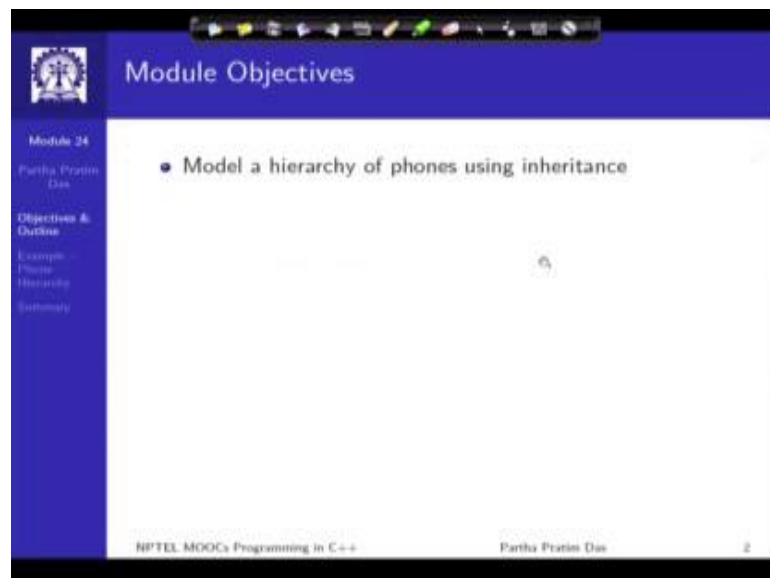


Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 39
Inheritance: Part IV

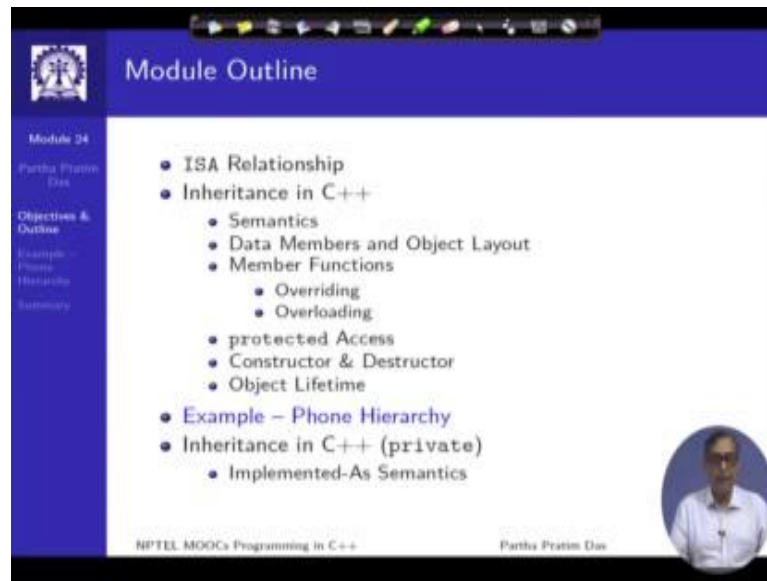
Welcome to Module 24 of Programming in C++. From the last couple of modules we have been discussing about inheritance in C++.

(Refer Slide Time: 00:38)



We have understood the basic definition and structure of inheritance and in that connection in this module we would like to try to work out an example of hierarchy of phones that we use today. We would like to show during this module that, how we can actually take the abstraction of different concepts, try to create the class modules for those in C++, organize them the resulted ISA hierarchy in terms of C++ code and create possibilities for extension.

(Refer Slide Time: 01:30)



So, in terms of the outline as I had mentioned this is the complete outline of what we are discussing in the basic level of inheritance, we have already talked about ISA relationship in modeling that OOAD frequently used and using that we have shown how to express ISA relationship in terms of two or more C++ classes, based on that we have defined detailed semantics for the inheritance of data members, the inheritance of member functions.

We have seen that when a class is derived from another base class then it inherits all the data members and member functions. And the member functions can be redefined with the same signature to overwrite them or we can introduce definitions of member functions by the existing name or inherited name with the different signature to overload that member function as we could do earlier. Further, we have seen how to add new data members, we have also taken a look into the access specification of the data and member and functions of a derived class as derived from the base class, we have introduced a new access specifier called protected, which has special semantics for derived classes. For derived classes all protected data members of the base are accessible, whereas these protected data members are not accessible for external functions and other classes.

(Refer Slide Time: 03:34)

The image is a screenshot of a presentation slide. At the top, there is a navigation bar with various icons. The slide title is "Phone Hierarchy". On the left side, there is a vertical menu with the following items: "Module 24", "Partha Pratim Das", "Objectives & Outline", "Example - Phone Hierarchy", and "Summary". The main content area contains a bulleted list:

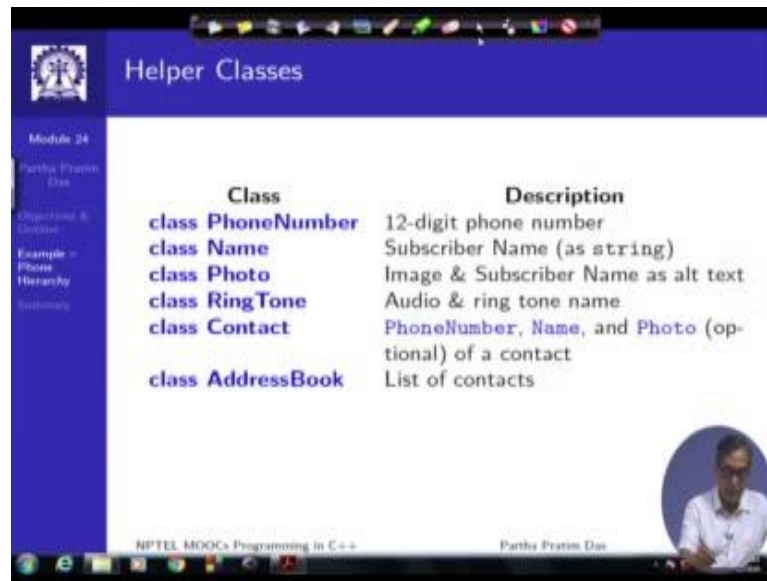
- Let us model a hierarchy of phones comprising:
 - Land line Phone
 - Mobile Phone
 - Smart Phone
- We model Helper classes
- We model each phone separately
- We model the phone hierarchy

In the bottom right corner, there is a circular portrait of a man with glasses, wearing a white shirt. At the bottom of the slide, the text "NPTEL MOOCs Programming in C++" and "Partha Pratim Das" is visible.

We have also seen the construction and destruction process and the object life time. Using all those notions we would now move into discussing a creation of a basic module structure for a set of phones. So, we start with a model hierarchy of phones and we just start simple assuming that there are three kinds of phones that make our world. The landline phones, the typical ones that we had been using maybe 20 years back, 30 years back. Then the mobile phones, the so called nowadays what is more frequently called as a dumb mobile phone, which can do only restricted functions and the all pervading variety of smartphones that has invaded our life.

So, in the process of doing that, we will first model a set of helper classes, these helper classes will allow us to have different types for the attributes that we will need to handle then we will model each phone separately. For each of these, we will try to write down an outline of a class description and then we will see that given that, these phones have certain inherent hierarchy of specialization, how do we combine this code into a C++ inheritance hierarchy and how does that simplify the total model.

(Refer Slide Time: 05:06)



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

So, to start with we first have a set of helper classes. So, I have not included the details of these classes, you can work those out. One class naturally that we need is to represent the phone number certainly and phone numbers as we know that in India, the phone numbers are 12 digit including the country code. So, it is sum type, which presents 12 digit numbers. Then certainly we need the name of the subscriber, who is actually subscribing to the phone, we need the name also for our contacts, the people whom we want to call.

We have provision for having photos of our contacts, so will have some helper class as photo which will have image and the alternate text for the image and so on. We will have a class defining ringtones, the audio file and the name of that ringtone, etcetera. We will have helper class for contact, the minimum information of a contact is a phone number and name and optionally we could have a photo of that contact also. Of course, there are several other that are possible, like the designation, like company and so on. We are just keeping it simple right now. Finally, we will have a helper class address book which is a collection of maybe set or list of contacts, which we make and maintain in our phone. So, these were the simple set of basic helper classes let us go ahead with the design.

(Refer Slide Time: 06:52)

The screenshot shows a presentation slide with a blue header and a white content area. The header contains the text 'Land line Phone Model' and a small logo on the left. The content area is divided into three sections: a left sidebar with navigation links, a central bulleted list, and a right code block. The sidebar includes 'Module 24', 'Partha Pratim Das', 'Objective 9', 'Example - Phone Hierarchy', and 'Summary'. The central list contains 'Landline Phone', 'Call: By dial / keyboard', and 'Answer'. The code block shows the C++ class definition for LandlinePhone, including data members (number, subscriber, ringtones), a constructor, two public methods (Call and Answer), and a friend operator function for overloading the call operator.

```
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, first we want to model the landline phone. When we want to model the landline phone, we identify that if I have a landline phone; however, primitive you can think about those black heavy sets with rotating dial that, we use to use several years ago. The basic functionality that a landline phone must support is an ability to call and an ability to be called back. So, we have call functionality and we have an answering functionality, without that phone is not defined. So, if I have that, then as we go on to designing the class for this phone we are trying to create this model. Suppose, I have given it a name land line phone and for the call functionality, I have introduced a call method, call member function in the class and to be able to call we need the number that we have to call. So, the phone number will be a parameter to this call member function.

Similarly, if a call arise then we have to answer that call so, there is a member function answer introduced which will answer the phone call. In terms of the data members, what do we need certainly every landline phone has a number of its own, the number at which it can be called. So, we will have a number member the phone must have been subscribed by someone so, that is a subscriber's name and we may have a ringtone, which for the very old phones are fixed ringtones of what we typically would say like crink crink sound of the phone, buzzing sound of the phone.

Based on this naturally, we need the constructor to be defined, which will at least take the name and the number and the subscriber. These two members have to be initialized

for any meaning full construction of the phone, the initialization of this ringtone we could keep optional, which say that by default it has some specific ringtone. Now, to end we have also would introduce an overloading in terms of the friend operator function for out putting the information about this particular landline phone objects. This is required more for debugging and programming trace purposes then actually the phone functionality. So, it is an auxiliary functionality, which we add in terms of a friend function to be able to work with this particular class.

(Refer Slide Time: 09:59)

The slide is titled "Mobile Phone Model" and is part of an NPTEL MOOC on Programming in C++. It features a navigation sidebar on the left with options like "Module 34", "Partha Pratim Das", "Object Oriented Programming", "Example - Phone Hierarchy", and "Summary". The main content area is divided into two columns. The left column shows a class hierarchy for "Mobile Phone" with the following features:

- Call: By keyboard – shows number
 - By Number
 - By Name
- Answer
- Redial
- Set Ring Tone
- Add Contact
 - Number
 - Name

The right column contains the C++ code for the `MobilePhone` class:

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

public:
    MobilePhone(const char *num,
               const char *name);

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

    void Answer();

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

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

At the bottom of the slide, it says "NPTEL MOOCs Programming in C++" and "Partha Pratim Das" next to a small circular portrait of the speaker.

Let us move on, let us now talk about the mobile phone. Certainly, the mobile phone will have a call functionality, will have an answer functionality, In terms of the call functionality, usually in the mobile phone, we would either be able call the number as we could do in a landline phone, but it is also possible that the mobile phone will have some kind of a address book mechanism, so that I can keep a set of contacts and I can pick up some contact by the name and I can call that contact by that name. So, in terms of call we do see, in terms of mobile phone two options, which one of which that is calling by name was not available in the landline phone.

In addition to that we have usually we have an option to redial, typically to redial the last number that was called and in a majority of mobile phones we also have the option to set

a ringtone. So, this of course there are several other that we can think off, but certainly by mobile phone as I mentioned I am talking about the cellular phones of kind of the early generations like, typically many of you may have seen and used Nokia 1100 series kind of phones where you just have ability to be mobile, but you do not have all those different fancy functionalities that we see today.

Let us see for this simple mobile phone class how do we make the definition, so we have created the class with the name mobile phone for each of the functionality that, we see they will have to be some member functions associated. So, for call we will have a member function, this looks pretty much like the call member functions we are done for the landline phone class, so which takes a phone number and makes the call, but now we see that there is additional functionality that, I can call someone by name. So, we will keep another call member function which takes the name of a person and makes the call.

So, we can see that here because of the duality of the call function, we are having to introduce overloaded member functions. We will have an answer, which is the answer member function. We introduce a redial member function to be able to redial a member function. To set the ringtone which certainly will have to take the particular ringtone that I want to set and for adding new acquaintances to my address book, I need a add contact where, I need to specify the number and I need to specify the name of the person. So, with this we have so, as we start after deciding the name of the class, these are first things that we complete which will give us the interface, as we say of what this class should be doing that is a basic functionality of the class, the set of operations for the class.

Having specified that, now we look into the internals of the classes, if I have to support these member functions, if I have to support the functionality that I want, what are the data members that I will need? Certainly so, looking into that certainly I need the number of the phone which is as before, I need a subscriber name to reach this particular mobile phone is been subscribed and being mobile phone in all likelihood, it will have the possibility of setting different ringtones so I have a ringtone members. So, these are pretty much like what we had seen before, but now we have assumed that it is possible that we keep the context in our phone. So, I need an address book. So, a book is a

member which will keep, which is an address book that is it will keep a list or set of contacts that I would like to refer to at often times.

So, if I add do add contact that basically will add the contact here and I would also need to need a feature, I have a provided an interface that, I would like to redial, just simply redial the number that I had dialed last. So, I need a member to remember, what is the last number that I had dialed? So, that brings in this data member, besides that I may need some of the; so these are the basic data members that I will require to support this functionality and in addition I will need some more member functions, for example, if I think about let us say redial.

(Refer Slide Time: 15:41)

The screenshot shows a presentation slide titled "Mobile Phone Model". On the left, there is a navigation menu with items: "Module 24", "Partha Pratim Das", "Object-Oriented Programming", "Example - Phone Hierarchy", and "Summary". The main content area is divided into two columns. The left column contains a class hierarchy for "Mobile Phone":

- Call: By keyboard – shows number
 - By Number
 - By Name
- Answer
- Redial
- Set Ring Tone
- Add Contact
 - Number
 - Name

Handwritten red text "Redial()" and "Call (last dial)" with arrows points to the "Redial" and "Call" items respectively. The right column contains C++ code for the MobilePhone class:

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

public:
    MobilePhone(const char *name,
               const char *number);

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

    void Redial();
    void SetRingTone(RingTone r);
    void AddContact(const char *name = 0,
                  const char *number = 0);

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

At the bottom, it says "NPTEL MOOCs Programming in C++" and "Partha Pratim Das".

So, if I think about redial, if want to then I will need to give a call to; this is the last dial. So, I will need to give a call to this member function. So, I will have to do call, redial that will be last dial and that will be my basic redial functionality. So, I need a way to set this data member, I need a way to remember this data member. So, what does it mean? It means that when I am dialing some number, I am calling some number I must remember that.

(Refer Slide Time: 16:25)

The slide is titled "Mobile Phone Model" and is part of "Module 34" by "Partha Pratim Das". It shows a class hierarchy for "Mobile Phone" with the following features:

- Call: By keyboard - shows number
 - By Number
 - By Name
- Answer
- Redial
- Set Ring Tone
- Add Contact
 - Number
 - Name

Handwritten notes in red ink on the slide include:

- "Call (PhoneNumber *p)" with an arrow pointing to the `Call` method in the code.
- "Set LastDial (P)" with an arrow pointing to the `setLastDial` method in the code.
- "LastDial = P" with an arrow pointing to the `lastDial` member variable in the code.

```
class MobilePhone {
    PhoneNumber number_;
    Name subscriber_;
    RingTone rTone_;
    AddressBook aBook_;
    PhoneNumber *lastDial_;
    void setLastDial(const PhoneNumber p);
    void showNumber();

public:
    MobilePhone(const char *num,
                const char *name);

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

    void Answer();

    void Redial();
    void SetRingTone(RingTone rTONE r);
    void AddContact(const char *num = 0,
                  const char *name = 0);

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

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

So, if I look into the call functionality of the member function, say for phone number star type then what I will need is to set this particular last dial function; last dial number. So, here I include this member function which I will invoke from my call method. Similarly, when I implement the other call, overloaded call method also I will have invoked this, which will set the last dialed number. Now, you may wonder as to, am I setting up separate member function to do that? The other option could have been instead of doing this I could have simply done last dialed or last dial is assigned. I could have simply set this to p because it is all in the domain of all in the same name space of the mobile phone class.

The reason I perceive that I will rather have a member function to start with because it is possible that when I set this last dial number I may want to keep some more properties along with the just the last dial number, for example, I might want to also remember as to when did I dial this person the last time. I may want to also remember as to what was fate of the last call, did it go through or was it missed and if it did go through then what is a duration for which I took the call and so on. The actual functionality could be extended, extensible in several different ways. So, it may be lot more than just setting this last dial number.

So, I try to model that in terms of a member function. So, that is a typical style that we will often try to follow that, whenever we want to set some member, data member or may be at times get some data member even from within the member function of the class, we may want conceive whether we would directly set that or we will use some private member functions to set them. So that if there are additional functionalities then those functionalities can also be put through.

(Refer Slide Time: 19:21)

Mobile Phone Model

Module 04
Partha Pratim Das
Objective 8: Oop
Example - Phone Hierarchy
Summary

- Mobile Phone
 - Call: By keyboard – shows number
 - By Number
 - By Name
 - Answer
 - Redial
 - Set Ring Tone
 - Add Contact
 - Number
 - Name

```

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

public:
    MobilePhone(const char *num,
               const char *subn);

    void Call(PhoneNumber *p);
    void Call(const Base& n);
    void Answer();

    void Redial();
    void SetRingTone(RingTone rRINGTONE r);
    void AddContact(const char *num = 0,
                  const char *subn = 0);

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

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, Naturally if we perceive it that way then we need this member function to be present in the mobile phone class and the question is this member function should it be in public, answer is no because we have already defined the interface. This is what we want the interface to be that is what others external classes and global function should be able to access because that what is a view of the phone that you have. So, if you think about a physical phone, you have a options for doing all of these interface activities on the keyboard or the graphics display, your touch screen, the phone, but do you really see any functionality where you can set the last dial number, you do not because you are not supposed to do that.

That is internal to the phone and therefore, such member functions are private and as we say this is for the interface, this member functions are for the implementation. So, you

will in the design you will need to keep this distinction clearly in mind. Similar to that I may have another member function as show number that when the dialing has happened when you are calling someone you might want to show the number that your calling or when an call has come the phone ringing and you want to answer you would like to see the number that is calling you.

(Refer Slide Time: 20:59)

The slide is titled "Mobile Phone Model" and is part of "Module 24" on "Partha Pratim Das". It contains a table of contents on the left, a class diagram, and C++ code for a MobilePhone class. The class diagram shows a MobilePhone class with methods: Call (By keyboard - shows number, By Number, By Name), Answer, Redial, Set Ring Tone, and Add Contact (Number, Name). The C++ code defines the MobilePhone class with private members: PhoneNumber number, bool subscribed, RingTone *tone, AddressBook *book, and PhoneNumber *lastDial. Public methods include: MobilePhone(const char *num, const char *name), Call(PhoneNumber *p), Call(const Name *n), Answer(), Redial(), SetRingTone(RingTone *tone), AddContact(const char *num, const char *name), and a friend ostream operator. Handwritten annotations in red ink show: "Call (const Name * n)", "AddressBook * book", "Call (p)", and "Call (p), n".

So, the show number is a member function which is supposed to do all this behavior. There will be several such additional member functions that you would need, but I just, I am illustrating two such to explain to you that there could be several member functions in the private part of the class as well, which are basically supporting functions for realizing the interface and other supporting functions, which you do not want to make feasible in the public space. Similarly, we may talk a little bit about this call, which basically call a person by name.

So, if I have this then what are the functionalities that we will need? Certainly, we need to, I am sorry, this should be n. So, I need to actually know the number this person has. So, I would assume that, why do I get the number certainly the number has to come from the address book. So, I would have to assume that address book will have some kind of such member function such method which given a name finds out and returns me the

phone number. So, I can have the phone number return in terms of that and once that have been returned then I can use the other interface function which can make a call based on a phone number.

We can use that to realize the actual functionality of this call because if you look between these two overloaded member functions, the basic functionality is the call which is realized by the first one and the added to your functionality is search person you want to call and then make a call that is what is realize by the second function here. So, this is where you write the code to search and then you actually make the call and that is how you should go ahead with doing design and as you go ahead your finding that I said that I will not give you the details of the helper classes because their interfaces, their member functions will kind of get derived from your design because we have just seen this requirement for an address book. We have seen the requirement to adding a contact the address book and so on. So, all this will derive the different interface methods that the address book class should have. By similar reasoning you should be able to find the different interface requirements of the other helper classes as well. So, we have fairly detail description of the mobile phone class.

(Refer Slide Time: 24:05)

Smart Phone Model

- 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_;
    Base subscriber_;
    RingTone rTone_;
    AddressBook aBook_;
    PhoneNumber *lastDial_;
    void setLastDial(const PhoneNumber p);
    void ShowNumber();
    unsigned int size_;
    void DisplayPhoto();

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

    void Call(PhoneNumber np);
    void Call(const Base& n);

    void Answer();

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

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

NPTEL MOOCs Programming In C++ Partha Pratim Das

So, let us move on to the smart phone with respect to the mobile phone, I kept the smart phone relatively simple; it can call by touch screen. The smart phones typically are characterized by having a touchscreen. So, earlier I was calling from the mobile phone with a keypad. Now, I am calling with a touch screen, the basic functionality remains the same. So, I am now showing the design requirements and the design of the class together. So, the basic requirement remains the same that I call a number or I call a name, but I would again need to have both of these, these are overloaded, but what you should now start realizing that these are situations of possibilities of overriding that is arising here because I have the same functionality of been able to call a phone number, but in that functionality what is changing is the implementation of this function.

Earlier it was by keypad, now it is by the touch screen. I have answer, redial these are all like what we discuss for mobile phone. In terms of add contact, now it thus has a touchscreen smart display. So, I will need to probably would like to have photos of my contacts also to be added. So, the interface looks pretty much similar though it may need reimplementations, these data members are the same as what we did in mobile phone. These methods, private methods are also same, but I may have additional data members for example, if it is a smart phone then one major criterion of finding a smart phone is a size of the display, is it a 7 inch phone, it is a six and half inch phone and so on.

So, I might want to put the size, when I am making a call or I am answering a call I would like to see the face of the person whom I am calling or who is calling me provided that person is in my address book. So, possibly I will have a functionality like display photo as a private member function in smart phone, there could be several others as well. So, these are basic class descriptions of the landline, mobile and smart phones.

(Refer Slide Time: 26:43)

Comparison of 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 9

So, this is the summary of their different functionality and as we had seen before that there is a strong sense of generalization specialization that exists amongst these concepts among these classes.

(Refer Slide Time: 26:59)

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**

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

So, we can quickly conclude that here we have mobile phone is a landline phone and smart phone is a mobile phone and with that now, we can look at the total functionality.

(Refer Slide Time: 27:15)

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

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

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

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 *sub):
        number_(num), subscriber_(sub) {}
    void Call(PhoneNumber &p);
    void Call(const Name& n);
    void Redial();
    void Answer();
    void SetRingTone(RingTone::RINGTONE r);
    void AddContact(const char *num = 0,
                   const char *sub = 0);

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

That, if we now place that design of the landline phone class and the mobile phone class and look at them side by side then we see that there are several data member which have basically common, but there are others which are new. Similarly, there are some methods which are common, there some methods need new signature and there several other methods which are new. So, with this observation we can plan actually to combine them in terms of a hierarchy and reduce minimize the design that we have.

(Refer Slide Time: 27:59)

These are just to show you in terms of how does it look in terms of the UML model? This is for your further understanding.

(Refer Slide Time: 28:05)

The slide displays two C++ class definitions side-by-side. The left class is 'LandlinePhone' and the right is 'MobilePhone'. 'MobilePhone' inherits from 'LandlinePhone'. The slide also includes a navigation bar at the top, a sidebar on the left with 'Module 34' and 'Phone Hierarchy', and a small video inset of a speaker in the bottom right corner.

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

public:
    LandlinePhone(const char *num,
                  const char *suba) :
        number_(num), subscriber_(suba),
        rTone_(0) {}

    void Call(const PhoneNumber *p);
    void Answer();

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

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

public:
    MobilePhone(const char *num,
                const char *suba) :
        LandlinePhone(num, suba), // Base ctor
        lastDial_(0) {}

    void Call(const PhoneNumber *p); // Override
    void Call(const Base& b); // Overload
    //void Answer();
    void Redial();
    void SetRingTone(RingTone r);
    void AddContact(const char *num = 0,
                  const char *suba = 0);

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

But now, I can just use this observation to model not just the mobile phone and the landline phone separately, but I can model as mobile phone is a landline phone. So, I introduce the inheritance here, the moment I introduce inheritance here the need for these data members do not exist because they exist in the base class and will automatically get inherited, these members certainly are added. Similarly, when I do this I if I perceive as we have discussed that we did not have any specific difference in terms of way the answering is done, we just pick up the call and start talking.

So, I might perceive that the way you answer in a landline phone and the way you answer in a mobile phone are the same. So, this commented out which means that this particular function will get inherited from the landline phone, the parent class, but in terms of the call we need to still define this signature that we need to over write because the way to call would be very different, for example, a landline phone I may just be using dial in a mobile. I am using a certainly I am not using a dial I am certainly using some kind of a keypad keyboard and further I need another version of the call function as we I has mentioned and which will mean that I have a overload here. So, with this the design of the mobile phone class gets further simplified.

(Refer Slide Time: 29:46)

```
SmartPhone ISA MobilePhone

Module 34
Partha Pratim Das
Object-Oriented Programming
Example - Phone Hierarchy
Summary

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

public:
    MobilePhone(const char *num,
                const char *sub):
        LandlinePhone(num, sub), // Base ctor
        lastDial_(0) {}

    void Call(const PhoneNumber& *p); // Override
    void Call(const Name& n); // Override
    //void Answer();
    void Redial();
    void SetRingTone(RingTone r);
    void AddContact(const char *num = 0,
                  const char *sub = 0);

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

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

public:
    SmartPhone(const char *num,
               const char *sub):
        MobilePhone(num, sub), // Base ctor
        lastDial_(0) {}

    void Call(const PhoneNumber& *p); // Override
    void Call(const Name& n); // Override
    //void Answer();
    void Redial(); // Override
    //void SetRingTone(RingTone r);
    //void AddContact(const char *num = 0,
    //                const char *sub = 0);

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

NPTEL MOOC's Programming in C++ Partha Pratim Das 14
```

We can use that to go further and relate this now with the smart phone. I am trying to model that smart phone is a mobile phone. So, smart phone is a mobile phone I make the inheritance, these were earlier commented out because they were common from the landline phone. In the smart phone, these members are also not required. These data members and methods and also not required because they will get inherited from the mobile phone, but I need to add the data member and member function that are specific to smart phone.

Coming to these naturally answer gets inherited, but call both of these versions of overloaded call functions in mobile phone need to be again over read. Now, we can see that this was over written and this is again being over written because the way you call through a touchscreen is quite different this is also. So, here in terms of the mobile this was the overloaded in terms of the smart phone. This is now over written from the definition that you had used in the mobile phone, you are not changing the interface any more, but you will certainly have a different implementation and all of these additional functions are certainly inherited. Of course redial also need to be over written because if your call is over written redial is just another version of the call function

(Refer Slide Time: 31:23)



```
class Phone {
public:
    virtual void Call(const PhoneNumber &p) = 0;
    virtual void Answer() = 0;
    virtual void Redial() = 0;
};

class LandlinePhone: public Phone {
public:
    LandlinePhone(const char *num,
                  const char *subn,
                  const char *rTune,
                  const char *aTune,
                  const char *s):
        number_(num), subscriber_(subn),
        rTune_(rTune), aTune_(aTune),
        s(s) {}
    void Call(const PhoneNumber &p);
    void Answer();
    friend ostream& operator<<(ostream& os,
                               const LandlinePhone& p);
};

class MobilePhone: public LandlinePhone {
protected:
    AddressBook aBook_;
    PhoneNumber *lastDial_;
    void SetLastDial(const PhoneNumber& p);
    void ShowNumber();
public:
    MobilePhone(const char *num,
                const char *subn) :
        LandlinePhone(num, subn), // Base ctor
        lastDial_(0) {}
    void Call(const PhoneNumber &p); // Override
    void Call(const Number& n); // Override
    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 *subn) :
        MobilePhone(num, subn), // Base ctor
        lastDial_(0) {}
    void Call(const PhoneNumber &p); // Override
    void Call(const Number& n); // Override
    void Redial(); // Override
    friend ostream& operator<<(ostream& os,
                               const SmartPhone& p);
};
```

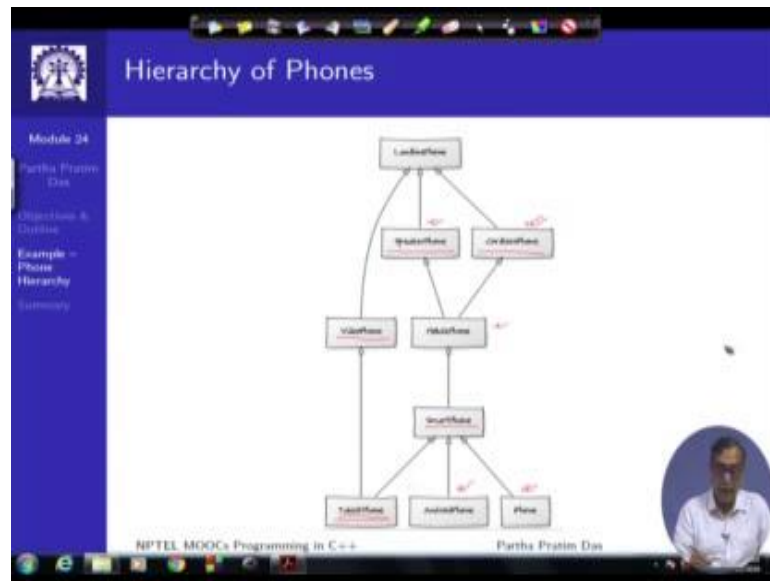
NPTEL MOOCs Programming in C++

Partha Pratim Das 15

So, with this put together, now we have if I combine them that I we have a landline phone, we have a mobile phone which is landline phone specialization. We have a smart phone which is a specialization of the mobile phone and this is how the class interface would look like and what I just out line here, I will not discuss it here now because we have not at ready with all the C++features to discuss this is. If we have a hierarchy like this then how about generalizing this further and just look at a concept of a phone.

So, the basic concept of a phone is I can call, I can answer, and I can redial. So, can I think about an abstract phone, which is the generalization of all kinds of phone. So, I say landline phone is a phone which is an abstraction of all kinds of phone. Then a mobile phone is a landline phone and smart phone is a mobile phone and so on and there is lot of advantages of being able to define such abstraction at the root of your hierarchy which will be the topic of our discussions in the modules may be talk about polymorphism, but this is what shows the you can create complete phone hierarchy and just for your I mean if still, if it looks very straight it is multi level simple hierarchy.

(Refer Slide Time: 32:45)



Then would like to draw your attention to something little bit more realistic in terms of what the phones look like. So, you have a landline phone, which may be cordless hand set phone or it could be speaker phone. When you have a mobile phone then it is cordless phone as well as a speaker phone because you can use it for both on the other side there are video phones which are landed these days. So, where you could actually make video calls and then you have smart phones as specialization of mobile phones that we have seen, but they are again specialized with whether they are I-phone or an android phone or I could have some kind of a tablet phone which is smart phone as well as can be used as a video phone and so on. So, I just suggest that based on these hierarchy we could try at home and try to built the similar set of C++inheritance classes which can represent this hierarchy

(Refer Slide Time: 33:45)

Module Summary

- Using the Phone Hierarchy as an example analyzed the design process with inheritance

NPTEL MOOCs Programming in C++ Partha Pratim Das 17

So, to summarize we have use the phone hierarchy here to show, how inheritance can be used to create effective C++code models for a realistic situations.