of the base class. So, with that a new type of access specifier, the protected access specifier is created to support a good semantics of inheritance. So, we will have to learn and understand about, what is this third type of visibility or access specification that is supported in inheritance.

Finally, naturally objects need to be constructed destructed and when we have inheritance between the base and the derived class, then a derived class instance will need a derived class constructor to be called, but the derived class constructor in turn will have to construct a base class object by calling the base class constructor. So, this is the basic understanding that we will need to build up as to how the base class object and the derived class object get related between themselves and how they can interact.

Similarly, when a derived class object will need to be destructed then the destructor of the base class will have to be invoked, so that you can destroy the base class part of the derived class object. So, this is just the tip of all the major semantics that we need to understand in terms of understanding how to use inheritance and how to model different real world scenario of hierarchy in a very efficient manner in the C++ language.

(Refer Slide Time: 23:53)



Module Summary

- Revisited Hierarchy or ISA Relationship in OOAD
- Introduced the Semantics of Inheritance in C++

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

So to summarize, we have in this just revisited the hierarchy of OOAD hierarchy concept of ISA relationship and class hierarchy concept of object oriented analysis and design, and we have introduced the basic notion of inheritance in C++ and noted what are the different aspects of semantics that we need to understand, that we need to really be master of so that we can use inheritance in an effective manner.