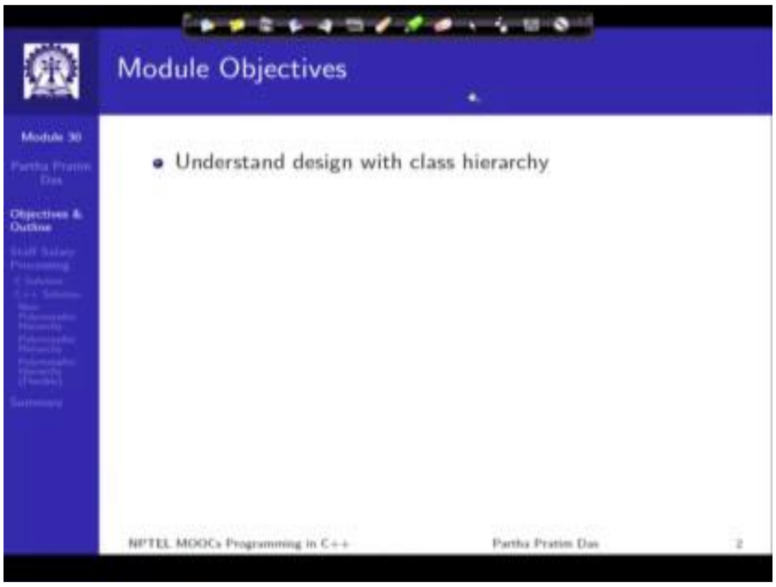
Programming in C++

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

Lecture – 45
Dynamic Binding (Polymorphism): Part V

Welcome to Module 30 of Programming in C++. We have been discussing about Polymorphism in C++, after introducing the various features of polymorphism from the last module we have started working on solving a specific problem and we want to explode how polymorphism could be effective in that solution of the problem.

(Refer Slide Time: 00:45)



Module Objectives

- Understand design with class hierarchy

Module 30

Partha Pratim Das

Objectives & Outline

Staff Salary Processing

C Solution

C++ Solution

Non-Polymorphic Hierarchy

Polymorphic Hierarchy

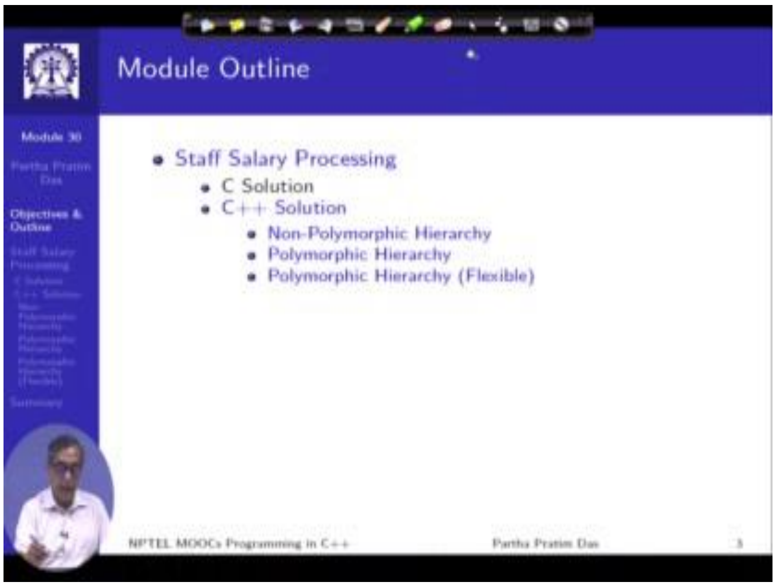
Polymorphic Hierarchy (Flexible)

Summary

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

So the objective is to continue to understand the designing with class hierarchy.

(Refer Slide Time: 00:51)



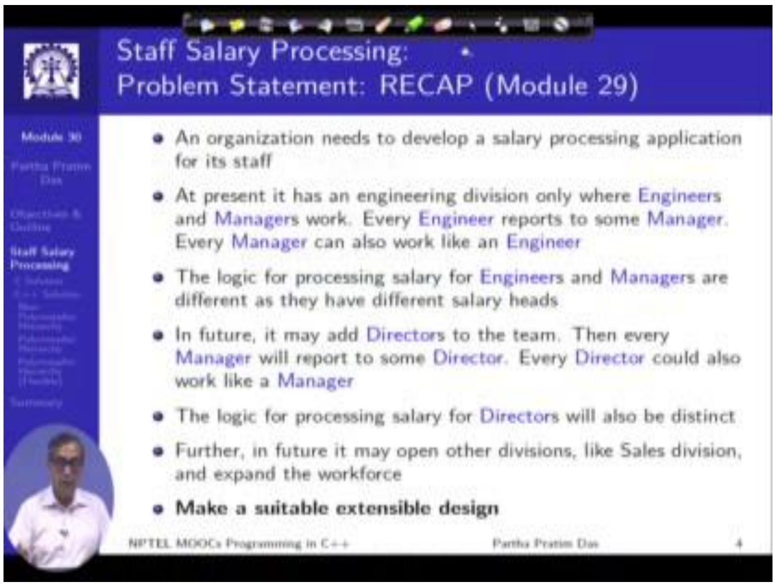
Module Outline

- Staff Salary Processing
  - C Solution
  - C++ Solution
    - Non-Polymorphic Hierarchy
    - Polymorphic Hierarchy
    - Polymorphic Hierarchy (Flexible)

NPTEL MOOCs Programming in C++ Partha Pratim Das 3

And the specific outline would be we had discuss the C solution in the last module we will discuss a C++ solution, and we will discuss free versions of the C++ solutions one after the other and you will see that on the left hand side of your slides.

(Refer Slide Time: 01:08)



Staff Salary Processing: Problem Statement: RECAP (Module 29)

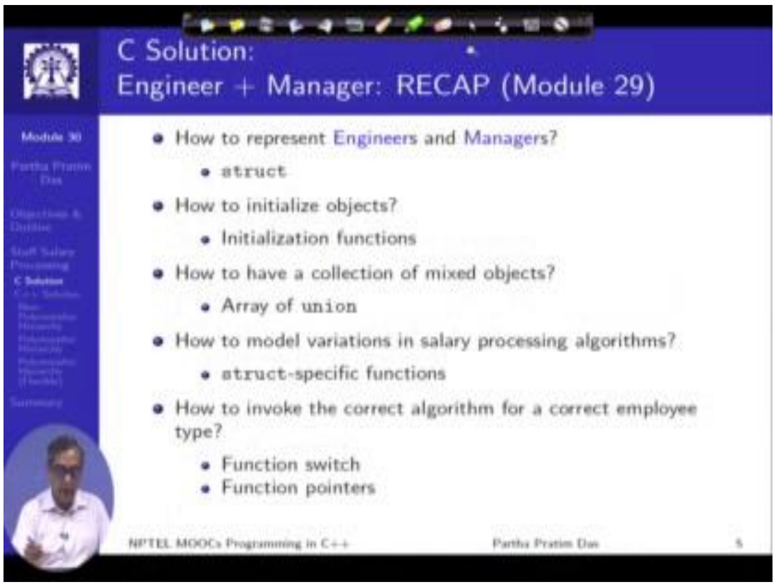
- An organization needs to develop a salary processing application for its staff
- At present it has an engineering division only where Engineers and Managers work. Every Engineer reports to some Manager. Every Manager can also work like an Engineer
- The logic for processing salary for Engineers and Managers are different as they have different salary heads
- In future, it may add Directors to the team. Then every Manager will report to some Director. Every Director could also work like a Manager
- The logic for processing salary for Directors will also be distinct
- Further, in future it may open other divisions, like Sales division, and expand the workforce
- **Make a suitable extensible design**

NPTEL MOOCs Programming in C++ Partha Pratim Das 4

Just a quickly recap, this is the staff salary processing problem and organization needs to process the salary they have one division with engineers and managers, managers can also behave as engineers, there a separate lo logic for processing salary for engineer as well as manager. In future the company intends to at the position of directors in the same division who will be able to also work as managers and will have a separate a salary processing logic for them which is different from the engineers as well as manager. And further down in future the company wants to keep it open that they can add other divisions like the sales division and have all whole lot of different kinds of staff there.

So, in view of this the whole challenge is to how we make a suitable extensible flexible design for the salary processing.

(Refer Slide Time: 02:04)



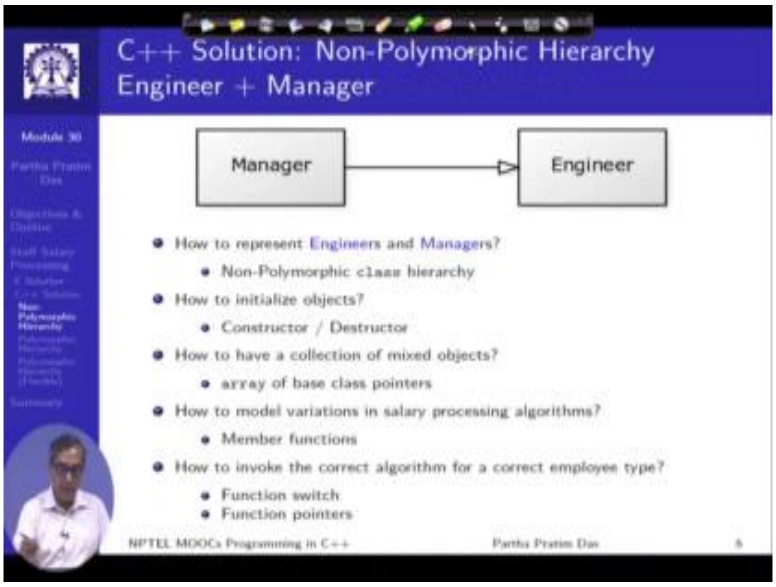
C Solution: Engineer + Manager: RECAP (Module 29)

- How to represent Engineers and Managers?
    - struct
- How to initialize objects?
    - Initialization functions
- How to have a collection of mixed objects?
    - Array of union
- How to model variations in salary processing algorithms?
    - struct-specific functions
- How to invoke the correct algorithm for a correct employee type?
    - Function switch
    - Function pointers

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

We started off with a C solution, and to attack the solution we identified some code questions that need to be unset to make a successful design and these are the C answers and we have already seen the solution in terms of the C answers. We have seen how the code will look like if I have just the engineer and manager and how what changes will be required when I add the directors and we saw different (Refer Time: 02:33) in terms of the (Refer Time: 02:34).

(Refer Slide Time: 02:33)



C++ Solution: Non-Polymorphic Hierarchy
Engineer + Manager

- How to represent Engineers and Managers?
  - Non-Polymorphic class hierarchy
- How to initialize objects?
  - Constructor / Destructor
- How to have a collection of mixed objects?
  - array of base class pointers
- How to model variations in salary processing algorithms?
  - Member functions
- How to invoke the correct algorithm for a correct employee type?
  - Function switch
  - Function pointers

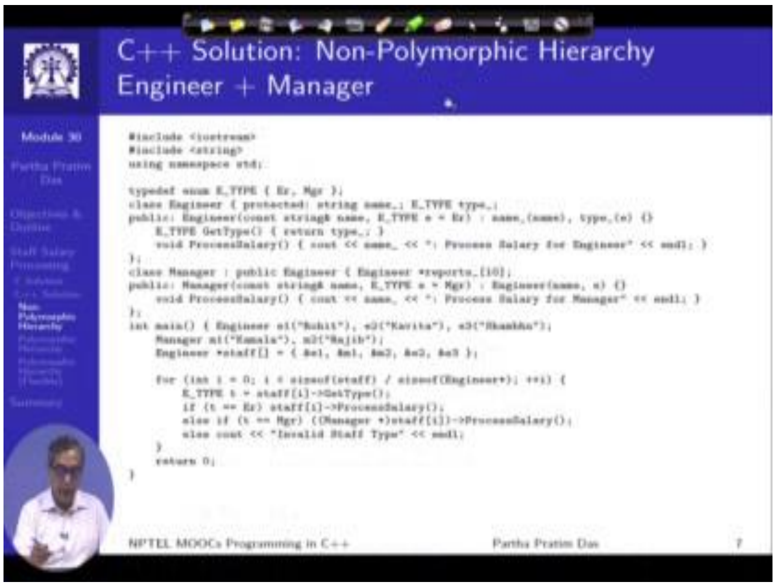
NPTEL MOOCs Programming in C++ Partha Pratim Das 6

Now move on to C++ solution and, we start by first modeling that on the fact that the company has said a manager can behave as a engineer as well. So, based on that we first start by making a model saying that a manager is a engineer, so that is a hierarchy beginning of the hierarchy. Since we have just starting off we do not how it will take shaped. Earlier just extract we are now saying that it is a non-polymorphic class hierarchy, this is a basic class hierarchy which has to be there. But the movement we come to C++ several solutions look different, the initialization etcetera become constructor and destructor. We do not need that for the collective container some problems are automatically getting solved because we have come to C++.

We do not for example need the rapper of union and the collection, I mean array of unions in terms of keeping the collection of next objects we can just use the. Since if this is this is the base class because manager is an engineer, so we can just keep and array of the pointers to base class and keep the objects their and what will be using in this is a future that the derive plus pointer can be cast as a base class pointer without any loss of information in terms of the (Refer Time: 04:05). So, this becomes are container now.

Certainly we had differently named functions for different structure types, structure put specific functions. Now in the class we will have member functions to do the job, and in terms of our dispatch we will continue to use the functions switch or the function pointers. This is our basic solution premise that we would like to see.

(Refer Slide Time: 04:31)



C++ Solution: Non-Polymorphic Hierarchy
Engineer + Manager

```
#include <iostream>
#include <string>
using namespace std;

typedef enum E_TYPE { Er, Mgr };
class Engineer { protected: string name_; E_TYPE type_;
public: Engineer(const string& name, E_TYPE e = Er) : name_(name), type_(e) {}
    E_TYPE GetType() { return type_; }
    void ProcessSalary() { cout << name_ << ": Process Salary for Engineer" << endl; }
};
class Manager : public Engineer { Engineer *reports_[10];
public: Manager(const string& name, E_TYPE e = Mgr) : Engineer(name, e) {}
    void ProcessSalary() { cout << name_ << ": Process Salary for Manager" << endl; }
};
int main() { Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib");
    Engineer *staff[] = { &e1, &m1, &m2, &e2, &e3 };

    for (int i = 0; i < sizeof(staff) / sizeof(Engineer*); ++i) {
        E_TYPE t = staff[i]->GetType();
        if (t == Er) staff[i]->ProcessSalary();
        else if (t == Mgr) ((Manager *)staff[i])->ProcessSalary();
        else cout << "Invalid Staff Type" << endl;
    }
    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

Let me just quickly take you through what the code could look like. So, we continue to have this we have a numerated type to remember the whether have an engineer or a manager will see why this is still required. What we change significantly is, we change from the structure to the class. So now you have a class, certainly the name data member becomes a protected member here, I have a type member which remembers that this is an engineer. Then I have a constructor which does a job of the initialization and it takes this name and type and sets it in this two variables.

Now, you will have to see that earlier in the C solution the structure did not have an e typed variable, because you had maintaining that outside of the objects in terms of the part of the structure with the union that we created in the collection. Now we are getting read of that so it has to be maintained within the object itself, which is kind of funny, which is kind of not a not a very healthy design because or the fact that it is an engineer object should be able to tell me that actually that the type is Er, but for the kind of non-polymorphic hierarchy we will possibly need this information. Certainly, besides the constructor there is a way to get this type information and there is a method to process the salary.

We are specialize from here, we go to the manager we has specialization of engineer we add new data members so that managers can actually keep the recording reporting information. The constructor for initialization the constructor has to use the constructor of engineer to actually initialize name. By this simple process of bringing it in terms of a hierarchy we are getting read of lot of code duplication that we had earlier in every structure we needed to have a string name. Now we just have that in the base and through the initialization of the derive class invoking the constructor of the base class we can set this name directly from here. So here, it comes to this and then is set to the actual base class part.

We have also been able to get read of differently named functions for different structures, because earlier it was process salary engineer process salary manager now you have process salary all that is happening is between these two there is a in returns in over writing, so we over write the process salary and now proved the logic of manager salary here.

So, tell these point things are pretty good and this is where we construct the objects because we have explicit constructor so we are just constructing the objective. We could have a dynamically constructed it also that does not add any special value. Then we say that we have an array of base pointers, so staff is a array of base pointers which is engineer and we put all these pointers there in the same order that we have done earlier. You just put the pointers to these objects. So this array holds whole the set of pointers, and so I can go for this.

Now, I come to the interesting part of how my application looks. My application again I have to iterate over this array which is the size, but my application again first has to know what is that type. Because you may if I access an array element, if I access staff i then I get an engineer pointer because that is a static type. But in actuality it could be a engineer object or it could be a manager object I do not know what it is, but unless I know that I do not know whether this function should be called or this function should be called I have no idea. So I have to decide on that, I have to decide on which function I would need to call.

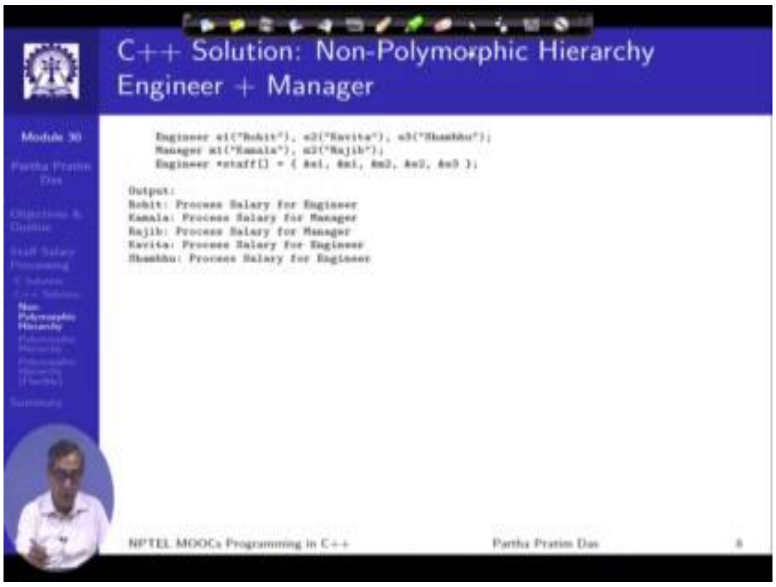
This is where the type information will be come handy. So, I take the type information I access it though they get type method in the base class the engineer class so this i get the type and then I compare whether it is. So, I am basically doing a function switch here, switch function as I did in case of C here, and if that does not match I will check for manager and so on. Once it matches, suppose it matches this then I can directly equal this because I know staff i is of type engineer. If this matches as e r then the type of object that I have is an engineer object, so I can directly invoke the process salary of the engineer class. But if it does not match then I go and check if it is a manager. If it is a manager so whether I have a manager objects which is what I will have if I go to the next one.

But now if this matches then I need to invoke this function, so how do I invoke this function? I need to cast the staff i into, I actually know by having access this information from the type I actually know that though the pointer is of engineer type i actually have a manager type. So, I take this pointer of which is engineer type and cast it to the manager type, I am trying to do something which is dangerous. Because what am I doing? I am doing a this is a engineer, this is manager, staff is a pointer here, staff i is a pointer here, pointed to this I am just trying to bring it down to be a pointer to manager. I am doing a down cast and I have done that cast in terms of the way C allows me to do the cast which is the forced cast.

Just recall if this has become hazy then I would suggest that you go back to our initial discussion on the casting and you will see that down cast can be forced in this way. So I force this. How could I confidently do that? I could confidently do that because I am managing that type. I know because when manager was created then certainly this type was passed on. So, it went here then it went here then it got set in the type field. So I know that it is manager type therefore I cast it to the manager type. Once I cast this pointer to be a manager type and then invoke process salary then certainly it invokes the process salary member function of the manager class.

Basically, this in this process though some of the issues that existed in the C designed has been removed but still the dispatch process remains to be quit vulnerable; dispatch process depends on being able to manage this type so well, and it is further problematic because if I have to add one more type of employee then several code will have to change and particularly disturbingly the application code has to change.

(Refer Slide Time: 11:57)



C++ Solution: Non-Polymorphic Hierarchy Engineer + Manager

```
    Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib");
    Engineer *staff[] = { &e1, &m1, &m2, &e2, &e3 };

Output:
Rohit: Process Salary for Engineer
Kamala: Process Salary for Manager
Rajib: Process Salary for Manager
Kavita: Process Salary for Engineer
Shambhu: Process Salary for Engineer
```

Module 30

Partha Pratim Das

Objectives & Outline

Staff Salary Processing

C Solution

C++ Solution

Non-Polymorphic Hierarchy

Polymorphic Hierarchy

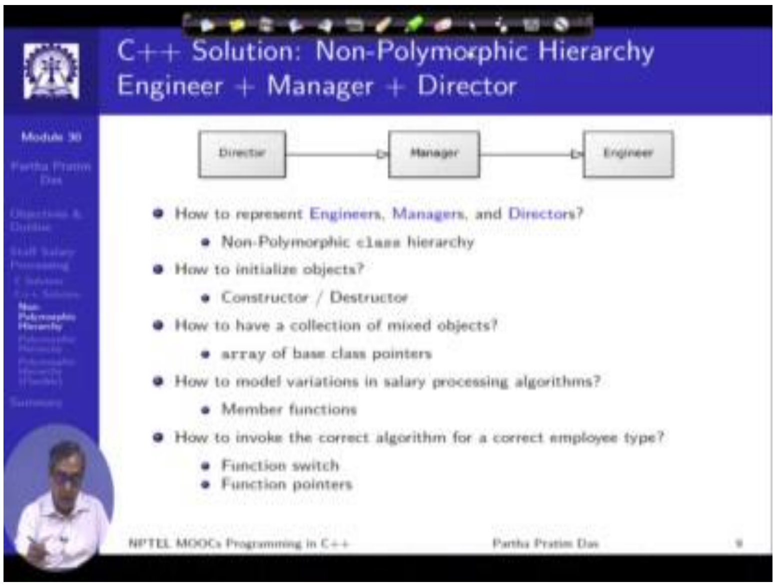
Polymorphic Hierarchy (Flexible)

Summary

NPTEL MOOCs Programming in C++ Partha Pratim Das 8

So let us take a look, this is the output you can check later on that output is correct.

(Refer Slide Time: 12:02)



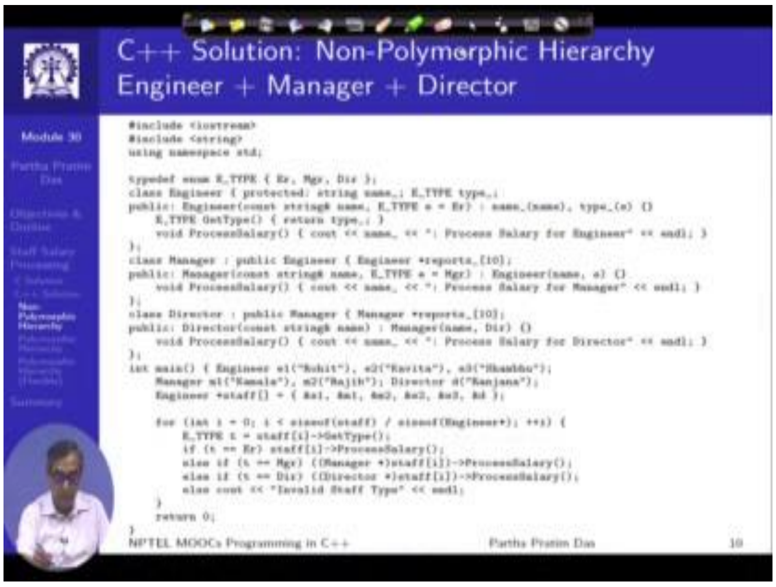
C++ Solution: Non-Polymorphic Hierarchy Engineer + Manager + Director

- How to represent Engineers, Managers, and Directors?
  - Non-Polymorphic class hierarchy
- How to initialize objects?
  - Constructor / Destructor
- How to have a collection of mixed objects?
  - array of base class pointers
- How to model variations in salary processing algorithms?
  - Member functions
- How to invoke the correct algorithm for a correct employee type?
  - Function switch
  - Function pointers

NPTEL MOOCs Programming in C++ Partha Pratim Das 9

So, let us move on and let us try to add the director the first step in the future. Director is a manager which we have got the information. So you continue to use the non-polymorphic class hierarchy, rest of the design do not change so it just that the hierarchy has to extend.

(Refer Slide Time: 12:20)



C++ Solution: Non-Polymorphic Hierarchy Engineer + Manager + Director

```
#include <iostream>
#include <string>
using namespace std;

typedef enum E_TYPE { Er, Mgr, Dir };
class Engineer { protected: string name_; E_TYPE type_;
public: Engineer(const string& name, E_TYPE e = Er) : name_(name), type_(e) {}
    E_TYPE GetType() { return type_; }
    void ProcessSalary() { cout << name_ << ": Process Salary for Engineer" << endl; }
};
class Manager : public Engineer { Engineer *reports_[10];
public: Manager(const string& name, E_TYPE e = Mgr) : Engineer(name, e) {}
    void ProcessSalary() { cout << name_ << ": Process Salary for Manager" << endl; }
};
class Director : public Manager { Manager *reports_[10];
public: Director(const string& name) : Manager(name, Dir) {}
    void ProcessSalary() { cout << name_ << ": Process Salary for Director" << endl; }
};
int main() { Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib"); Director d("Ranjana");
    Engineer *staff[] = { &e1, &m1, &m2, &e2, &e3, &d };

    for (int i = 0; i < sizeof(staff) / sizeof(Engineer*); ++i) {
        E_TYPE t = staff[i]->GetType();
        if (t == Er) staff[i]->ProcessSalary();
        else if (t == Mgr) ((Manager *)staff[i])->ProcessSalary();
        else if (t == Dir) ((Director *)staff[i])->ProcessSalary();
        else cout << "Invalid Staff Type" << endl;
    }
    return 0;
}
```

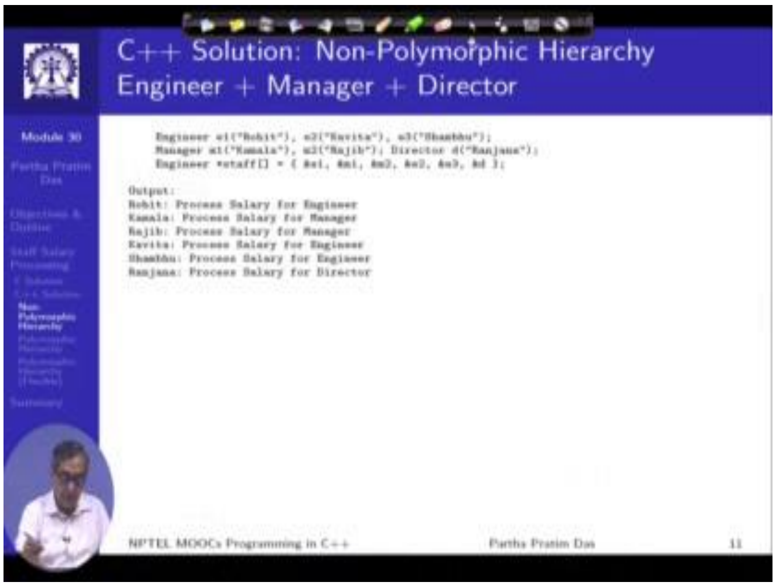
NPTEL MOOCs Programming in C++ Partha Pratim Das 10

If we do that then in terms of the type I still need to add the director type and in terms of here I need to have a director class which is a specialization of the manager, it may have another field which gives the reporting managers and so on. And when a director is constructed then you actually put a name and invoke the manager constructor and pass this director type here. If the process salary is again over written for the director and now you have the logic for processing director's salary. Rest of it remains same in the application we have instantiated a director we have added that the director to the array. But if you look into the actual application code here, then you see that earlier we had these two lines because we had only engineer and manager now you have a director.

So if my type t fails to match Er and Mgr I have to then check for whether it matches the Dir. If it matches the Dir then again I will have to do something risky I have to take this staff pointer which is engineer pointer and forcibly cast it to a director pointer so that this whole thing now becomes a pointer to director. And I invoke the process salary which will invoke the process salary for the director.

You can see that like it was in the case of C it is possible to keep on extending adding newer and newer types of employees, but in terms of quiet a bit of cost, in terms of quiet a bit if vulnerability and possible error because I have to manage the type explicitly by myself I have to propagate that properly and I have to every time I add a type my application code has to change. So just thing of that if I have tens and hundreds of different types coming in then it is how difficult and how combers some this is going to be. Let us see of whatever C++ we have learnt whether we can have a better design.

(Refer Slide Time: 14:34)



C++ Solution: Non-Polymorphic Hierarchy Engineer + Manager + Director

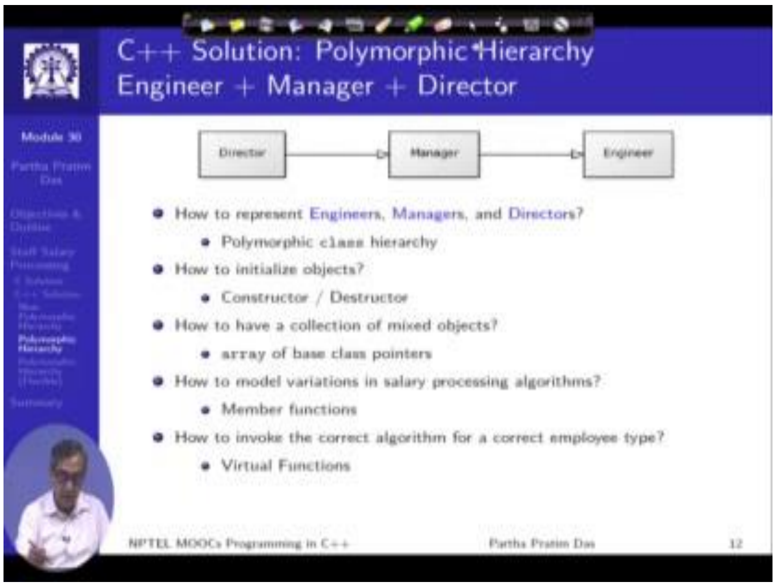
```
    Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib"); Director d("Ranjana");
    Engineer *staff[] = { &e1, &m1, &m2, &e2, &e3, &d };

Output:
Rohit: Process Salary for Engineer
Kamala: Process Salary for Manager
Rajib: Process Salary for Manager
Kavita: Process Salary for Engineer
Shambhu: Process Salary for Engineer
Ranjana: Process Salary for Director
```

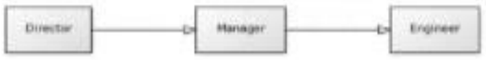
NPTEL MOOCs Programming in C++ Partha Pratim Das 11

This is output you can check later on

(Refer Slide Time: 14:38)



C++ Solution: Polymorphic Hierarchy Engineer + Manager + Director



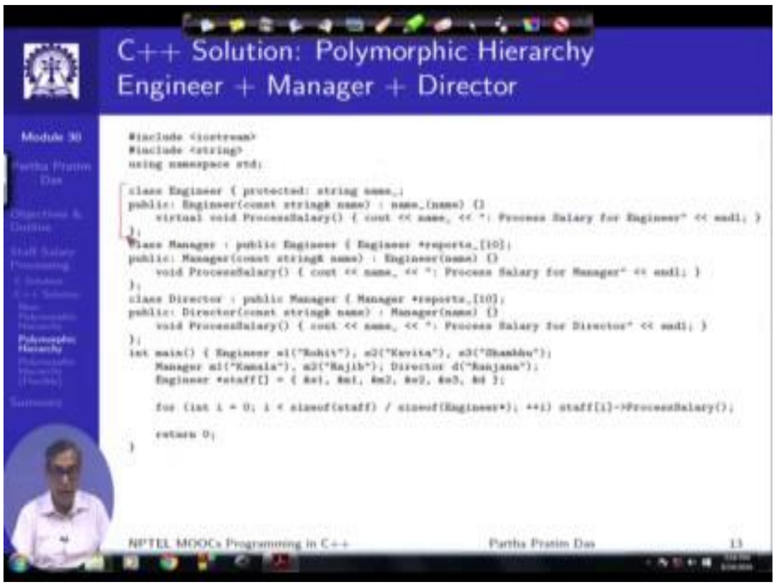
- How to represent Engineers, Managers, and Directors?
  - Polymorphic class hierarchy
- How to initialize objects?
  - Constructor / Destructor
- How to have a collection of mixed objects?
  - array of base class pointers
- How to model variations in salary processing algorithms?
  - Member functions
- How to invoke the correct algorithm for a correct employee type?
  - Virtual Functions

NPTEL MOOCs Programming in C++ Partha Pratim Das 12

Next design works with a polymorphic hierarchy. So what you change, we are again with the director, manager, engineer hierarchy, but what you change is we change from non-polymorphic to polymorphic class hierarchy.

Rest of these remains same, this remains same, this remains same, but the movement we change to polymorphic class hierarchy then our dispatch mechanism can change to virtual functions. I do not need, because this is precisely what the virtual functions are for, that if I have a base class pointer and I am pointing to a certain derive class object then I can call a function on the derive class object blindly if that function is a virtual function. So this is the precise job that the functions switch was doing which we can now realize in terms of the virtual function let us see how.

(Refer Slide Time: 15:29)



C++ Solution: Polymorphic Hierarchy Engineer + Manager + Director

```
#include <iostream>
#include <string>
using namespace std;

class Engineer { protected: string name_;
public: Engineer(const string& name) : name_(name) {}
    virtual void ProcessSalary() { cout << name_ << ": Process Salary for Engineer" << endl; }
};
class Manager : public Engineer { Engineer *reports_[10];
public: Manager(const string& name) : Engineer(name) {}
    void ProcessSalary() { cout << name_ << ": Process Salary for Manager" << endl; }
};
class Director : public Manager { Manager *reports_[10];
public: Director(const string& name) : Manager(name) {}
    void ProcessSalary() { cout << name_ << ": Process Salary for Director" << endl; }
};
int main() { Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib"); Director d("Ranjana");
    Engineer *staff[] = { &e1, &m1, &m2, &e2, &e3, &d };

    for (int i = 0; i < sizeof(staff) / sizeof(Engineer*); ++i) staff[i]->ProcessSalary();

    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 13

We have the designed again this is the enumeration of the implied type is gone so is the related field in each on of this classes we just have the name, we just have the constructor setting the name the processing salary. The manager is a specialization of employee, director is a specialization of manager and so on. So certainly all those type information is gone because now I do not need to maintain enumerated type values to know what kind of object I have. The class itself will maintain that value will contain that value for me.

In terms of creating the object and in terms of you know setting the store nothing changes, but look at the application code this follow (Refer Time: 16:19) same, but this application code has become just this much. How will this work? This will work because of this dynamic bounding and dispatch mechanism of virtual functions. So what will happen? If I am going through this when I am 0 when i is 0 I have ampersand percent e 1 which is an employee pointer. What I do have on one side I have staff i, so this has a static type which is engineer star. All invocations are with that so it tells me that whenever I try to invoke I will always start looking in the base class.

So what is a function that is being invoked, which is process salary, which is a member function in the base class and that member function is virtually. What will happen? If I have i 0 then the actual pointed object at staff 0 is e 1 which is an employee object. Because this is a virtual function the call will get delegated to the type of the object pointed too, so this function will get called which is the correct one. Think about the next one when i is 1, when i is 1 it is actually a manager pointer. When I try to do this dispatch I again start here this function this virtual and I know that the actual object been pointed to is a manager object. This virtual function will get delegated because manager classes over did in this function with the logic of how to compute manager salary and this will correctly invoke the process salary of the manager.

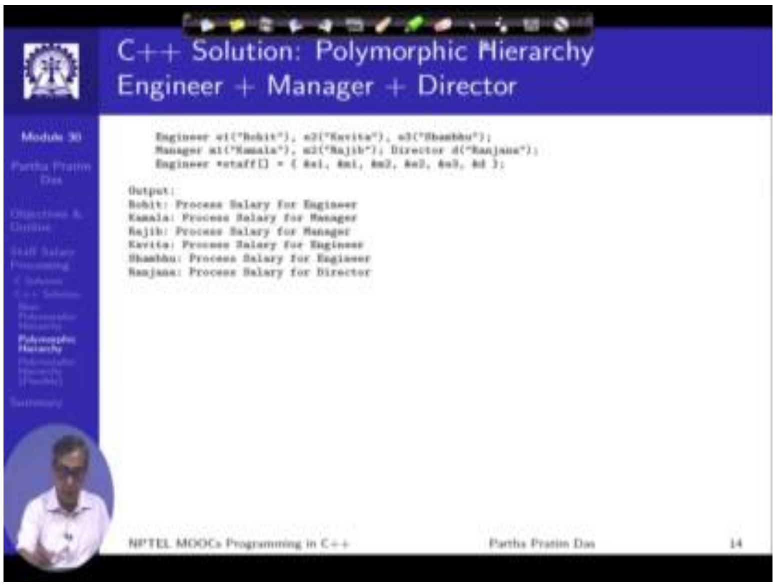
So that is the basic beauty off. The switching that we needed to do explicitly by maintaining type and then doing and if else in the application code is now all assumed in the basic feature of virtual functions where basic feature of dynamic dispatch. So that is the power of dynamic dispatch that makes the designs really better. If I continue with two this will again be same this function will get called because it is a manager object 3, this will get called because in employee object 3, this will get called because is an employee object.

If we come here with d it is a director object. So again from staff i we will come to class engineer and then in class engineer again we will see this is a virtual function. So we will try to delegate it to the type of the object that is the class director. And the class director has over read in this function. In case of this last pointer the invoke function would be the process salary in the director class which will have the logic for how to process directors salary.

We can see that in two steps first moving on to C++ which gives us a lot of benefit in terms of putting in a encapsulation then having constructor destructors to take care of in initialization, d initialization, then being able to create an array of base class pointers to create a convenient store and so on. We got one set of advantages and then when we move in the C++ design from the non-polymorphic hierarchy to polymorphic hierarchy we reap the maximum benefit in the designing terms of being able to support a dynamic dispatch which is built in to the language which is not to be coded by the application.

So you can see from the structure of this code, that here no where all that you need to know that it is of type engineer which is a base type. But if you if you add three most specializations from engineer you do not need to make any changes in this application code it can remain exactly the same you do not even need to recompile that, because you just need to recompile this class parts and link with it, because the whole dispatch mechanism is in terms of the definition of the virtual functions as they are put in the class hierarchy. So, we have made quit some progress in terms of this.

(Refer Slide Time: 20:56)



C++ Solution: Polymorphic Hierarchy Engineer + Manager + Director

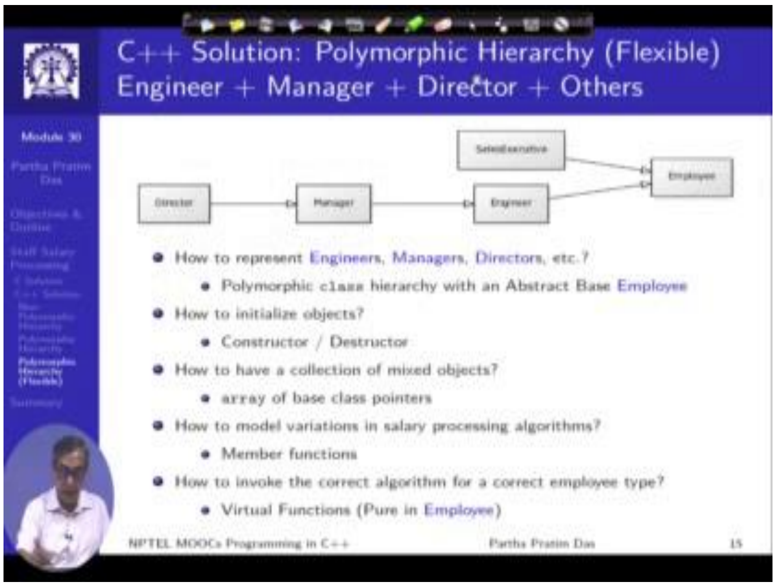
```
    Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib"); Director d("Ranjana");
    Engineer *staff[] = { &e1, &m1, &m2, &e2, &e3, &d };

Output:
Rohit: Process Salary for Engineer
Kamala: Process Salary for Manager
Rajib: Process Salary for Manager
Kavita: Process Salary for Engineer
Shambhu: Process Salary for Engineer
Ranjana: Process Salary for Director
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 14

This is the output for you to check.

(Refer Slide Time: 20:59)


C++ Solution: Polymorphic Hierarchy (Flexible) Engineer + Manager + Director + Others

- How to represent Engineers, Managers, Directors, etc.?
  - Polymorphic class hierarchy with an Abstract Base Employee
- How to initialize objects?
  - Constructor / Destructor
- How to have a collection of mixed objects?
  - array of base class pointers
- How to model variations in salary processing algorithms?
  - Member functions
- How to invoke the correct algorithm for a correct employee type?
  - Virtual Functions (Pure in Employee)

NPTEL MOOCs Programming in C++ Partha Pratim Das 15

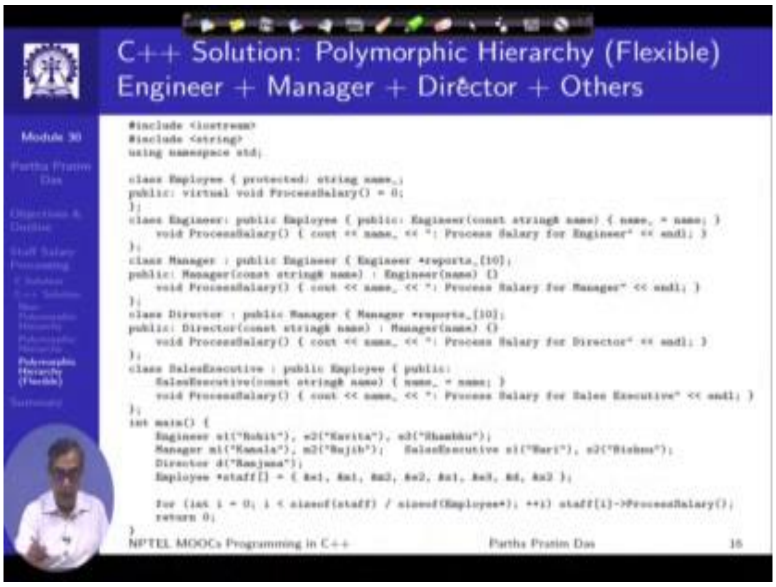
Now to the last part. So, what does the company want us to do? The last specialization was that in future they might want to add some division and in that division they might want to add more types of employees. So far our hierarchy was only this part. If they add in other division possibly something like sales division possibly something like sales executive will come in. We do not know in future what all divisions they are want to add, we do not know what are the different types of employees they are going to add; we do not know how many specializations of these employee types will existence and so on.

So, it becomes imperative, but we have all just seen that if we can have all this classes on a singly routed polymorphic hierarchy then my whole representation my whole code becomes very convenient to write. So we introduce a new route which they have not said, which the specialization has not said we say let they are be a concept call employee. Now there is nobody called employee in that organization. They are called engineer, manager, director and possibly in feature they will be some sells executive whatever. But we introduce there is a concept called employee and we said that let this be an abstract concept. There will be nobody who is an employee, but anybody who will perform the rule of an employee will be a specialization of this concept, specialization of this class there is a basic notion of the abstract base class.

So, we have now saying that let us extend this polymorphic hierarchy with abstract base class. Further, if this is a concept alone if there is no physical employee; which match this class then certainly it will mean that the processing salary logic for this particular class will not be known. So the process salary function member function that we had been using that will not be known for this employees, and that very nicely effects in to the abstractness of the concepts and this is not known all that we will need to do is to make that process salary a PR virtual function in this class because we will have no obligation to provide the processing logic for the processing of the salary in this class. So that is the change in the design that we bring in for the further extension we were working with a polymorphic class hierarchy now we have a polymorphic class hierarchy with an abstract base class.

All of these remain same, but in terms of virtual functions certainly we will have an additional virtual function in the employee class the route class which is where the all dispatch will enter which will be made a pure virtual function. So that is the basic design considerations that go in and let us look at the solution.

(Refer Slide Time: 23:59)



C++ Solution: Polymorphic Hierarchy (Flexible) Engineer + Manager + Director + Others

Module 30

Partha Pratim Das

Objectives & Outline

Staff Salary Processing

C Solution

C++ Solution

Non-Polymorphic Hierarchy

Polymorphic Hierarchy

Polymorphic Hierarchy (Flexible)

Summary

```
#include <iostream>
#include <string>
using namespace std;

class Employee { protected: string name_;
public: virtual void ProcessSalary() = 0;
};
class Engineer: public Employee { public: Engineer(const string& name) { name_ = name; }
    void ProcessSalary() { cout << name_ << ": Process Salary for Engineer" << endl; }
};
class Manager : public Engineer { Engineer *reports_[10];
public: Manager(const string& name) : Engineer(name) {}
    void ProcessSalary() { cout << name_ << ": Process Salary for Manager" << endl; }
};
class Director : public Manager { Manager *reports_[10];
public: Director(const string& name) : Manager(name) {}
    void ProcessSalary() { cout << name_ << ": Process Salary for Director" << endl; }
};
class SalesExecutive : public Employee { public:
    SalesExecutive(const string& name) { name_ = name; }
    void ProcessSalary() { cout << name_ << ": Process Salary for Sales Executive" << endl; }
};
int main() {
    Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib");   SalesExecutive s1("Hari"), s2("Bishnu");
    Director d("Ranjana");
    Employee *staff[] = { &e1, &m1, &m2, &e2, &s1, &e3, &d, &s2 };

    for (int i = 0; i < sizeof(staff) / sizeof(Employee*); ++i) staff[i]->ProcessSalary();
    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 16

This part engineer to director there is no change except that we have introduced a new base class. Since we have introduced a new base class the engineer is now specialization from that. Since we have done that we have moved the data member here. What is the need for moving the data member here, because earlier see if we just add an employee class and let us say we allow engineer to specialize for that and have the name here then certainly there will be no issue as you go to the manager and director and so on.

But once you open another division and here I have sells executive I will need to have name for the sells executive here as well. So, I will have duplication of code. How do you factor this out? You just factor this out by moving them to the common part. So that is why you get the name moving up to the abstract base class. So having said that then we have added another class sells executive which possibly comes from the new division which is yet not there, but we are just trying out our code and you see that the sells executive again directly inherits from employee because a it is going on a different part of the hierarchy.

Whereas, manager continues to inherit from the engineer, director continues to inherit from manager and so on, and each one of them has different implementation for the processing logic. And what binds them together, is put the processing logic member function this virtual member function in the base class employee and make that pure.

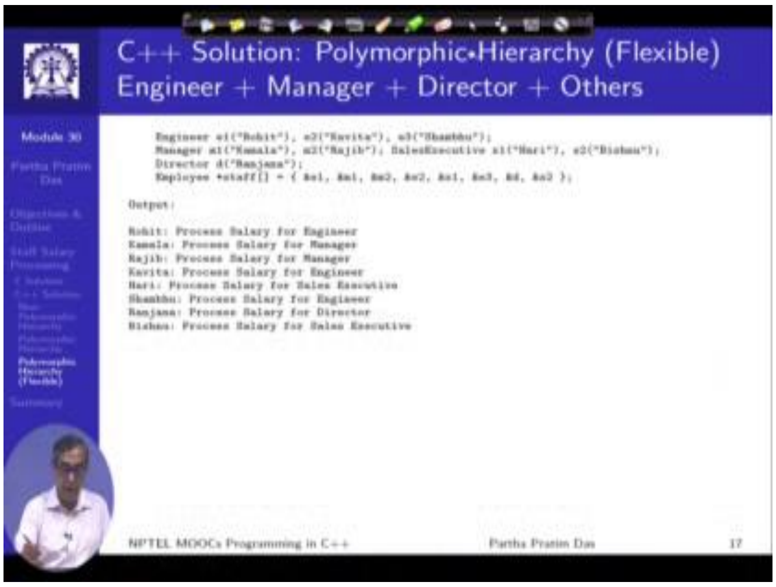
So that ensures that no employee instance can be constructed which gives us comfort because we do not know what would it have meant to construct an instance of an employee because we do not have any details about that. But it does allow us that we can now create our collection of employees as in employee array of employee pointer, earlier we were doing with array of engineer point. Now it is an array of employee pointer this pointers so that that is something very very beautiful you are you cannot actually construct an object, but you can always have a pointer of that type because pointer does not need an instance you just skipping an address and thinking about that address.

So, staff is thought of as if of this type and will use that information for doing the polymorphic dispatch, but it will actually have objects where none of them is of employee take their all of the specialize type which are concrete classes.

So, that makes the whole thing buying together multiple stands buying together. Interesting there is no change in this follow there is no change in this application code from what we did with a polymorphic hierarchy which did not have an abstract base class in the route. So, there is no impact in terms of the application code there is no impact in terms of any of the earlier classes that we have, there is no impact in terms of the container in which we have putting the object. But we have been able to further reduce refactor the code make it shorter and you had made it possible that from the employee any kind of other hierarchies could be made to specialize and my application code, my basic information here will not need to change.

So that is the power of polymorphic hierarchy particularly when you need to extend the designs which we often need to do and particularly when we use it with certain route classes which are abstract so that we can combined concepts which may not always be concretely realizable in terms of abstract base classes, but still can define operations for them, still can define pointers to those abstract base classes and use those pointers and the actual instance at the runtime to a dynamically bound navigation and a polymorphic dispatch for calling the actual member function for the object instance that we have at hand.

(Refer Slide Time: 29:02)



C++ Solution: Polymorphic Hierarchy (Flexible) Engineer + Manager + Director + Others

```
    Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");
    Manager m1("Kamala"), m2("Rajib"); SalesExecutive s1("Hari"), s2("Bishnu");
    Director d("Ranjana");
    Employee *staff[] = { &e1, &m1, &m2, &e2, &s1, &e3, &d, &s2 };
```

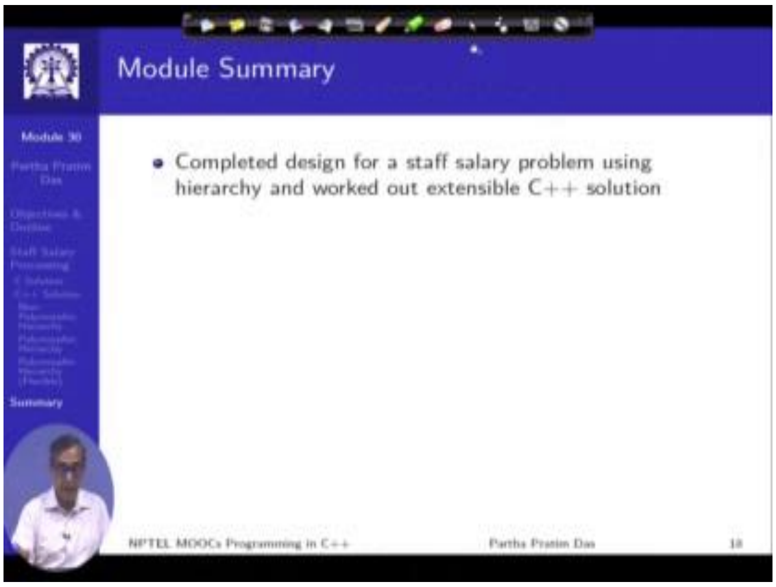
Output:

```
Rohit: Process Salary for Engineer
Kamala: Process Salary for Manager
Rajib: Process Salary for Manager
Kavita: Process Salary for Engineer
Hari: Process Salary for Sales Executive
Shambhu: Process Salary for Engineer
Ranjana: Process Salary for Director
Bishnu: Process Salary for Sales Executive
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 17

At present this is our final design, we might one to see if we want more refinements of that I would really like if you all you try to work this out and if you think that this design could be improved in some ways.

(Refer Slide Time: 29:11)



Module Summary

- Completed design for a staff salary problem using hierarchy and worked out extensible C++ solution

Module 30

Partha Pratim Das

Objectives & Outline

Staff Salary Processing

C Solution

C++ Solution

Non-Polymorphic Hierarchy

Polymorphic Hierarchy

Polymorphic Hierarchy (Flexible)

Summary

NPTEL MOOCs Programming in C++ Partha Pratim Das 18

Then you can just write on the forum you can take up and discuss that. But for now this is the completion of our design exercise on the staff salary application problem. In this we have tried to show a complete landscape of possible designs and solutions that you could do starting from c and in c 2 we started and showed how the whole thing and still be modeled, but there are several (Refer Time: 29:39).

Then you move to C++ and in this we have shown three stages of the solution - first, with non-polymorphic hierarchy; then, with polymorphic hierarchy; and then with a polymorphic hierarchy with abstract base class. And I hope that will give you lot of strength in terms of being able to do more designs in future.

In the next module and two we will take a brief look into all that we are saying in terms of the polymorphic dispatch the dynamic binding, how does that actually work, and then continue with other features of C++.