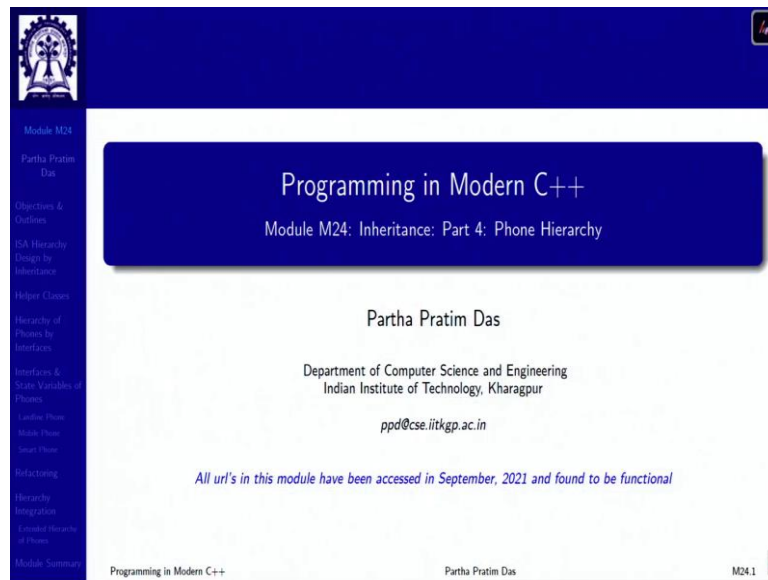


Programming in Modern C++
Professor Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Lecture 24
Inheritance: Part 4
Phone Hierarchy

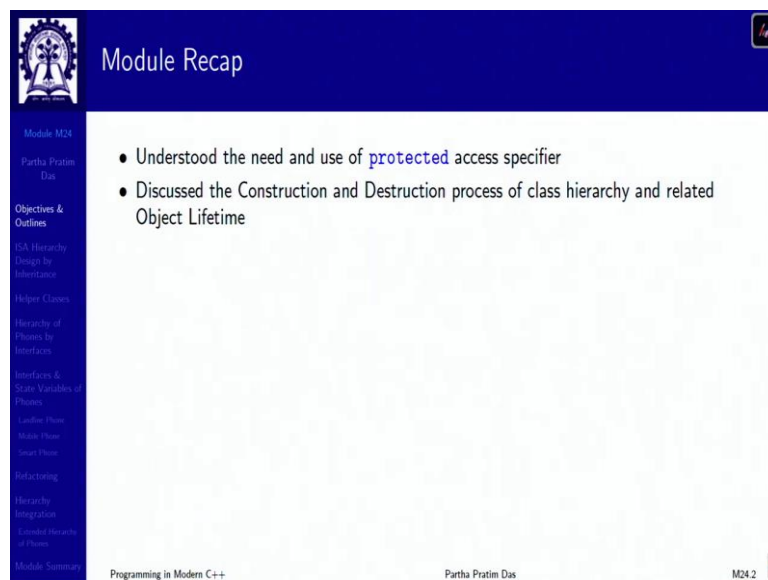
(Refer Slide Time: 00:34)



The slide features a dark blue header with the IIT Kharagpur logo on the left and a small icon on the right. Below the header is a white navigation sidebar with a list of topics: Module M24, Partha Pratim Das, Objectives & Outlines, ISA Hierarchy Design by Inheritance, Higher Classes, Hierarchy of Phones by Interfaces, Interfaces & State Variables of Phones, Landline Phone, Mobile Phone, Smart Phone, Abstracting, Hierarchy Integration, Extended Hierarchy of Phones, and Module Summary. The main content area has a dark blue background with white text. At the top, it says 'Programming in Modern C++' and 'Module M24: Inheritance: Part 4: Phone Hierarchy'. Below that, the author's name 'Partha Pratim Das' is displayed, followed by his affiliation: 'Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur' and his email 'ppd@cse.iitkgp.ac.in'. A note at the bottom states: 'All url's in this module have been accessed in September, 2021 and found to be functional'. The footer contains 'Programming in Modern C++', 'Partha Pratim Das', and 'M24.1'.

Welcome to Programming in Modern C++, we are in week 5, and I am going to discuss module 24 in the series of inheritance modules now.

(Refer Slide Time: 00:41)

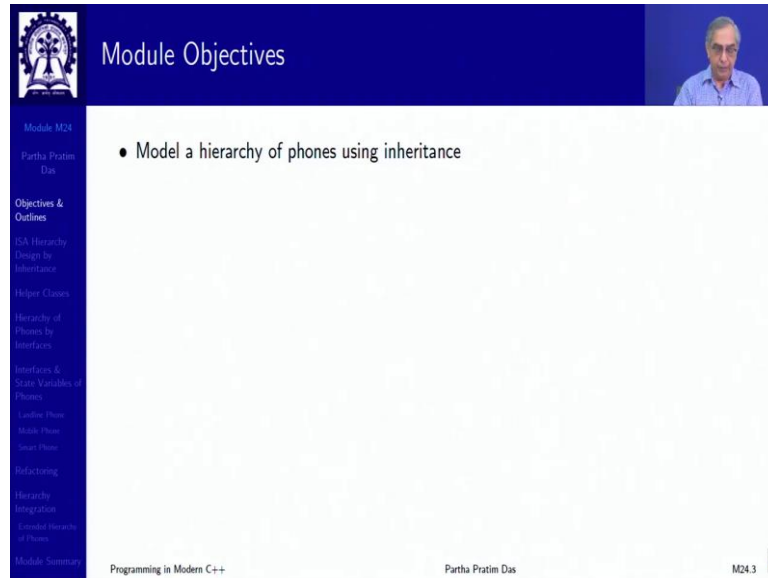


The slide features a dark blue header with the IIT Kharagpur logo on the left and a small icon on the right. Below the header is a white navigation sidebar with the same list of topics as the previous slide. The main content area has a white background with a dark blue header that says 'Module Recap'. Below the header, there is a bulleted list: 'Understood the need and use of `protected` access specifier' and 'Discussed the Construction and Destruction process of class hierarchy and related Object Lifetime'. The footer contains 'Programming in Modern C++', 'Partha Pratim Das', and 'M24.2'.

So, in the last module, we have understood the need and use of protected access specifier which is very critical to provide the extension of the encapsulation, information hiding

framework in combination with the ISA hierarchy or inheritance, we discuss the construction and destruction process of the class hierarchy and the related object lifetime.

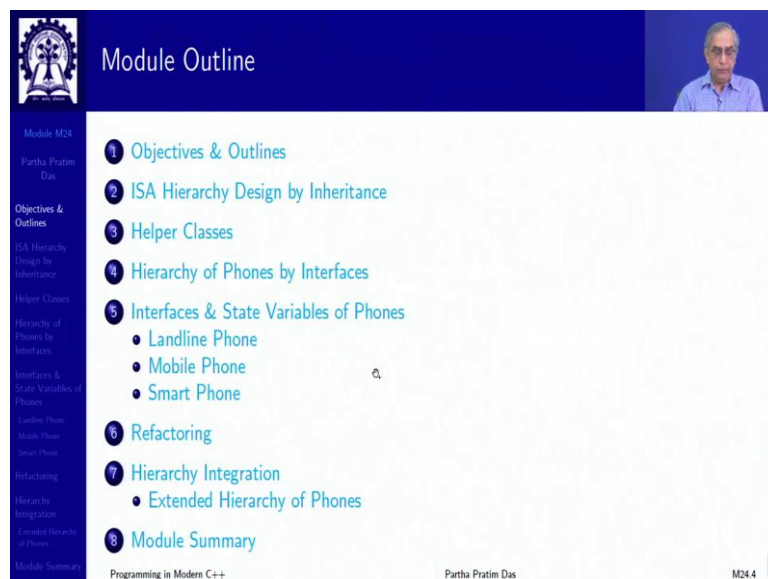
(Refer Slide Time: 01:17)



The slide is titled "Module Objectives" and features a video feed of the presenter in the top right corner. On the left, there is a vertical navigation menu with the following items: "Module M24", "Partha Pratim Das", "Objectives & Outlines", "ISA Hierarchy Design by Inheritance", "Helper Classes", "Hierarchy of Phones by Interfaces", "Interfaces & State Variables of Phones", "Landline Phone", "Mobile Phone", "Smart Phone", "Refactoring", "Hierarchy Integration", "Extended Hierarchy of Phones", and "Module Summary". The main content area contains a single bullet point: "• Model a hierarchy of phones using inheritance". The footer includes "Programming in Modern C++", "Partha Pratim Das", and "M24.3".

So, in a way, we are now equipped with the basic C++ features for support inheritance, support on ISA hierarchy. So, it is time to look at a small example, to get a comprehensive idea about how to put all these together and design a hierarchy of concepts in terms of C++ code from the abstract ISA hierarchy or ISA relationship of multiple concepts in object-oriented analysis and design. So, this module we will focus on that using the hierarchy of phones, simple hierarchy of phones that we had introduced earlier in earlier module.

(Refer Slide Time: 02:07)



The slide is titled "Module Outline" and features a video feed of the presenter in the top right corner. On the left, there is a vertical navigation menu with the following items: "Module M24", "Partha Pratim Das", "Objectives & Outlines", "ISA Hierarchy Design by Inheritance", "Helper Classes", "Hierarchy of Phones by Interfaces", "Interfaces & State Variables of Phones", "Landline Phone", "Mobile Phone", "Smart Phone", "Refactoring", "Hierarchy Integration", "Extended Hierarchy of Phones", and "Module Summary". The main content area contains a numbered list of 8 items: "1 Objectives & Outlines", "2 ISA Hierarchy Design by Inheritance", "3 Helper Classes", "4 Hierarchy of Phones by Interfaces", "5 Interfaces & State Variables of Phones" (with sub-bullets: "• Landline Phone", "• Mobile Phone", "• Smart Phone"), "6 Refactoring", "7 Hierarchy Integration" (with sub-bullet: "• Extended Hierarchy of Phones"), and "8 Module Summary". The footer includes "Programming in Modern C++", "Partha Pratim Das", and "M24.4".

So, this is the outline which will be available on left.

(Refer Slide Time: 02:12)

ISA Hierarchy Design by Inheritance

Module M24
Partha Pratim Das
Objectives & Outlines
ISA Hierarchy Design by Inheritance
Helper Classes
Hierarchy of Phones by Interfaces
Interfaces & State Variables of Phones
Ladder Phone
Mobile Phone
Smart Phone
Refactoring
Hierarchy Integration
Extended Hierarchy of Phones
Module Summary

Programming in Modern C++ Partha Pratim Das M24.5

So, we start with ISA hierarchy designed by inheritance.

(Refer Slide Time: 02:18)

Approach to Modeling Hierarchy

Module M24
Partha Pratim Das
Objectives & Outlines
ISA Hierarchy Design by Inheritance
Helper Classes
Hierarchy of Phones by Interfaces
Interfaces & State Variables of Phones
Ladder Phone
Mobile Phone
Smart Phone
Refactoring
Hierarchy Integration
Extended Hierarchy of Phones
Module Summary

- Identify the **Concepts and their ISA relationships** to define the hierarchy: model with public inheritance
- Identify and model **Helper classes** - lower level UDTs to define components
- Identify the **Interface** of each concept: signatures of public member functions
- Identify the **State Variables** of each concept: types of private / protected data members (also member functions used for ease of implementation)
- **Refactor** common data members and member functions between specialized and generalized classes to link the classes by inheritance
- **Integrate the hierarchy** with abstract (pure) interface
- Explore **extendability**
- *We illustrate with the phone hierarchy*

Programming in Modern C++ Partha Pratim Das M24.6

So, there is no very strict process to do the modeling and implementation of the hierarchy using the inheritance in C++, but I have tried to highlight a kind of flow that you can adopt to for your cases whenever you need to do that. So, I am calling this a kind of overall approach to modeling hierarchy. Now, I will start from the second point, I will come back to the first.

So, the key of this design is identifying the concepts and there is a relationship to define the hierarchy. So, because it is that real world, so, this is something which you will have to capture from the real world, understanding what is happening in the real world and see

what are the different concepts which are on one side kind of similar kind of related, but at the same time, they are not the same. So, there are variations between them, which we are trying to capture by the ISA relationship.

And we will try to model that with public inheritance. That is the first step that is captured the ISA relationships. Once you have done that, then you will need to identify the interface for each concept that is to deal with that concept. What does a user need to do? Or how does the concept manifest itself? So, that will proliferate it in terms of signatures of public member functions, which you need to identify.

So, this interface identification for every concept on the ISA relationships is a hierarchy is the next step to perform. Now, that will still remain abstract because you are just giving an interface which does not say how to implement that. So, for implementation, you have to identify the state variables or in other words, data members for each concept, which can keep information specific to that concept and help the interface or the method or the member functions, to use those values of the state variables to actually provide the service to the user.

And along with that, you will need to create proper encapsulation with the use of private or protected access specifiers. So, with this, having done this, you will now have, you will have the original ISA relationships, you have the concepts modelled in terms of tiny or maybe not so tiny classes. So, now you will have to link them by inheritance. So, these individual concepts will now have to be interrelated with the ISA relationship by providing the inheritance public inheritance mechanism that C++ has given.

We have to probably, we will try to provide an abstract or pure interface, we have not yet discussed it in the context of C++, but it is just the conceptual interface, which maybe everything in that hierarchy or most of the things in that, most of the concepts in that hierarchy should be supporting. And then we will have to refactor common data members because in terms of this, we will find that there are a lot of data members in different concepts which will be common.

So, we will try to put the common ones more towards the base classes or higher in the hierarchy and use the member functions to be specialized or generalized as the cases maybe. Once this is done, then you are ready with your classes in the hierarchy put it all together, this refactoring is very, very important, as we will illustrate with the classes in the hierarchy, and once you have done that, keep option to explore extendibility.

Normally, the hierarchy that you create in terms of ISA relationships is not a closed one, there may be more concepts that come in future and you may want to keep adding to that, so explore that extendibility. And this is to put in brief could be a broad approach that you can take to design the hierarchy and coded in C++. So, illustrate with the phone hierarchy.

(Refer Slide Time: 07:25)

Slide M2.7: Helper Classes. The slide title is "Helper Classes". The main content area is blank with the text "Helper Classes" written in red. The slide includes a navigation sidebar on the left and a video feed of the presenter in the top right corner.

Slide M2.8: Helper Classes. The slide title is "Helper Classes". The main content area contains a table listing helper classes and their descriptions. The slide includes a navigation sidebar on the left and a video feed of the presenter in the top right corner.

Class	Description
<code>class <u>PhoneNumber</u></code>	12-digit phone number
<code>class <u>Name</u></code>	Subscriber Name (as string)
<code>class <u>Photo</u></code>	Image & Subscriber Name as alt text
<code>class <u>RingTone</u></code>	Audio & ring tone name
<code>class <u>Contact</u></code>	<u>PhoneNumber</u> , <u>Name</u> , and <u>Photo</u> (optional) of a contact
<code>class <u>AddressBook</u></code>	<u>List of contacts</u>
	o
	o
	+

Now, before I start doing it, you will find that these concepts on which you are actually building the hierarchy will often need to use a lot of other concepts, maybe small, maybe little big, which are not related to this hierarchy. But these are like user defined types that will be good to have very compact representation readable as well as encapsule level design of the entire hierarchy.

So, this is what I call as helper classes. So, for example, since we are dealing with phones, naturally irrespective of where I am in the hierarchy, what type of phone you will need to deal with phone numbers. So, I want to define a class for phone number, which keeps a 12-digit phone number that could be further details inside of this class which say whether it is, whether you have international country code.

Whether you have STD code as a part of it or it is just a mobile number, whether you have any other kind of structuring information in the phone number and so, on, you can have a lot of those. And in this current discussion, we are not getting into these aspects of the design because we are not going to do the design on the phone numbers, we are doing the design on the phones.

So, we assume that there is some helper class called phone number is already been designed or will be designed, we can just have it as an incomplete class to start with, which can be used whenever I need a phone number. Naturally, the subscribers, so will have their names. So, we will have subscriber name as string maybe first name, middle name, last name, there could be other information related to the name that you can keep.

You may want to have a helper class called photo to keep images of how does the subscriber look and have the alt text, alternate text for the image also captured in this class, you may have class for various kinds of ringtones, their IDs, names and their audio files and so on. We will have a helper class contact containing the phone number, name and optionally the photo of a contact to create your every element of the address book, when the address book necessarily becomes a list of contacts.

There could be more as well as you can understand, but this is for the purpose of illustration, this is a basic set of kind of unitary concepts I can say utilities which are needed to be able to design my phone hierarchy in an easier way without this, I will need to kind of describe a lot of very low-level data like int, string and so on. I want to more make it object oriented and these objects of these classes will be frequently used in defining my actual hierarchy items.

(Refer Slide Time: 10:47)

Hierarchy of Phones by Interfaces

Programming in Modern C++ Partha Pratim Das M24.9

Hierarchy of Phones

```
graph LR;
    SmartPhone[SmartPhone] -- ISA --> MobilePhone[MobilePhone];
    MobilePhone -- ISA --> LandlinePhone[LandlinePhone];
```

- MobilePhone ISA LandlinePhone
 - LandlinePhone is *generalization* ✓
 - MobilePhone is *specialization* ✓
 - MobilePhone inherits the properties of LandlinePhone
- SmartPhone ISA MobilePhone
 - MobilePhone is *generalization*
 - SmartPhone is *specialization*
 - SmartPhone inherits the properties of MobilePhone
- ISA is transitive ✓

Programming in Modern C++ Partha Pratim Das M24.10

So, then I go to defining the hierarchy of phone by interfaces. So, the basic hierarchy the ISA relationship that I have is mobile phone, this is what we have done earlier, mobile phone ISA landline phone. So, landline is a generalization, mobile is a specialization, it inherits the properties of landline phone, the mobile phone. Similarly, smartphone ISA mobile phone.

So, mobile phone is a generalization of the smartphone and so on. It has a transitive relationship. So, we could have it more. And this is the basic starting point that I identify that the three concepts landline phone, mobile phone and smartphone, and this is the interrelationships in terms of generalization and specialization. So, (we will) our hierarchy is multi level, linear and we will start building upon that.

(Refer Slide Time: 11:47)

Interfaces of Phones

- **Landline Phone**
 - Call: By dial / keyboard
 - Answer
 - Caller ID (with special attached device)
- **Mobile Phone**
 - Call: By keyboard – shows number
 - ▷ By Number
 - ▷ By Name
 - Answer
 - Caller ID
 - Redial
 - Set Ring Tone
 - Add Contact
 - ▷ Number
 - ▷ Name
- **Smart Phone**
 - Call: By touchscreen – shows number & photo
 - ▷ By Number
 - ▷ By Name
 - Answer
 - Caller ID
 - 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**

Programming in Modern C++ Partha Pratim Das M24.11

So, interfaces you have already seen this slide before. So, we have already identified the interfaces that if these are the different concepts, then these interfaces basically tell us what is the service that this every concept does provide. Now, naturally, (you do not have an) as a phone user, you are really not concerned about what happens when you do a call or what happens when you have to do a redial or how the ringtone files are basically created.

And you are not concerned with any of that all that you want to do is I am a phone user I must be able to call, I must be able to answer a call, I want to redial on a number, I want to add contacts, I want to put a photo for my contact and so on. So, here for every concept, we define the interfaces that should be available for these concepts and that is what we will try to model in terms of our inheritance hierarchy in this case.

(Refer Slide Time: 12:57)

The slide is titled "Interfaces & State Variables of Phones". On the left, there is a navigation menu with the following items: Module M24, Partha Pratim Das, Objectives & Outlines, ISA Hierarchy Design by Inheritance, Helper Classes, Hierarchy of Phones by Interfaces, Interfaces & State Variables of Phones (highlighted), Landline Phone, Mobile Phone, Smart Phone, Refactoring, Hierarchy Integration, Unified Hierarchy of Phones, and Module Summary. The main content area is mostly blank with the text "Interfaces & State Variables of Phones" in red. At the bottom, it says "Programming in Modern C++", "Partha Pratim Das", and "M24.12".

So, let us, now, this was mostly conceptual, these interfaces. So, now, let us slowly start putting it in terms of our C++ paradigm, actual class paradigm to define the interfaces and the state variables in every case.

(Refer Slide Time: 13:15)

The slide is titled "Interface & State Variable: Landline Phone". On the left, the navigation menu is the same as the previous slide, with "Landline Phone" highlighted. The main content area shows a C++ class definition for `LandlinePhone` with handwritten checkmarks and annotations. The class has three private members: `PhoneNumber number_;`, `Name subscriber_;`, and `RingTone rTone_;`. The public methods are `LandlinePhone(const char *num, const char *subs);`, `void Call(const PhoneNumber *p);`, and `void Answer();`. There is also a friend function: `friend ostream& operator<<(ostream& os, const LandlinePhone& p);`. Handwritten annotations include a bracket grouping "Call: By dial / keyboard" and "Answer" with lines pointing to the `Call` and `Answer` methods. The footer contains "Programming in Modern C++", "Partha Pratim Das", and "M24.13".

So, what I do I take every concept and try to, based on the interface that I have already identified in terms of the hierarchy, I try to write a class corresponding to that. So, interface this will become member functions. And using the helper classes, I will try to identify a set of state variables or data members that will be required to support these interfaces.

So, if we look into landline, there are primarily two interfaces that we had identified, I have dropped the caller ID from here, because it is a little bit makes it more complex it is more

for illustration. So, I can say corresponding to this call, I will have a member function in the (landline class) landline phone class, which will call, what does it need to know for the call, at least it needs to certainly know the phone number.

So, it will take I mean here I have taken a const pointer it could be a const reference also, does not matter. I am not going into those details, but the basic idea is that it has to make a call and that is what is identified by this particular method. Similarly, to answer I will provide another member function which has to answer. Assume that these are just action items so there is no return value from this.

Now, since I have to make the call, I am expecting to call by the number. So, naturally as an entity of myself, I need to have a number of myself only then I can make a call, I cannot make a call to a number unless I have a number myself. So, I assume that there is a, there has to be a state variable, which is the phone number. And I have already taken care of this helper classes.

So, assuming that those helper classes are there, I am not going into discuss about what those details of those classes are, those are like tiny concepts that are basically supporting me. Naturally, I have to have my name as a subscriber and possibly I will have a ringtone, maybe in a landline is not suitable. So, it is does not appear in the, it does not appear in terms of the interface, but maybe it has a default value.

(Refer Slide Time: 16:06)

```
class LandlinePhone {
    PhoneNumber number_;
    Name subscriber_;
    RingTone rTone_;
};

public:
    LandlinePhone(const char *num, const char *subs);

    void Call(const PhoneNumber *p);

    void Answer();

    friend ostream& operator<<(ostream& os, const LandlinePhone& p);
};
```

So, what else do I need? Naturally, I will have a constructor. So, when I get the phone, I construct my landline phone object with my number and my subscriber name, ringtone I

have kept optional. So, this is a basic rationale of a very minimal design for the landline phone class. In addition, I have provided a friend function, for friend function for operator output streaming, which writes down say the landline number, this is not you can see that this is a friend function.

So, this is not the interface of the class, but this is more for us the developer, because while I develop this, I may need to do some debugging. And to do debugging, I need to be able to at any point I should be able to print that object details, so that I can get a view about the state variables. That is the reason it takes it is basically output streamer to be able to write any object of this class and this style will be followed everywhere. In fact, a very generic style for any kind of object to, any kind of C++ class design.

(Refer Slide Time: 17:28)

Interface & State Variable: Mobile Phone

- **Mobile Phone**
 - Call: By keyboard } shows number
 - ▷ By Number ✓
 - ▷ By Name ✓
 - Answer ✓
 - Redial ✓
 - Set Ring Tone ✓
 - Add Contact ✓
 - ▷ Number ✓
 - ▷ Name ✓

```

class MobilePhone {
    PhoneNumber number_;
    Name subscriber_;
    RingTone rTone_;
    AddressBook aBook_;
    PhoneNumber *lastDial_;
    void SetLastDialed(const PhoneNumber& p);
    void ShowNumber();

public:
    MobilePhone(const char *num, const char *subs);
    void Call(PhoneNumber *p);
    void Call(const Name& n);
    void Answer();
    void Redial();
    void SetRingTone(RingTone::RINGTONE r);
    void AddContact(const char *num = 0,
                   const char *subs = 0);
    friend ostream& operator<<(ostream& os, const MobilePhone& p);
};
  
```

Moving on, I have the mobile phone concept. So, now I try to do the mobile phone. First, the interfaces the call will have, call which is by phone number, but it also has an option to call by name. So, I include another call interface or in other words I overload the call. I will have to have an answer, then I will have to be a redial, setting of ringtone where I specify assuming that there is some enumerated names of ringtone specify which ringtone, I should be able to add a contact given the phone number and the name.

So, directly from the concept of the interfaces, I directly put them in terms of proper member functions of the class. Coming to the state variable naturally, I will need to have the phone number, the name, the ringtone. And certainly, now, I am assuming that there is a contact coming in. So, when will that contact recite? So, I need an address book. So, I am putting an address book here which as you know it should be a list of contacts.

I am talking about redialling, if I redial what does it mean, dial the last number to which a call was made either through call or by redial. So, I need to remember the phone number that was last dialled. I mean, this is the way you actually come to the state variables given the interfaces that you are trying to do. So, you are trying to take every interface and see well to be able to do this what do I need to know in terms of the state variables.

If I have to keep track of the last dial, I also need a member function to set this value. And this member function is not on my interface. This should get automatically set when I am doing a call or when I am doing a redial and I do not want the users to be able to come in and set a number because it is based on the internal functionality of the mobile phone.

So, I provide for a member function here, but just note that this member function is provided as a part of the private specifier which means that it is not visible externally, it is not a part of the interface of this class, but it is available to the developers for doing some tasks. Similarly, I have one for the showing number which will help us in terms of debugging, we also have output streaming.

(Refer Slide Time: 20:43)

The slide displays the following content:

- Smart Phone**
 - Call: By touchscreen - shows number & photo
 - By Number
 - By Name
 - Answer
 - Redial
 - Set Ring Tone
 - Add Contact
 - Number
 - Name
 - Photo

```
class SmartPhone {
    PhoneNumber number_;
    Name subscriber_;
    RingTone rTone_;
    AddressBook aBook_;
    PhoneNumber *lastDial_;
    void SetLastDialed(const PhoneNumber& p);
    void ShowNumber();
    unsigned int size_;
    void DisplayPhoto();

public:
    SmartPhone(const char *num, const char *subs);

    void Call(PhoneNumber *p);
    void Call(const Name& n);

    void Answer();

    void Redial();
    void SetRingTone(RingTone::RINGTONE r);
    void AddContact(const char *num = 0,
                   const char *subs = 0);

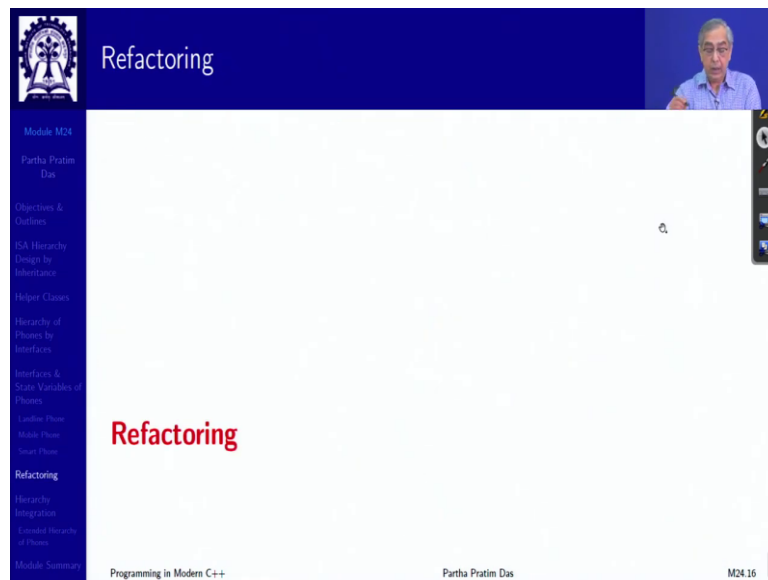
    friend ostream& operator<<(ostream& os, const MobilePhone& p);
};
```

Once you get this idea then we will understand that if we, when we take it further in terms of the smartphone the things just keep on happening in the same way I have two ways to call, I have a way to answer, I have a redial, I have a provision to set ringtone, I have add contact, I may be able to put photos as a part of that, I have the constructor the basic value. I need a state variable, phone number, subscriber name, tone, address book, last dialled, set dialled function for setting each showing the number.

And I will also, I am expecting that the photo will also come in. So, if the photo comes in, then when I like the show number, I need to be able to show the number of the address book content, address book contact. Similarly, I need to have the size of and display the photo that comes in. So, these are kind of you can assume it in a different way also, you can assume that the entire display is a, it is an interface of the contact.

So, you are just is going to use that interface. Or you can assume that you have mechanisms for doing this in your own implementation. But the key point to note that the things that we are putting in here alongside the state variables are not part of the interface, they are basically part of our implementation.

(Refer Slide Time: 22:30)



So, we have now been able to basically design the classes for the three concepts that are related to by the ISA hierarchy. And in the process, as the ISA hierarchy exists, ISA relationship exists between them, generalization specialization exists between them, there are a lot of commonalities, and we need to refactor to remove those commonalities and bring in the actual inheritance mechanism of C++ in.

(Refer Slide Time: 23:01)

```
class LandlinePhone { protected:
    PhoneNumber number_;
    Name subscriber_;
    RingTone rTone_;

public:
    LandlinePhone(const char *num,
                  const char *subs) :
        number_(num), subscriber_(subs),
        rTone_() {}
    void Call(const PhoneNumber *p);
    void Answer();

    friend ostream& operator<<(ostream& os,
                              const LandlinePhone& p);
};

class MobilePhone : public LandlinePhone { protected:
    //PhoneNumber number_;
    //Name subscriber_;
    //RingTone rTone_;
    AddressBook aBook_;
    PhoneNumber *lastDial_;
    void SetLastDialed(const PhoneNumber& p);
    void ShowNumber();

public:
    MobilePhone(const char *num,
                const char *subs) :
        LandlinePhone(num, subs), // Base ctor
        lastDial_(0) {}
    void Call(const PhoneNumber *p); // Override
    void Call(const Name& n); // Overload
    //void Answer(); // Inherited
    void Redial();
    void SetRingTone(RingTone::RINGTONE r);
    void AddContact(const char *num = 0,
                   const char *subs = 0);
    friend ostream& operator<<(ostream& os,
                              const MobilePhone& p);
};
```

So, in this refactoring process, say, I know mobile phone is a landline phone. So, I put these two classes side by side. As I put them side by side, I find that there is a call here, there is a call here. How will they be related as a specialization? Because I am trying to do this mobile phone, public landline phone to capture my actual ISA relationship. As I do that, I will, in mobile phone have a call function, call interface, which takes a constant pointer to a phone number. But the way to call will be different.

The internal implementation of the call will be different. So, I will need, I cannot use the implementation of this function in terms of the call in the mobile phone. So, I have provided the interface provided the signature once again. And in other words, I am just connecting them by overriding this call function by this call function. I see, I also observed that mobile phone has another mechanism to call which my landline phone does not have.

So, I have to put retain that call function, which means that I am basically overloading. I see that I have to answer. So, there is an answer in the landline phone, there is an answer here. And I have just assumed that there is nothing else or nothing specific that you need to do for the answer. So, I do not need to override it, I just inherited. Here, actual situation you will have to resolve out.

So, this is by this refactoring, I am combining these two through the insertion of the ISA constraint, the inheritance constraint and refactoring by making one as an overriding member function, one as an overloaded member function and one as a inherited member function. So, does my constructor will change because now landline phone

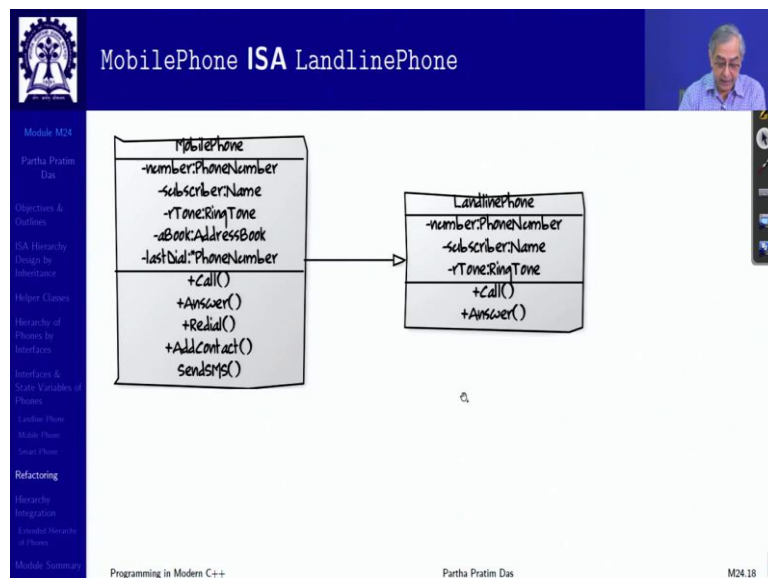
becomes a base for the mobile phone. So, that base will have to be constructed. I have last dialled here.

So, that will have continued to have here. And now this is on the interface side. What do I have to do in the case of the state variable side? These three state variables are common. So, if I have them as a part of my base class landline phone, I will certainly not be able to redefine them. So, when I create a mobile phone, I will create a landline phone object as a part of it in the base, which will have the phone number, subscriber and ringtone for the mobile phone object that I am creating.

So, I basically will omit these data members so that is why I have made them commented them out, rest of the data members naturally are new to mobile phone and will have to be included. Last but not the least, I have changed the placement of these data members and the kind of implementation required member functions earlier they were in private now I put them in protected.

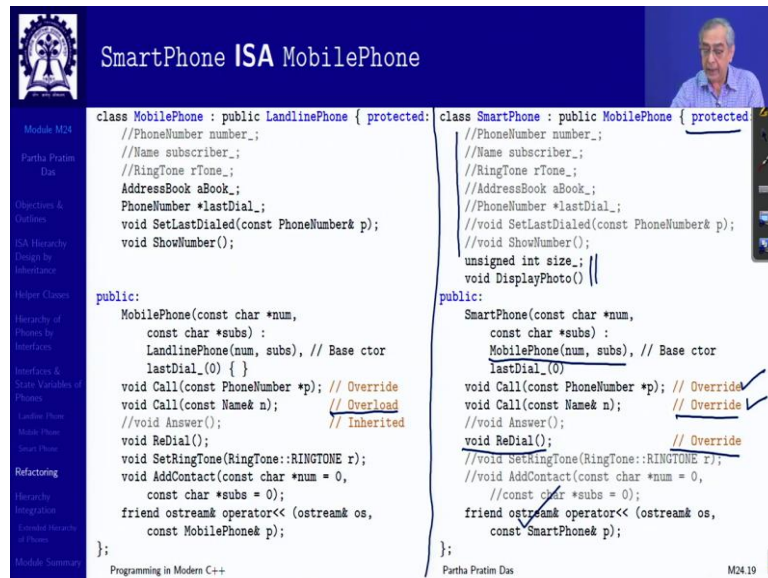
They were in private I have put them in protected because unless I put them in protected the next class the specialized class mobile phone cannot use it. So, unless I make this protected, the next class the smartphone will not be able to inherit them, not able to use them, we have seen all these. And in terms of output streaming, I have discussed this execute example earlier, I have to keep the streamer in the respective classes.

(Refer Slide Time: 27:23)



Now, once you. So, this is in the in terms of the object-oriented analysis and design diagrams.

(Refer Slide Time: 27:30)



```
class MobilePhone : public LandlinePhone { protected:
//PhoneNumber number_;
//Name subscriber_;
//RingTone rTone_;
AddressBook aBook_;
PhoneNumber *lastDial_;
void SetLastDialed(const PhoneNumber& p);
void ShowNumber();

public:
MobilePhone(const char *num,
const char *subs) :
LandlinePhone(num, subs), // Base ctor
lastDial_(0) { }
void Call(const PhoneNumber *p); // Override
void Call(const Name& n); // Overload
//void Answer(); // Inherited
void ReDial();
void SetRingTone(RingTone::RINGTONE r);
void AddContact(const char *num = 0,
const char *subs = 0);
friend ostream& operator<< (ostream& os,
const MobilePhone& p);
};

class SmartPhone : public MobilePhone { protected:
//PhoneNumber number_;
//Name subscriber_;
//RingTone rTone_;
//AddressBook aBook_;
//PhoneNumber *lastDial_;
//void SetLastDialed(const PhoneNumber& p);
//void ShowNumber();
unsigned int size_;
void DisplayPhoto() { }

public:
SmartPhone(const char *num,
const char *subs) :
MobilePhone(num, subs), // Base ctor
lastDial_(0)
void Call(const PhoneNumber *p); // Override ✓
void Call(const Name& n); // Override ✓
//void Answer();
void ReDial(); // Override
//void SetRingTone(RingTone::RINGTONE r);
//void AddContact(const char *num = 0,
//const char *subs = 0);
friend ostream& operator<< (ostream& os,
const SmartPhone& p);
};
```

Now, once you have understood this process, which is the basic refactoring, you will understand what happens when you want to enforce that mobile phone is a smartphone, exactly the same kind of things happen. Now, this we are specializing. So, you have further cases of override.

This here was overload because it was introduced in mobile phone, in case of smartphone it becomes the override because it is already there in the parent, your redial is also being made into an override, because in smartphone you will redial in a different way, all of these data members or member functions for implementation are already coming from the mobile phone class. So, you do not need them in the smartphone class, you just omit those.

You are left with what is specific to that certainly everything is in protected. You need to construct mobile phone base class in the constructor and need to provide output stream. So, the style becomes very similar very repeated, we just keep on doing that. And in this way, you are one by one stitching on the hierarchy in terms of the ISA relationship realizing that.

(Refer Slide Time: 28:50)

Hierarchy Integration

```
// Abstract Base Class - A Pure Interface
class Phone { public:
    virtual void Call(const PhoneNumber *p) = 0;
    virtual void Answer() = 0;
    virtual void ReDial() = 0;
};

class LandlinePhone: public Phone {
protected:
    PhoneNumber number_;
    Name subscriber_;
    RingTone rTone_;
public:
    LandlinePhone(const char *num,
        const char *subs) :
        number_(num), subscriber_(subs),
        rTone_() {}
    // Implementations for interfaces
    void Call(const PhoneNumber *p);
    void Answer();
    // Dummy implementation not for use
    void ReDial()
    { cout << "Not implemented" << endl; }
    friend ostream& operator<<(ostream& os,
        const LandlinePhone& p);
};

class MobilePhone : public LandlinePhone { protected:
    AddressBook aBook_;
    PhoneNumber *lastDial_;
    void SetLastDialed(const PhoneNumber& p);
    void ShowNumber();
public:
    MobilePhone(const char *num, const char *subs) :
        LandlinePhone(num, subs), lastDial_(0) {}
    void Call(const PhoneNumber *p); // Override
    void Call(const Name& n); // Overload
    void ReDial(); // Override
    friend ostream& operator<< (ostream& os,
        const MobilePhone& p);
};

class SmartPhone : public MobilePhone {
protected: unsigned int size_;
    void DisplayPhoto();
public:
    SmartPhone(const char *num, const char *subs) :
        MobilePhone(num, subs), lastDial_(0) {}
    void Call(const PhoneNumber *p); // Override
    void Call(const Name& n); // Override
    void ReDial(); // Override
    friend ostream& operator<< (ostream& os,
        const SmartPhone& p);
};
```

So finally, you put things together, put all of these. So, now I have erased all comments and all which are commented out things which are not needed. So, this is your landline phone, which you started with. This is your mobile phone. This is your smartphone put in a compacted form. And you have smartphone ISA mobile phone ISA landline phone implemented.

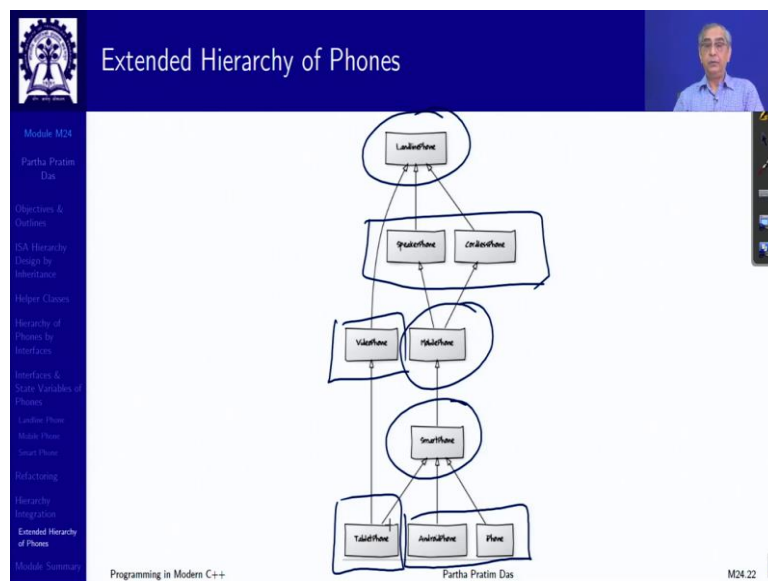
Your hierarchy is in place, now you need to put in the member function details. Now typically, we will see this more but typically when we do such a hierarchy we are specializing from a generalized concept. So, a smartphone can also have other specializations. And we have kept things in the in the protected to make sure that it can extend. Similarly, on the top is landline phone the ultimate.

No, I want to talk about an abstraction called a phone, what is a phone, where I can call it an answer I can redial, anything any device in which I can do that is a phone. So, I define a, and this is a part of the integration I define a class which I call an abstract base class that I do not expect I have not seen that device I do not expect specific device like that, but I just want to say that this is the functionality which is in my mind, a phone.

So, that abstraction is put in here there are a lot of details in terms of C++ that exists here which you have not covered yet we will explicitly cover this. But assume just know that this is kind of an abstract base class where you just have interfaces, you cannot have data members, you cannot have any object of that but tells you the concept.

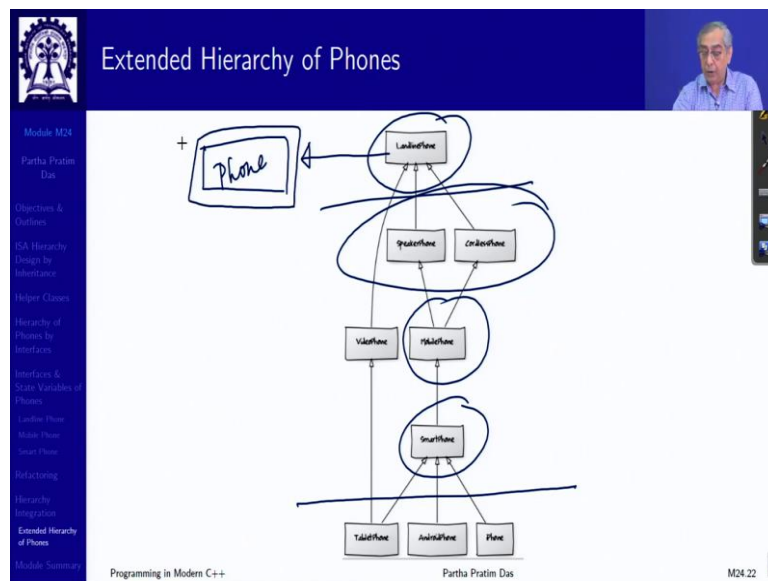
And then what I do instead of making landline phone as the root of the hierarchy I make the phone the root of the hierarchy. So, it is kind of very nice. And on the top, you have an abstract concept, then you have the hierarchy coming down specializing, specializing, specializing further and with this your entire hierarchy of phone systems is now integrated perfectly refactored and you have the entire thing in place.

(Refer Slide Time: 31:19)



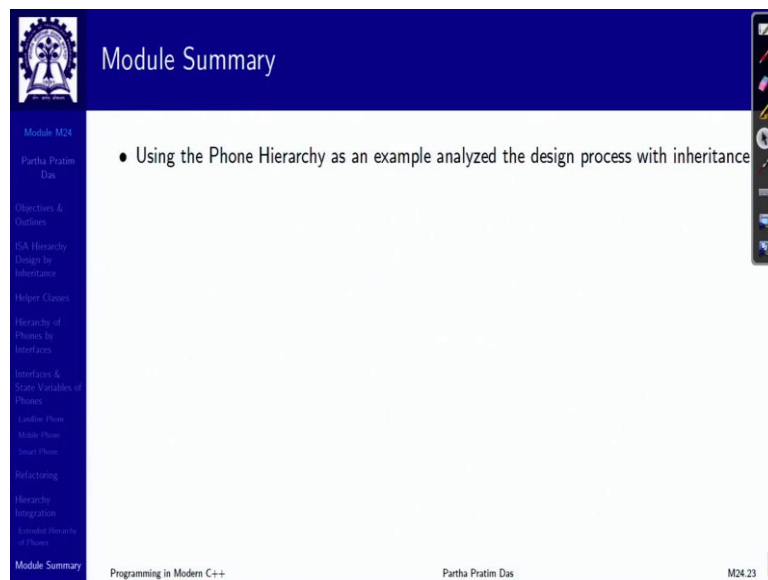
So, once you have done that, then maybe you would like to look at some of the expandability and see okay have you kept provision for that. So, what we have implemented in the hierarchy is this, this and this. So, I said it is transitive. So, mobile phone is transitively is a landline phone. But there are different other kinds of like speaker phone, cordless phones, which had between landline phone and mobile phone. There is a video phone, tablet phone, there are Android phone, there is iPhone, specialized forms of smartphones and so on.

(Refer Slide Time: 32:08)



So, just try to look at that if this is the hierarchy that I have done, I can actually put things below this, I can put things below here I can insert in the middle, all my hierarchy is open for future extension, extendibility is very important. Here I have not shown the abstraction which will come something like a phone here, if you do the complete diagram, but this is an abstract one.

(Refer Slide Time: 32:42)



So, by this we have, I have tried to show you that give for a given phone hierarchy example, how to go about doing the design process with inheritance and put everything together. We will take up more through tutorials and assignment examples to build your, but building this skill of implementation of hierarchy is very, very key to learning the C++

programming in the real strength of object orientation. Thank you very much. See you in the next module.