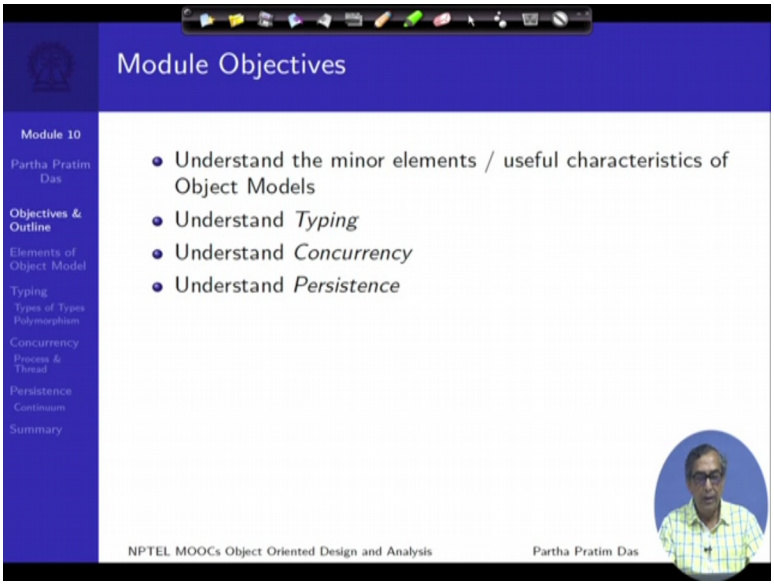
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 16
Elements of the Object Model (Minor): Typing, Concurrency and Persistence

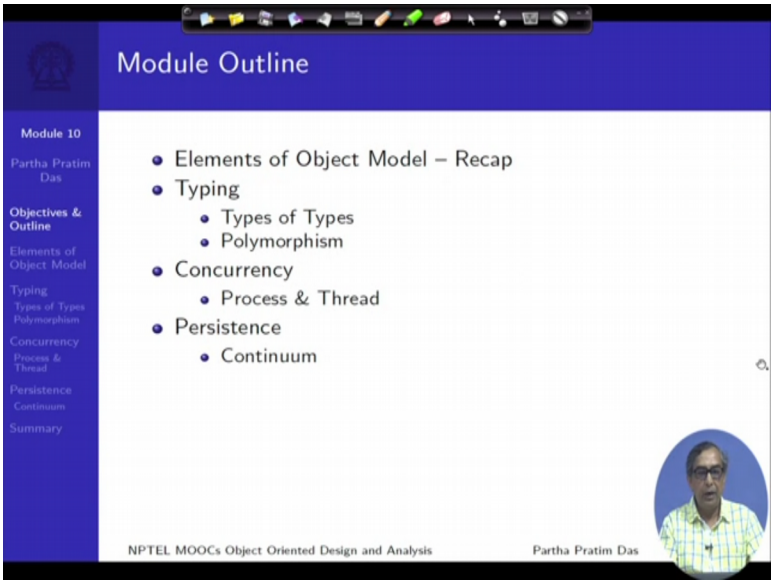
Welcome back to module 10 of object-oriented analysis and design. In the last modules 8 and 9 we have been discussing about major elements of object models. We have seen that object models ehh to be designed need to consider 4 major aspects of abstraction, encapsulation, modularity and hierarchy. They relate both to the aspects of design in the system when we talk about abstraction and hierarchy and they also strongly relate to how the implementation is done and how the code and different modules are organized.

(Refer Slide Time: 01:14)



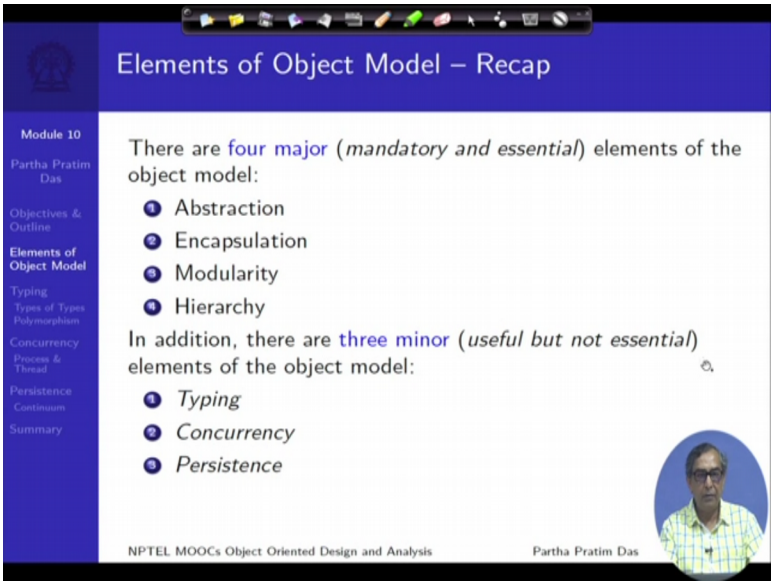
We continue on that and would like to conclude today on the elements of objects models and today we will discuss specifically about 3 minor elements these are useful elements in an object model but these are not essential depending on certain system requirements and the skill of the designer these elements will be considered. This will include typing, concurrency and persistence.

(Refer Slide Time: 01:41)



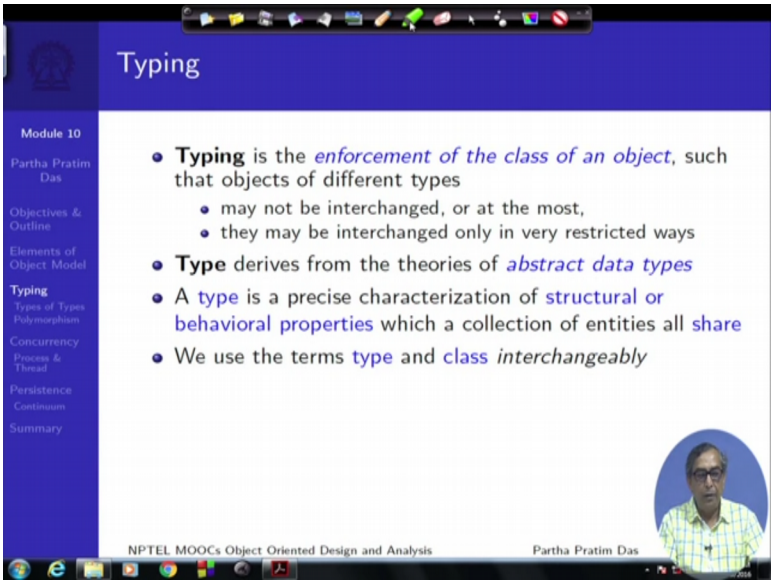
This is the module outline. This will be available on the left of every slide.

(Refer Slide Time: 01:49)



So, we have just this is just the context of the elements of the object model. Just for your recap, we have done the top 4 already. We are doing the bottom 3.

(Refer Slide Time: 02:00)



So, the first element minor element that we deal with is called typing. Am sure all of you uhmm you have done some bit of programming in C or C++ or java or python. You would know that whenever you need to use a value or a lateral or whenever you need to use a variable there are certain types associated with it. So, if I say am talking about a variable x then I many not by default know what kind of value does it contain

Or if I talk about 2.3 I need to understand that wherever it is a floating-point number, how many bits it is represented in and so on. and things really get complex when we talk about objects which is not simple primitive types of values but they are composition of they are collection of several independent value items which we say are properties. So, type is the core concept which defines how should an object behave. So, in formal terms the typing is the enforcement of the class of an object.

So, typing says that if I know the class of the object or if I know the type of the object then it will make sure that that notion or the conditions or constraints of that class of that type will be enforced whenever the object is used. So, it may not be interchanged for different object types and if it is interchanged occasionally in some restricted ways then it will follow certain pre-defined rules. what in simple terms it means that if I have the 2 variables in C say x representing an integer variable ah which will contain an integer value

And p representing a pointing to character which will probably mean a C string then we cannot use them usually in interchangeable manner. I can use the integer value to add another integer value to it. But if have a pointer I cannot take another pointer and add these 2 pointers. But again, I can take an

integer and add to a pointer so all these different restrictions of behavior will be derived and imposed based on the type of the object.

So ehhmm just to give you the background and we will not have enough time to talk about in depth. Type basically derives from the theories of adts or abstract data types. A this is a kind of development into programming systems which happen quite before the object-oriented paradigm became dominant which abstract data types have to say that if we are talking about any type basically have 3 different components that you need to deal with.

So, for an adt you talk about a domain of values, you talk about operations and you talk about what is known as axioms. So, domain say the whole set of values that a variable of a certain type can take. So, if am talking about say 1-byte unsigned integer in the domain is the set of values going from 0 through 255. Operations say what can I do with that value. So, we know if we are talking about integers, then I can add the integer, I can multiply an integer and so on.

But when I add the operation has very specific meaning. For example, if I am working only with one-byte unsigned integer then if the result of addition of two such integers happen to be more than 255 then certainly my domain cannot support that. I cannot hold that value and we kind of say that we have an overflow of the values wrap around and if I add 1 with 256, 255 that is the largest possible value in 1 but unsigned then the result is not 256 but the result becomes 0.

So, this is these are the kind of different operations which are defined and then finally axioms say how these different operations interplay between themselves. For example, if I add one to an integer such kind of unsigned 1-bit byte integer, if I add 1 to it and then if I subtract 1 form it then I should get that original result. So, an axiom could be of the form that x + 1 - 1 should give me x. now if you will you will understand that you the immediate example I talked about if am adding 255 with 1 that is if the value of x is 1, I is 255, then x + 1 will become 0 by the use of overflow.

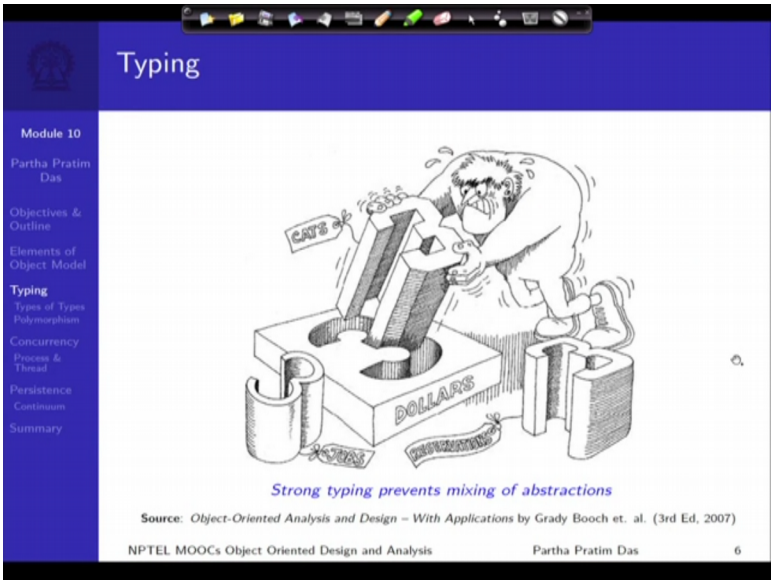
Now when I subtract 1 from 0 the value should become 255 by the underflow rule. So, this is the overflow rule, this will affect the underflow rule unless these rules are symmetric; this action will not work good. It may be that the overflow but underflow does not change the value. Then this action will be failed. So abstract data typing is the note in which tells you more on theoretical formal types as how should a type operate and over the years this has been used in several designs of several programming

languages.

But it has become less and less directly usable for the programmers because we have evolved the mechanism of object oriented programming where the use of the abstract data types or typing gives an underline principles underline promises on which the all these actions are ensured and you do not really need to explicitly know or talk about that. So, given that the type is precise characterization of the structural and behavioral properties we have seen about these properties, that means basically the attributes.

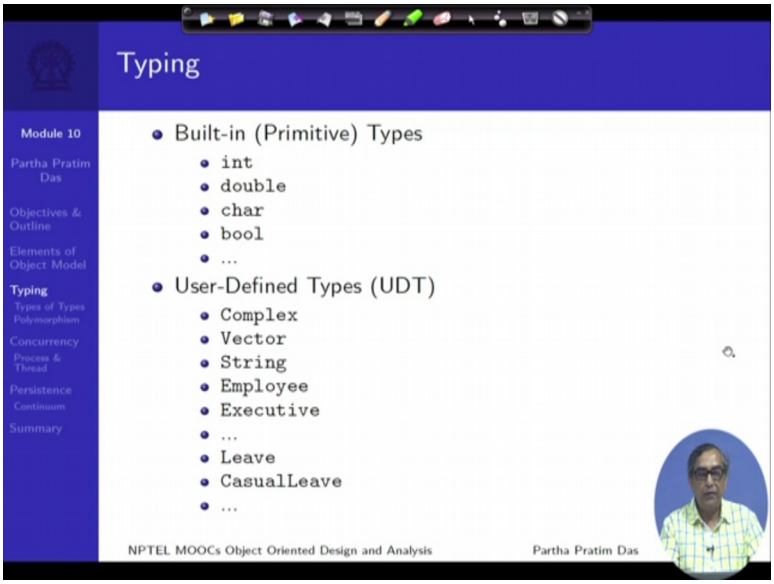
And the methods and it will talk about those properties which all entities of that type will share and just to clarify that we have been using the notion of class in the same context in the current course, we will use the type and class interchangeably though please remain informed that in general in the programming language theory or object-oriented theory, they are similar but not exactly the same but we will treat them as same in this course.

(Refer Slide Time: 09:18)



So, this is that very nice cartoon from our OOAD book, we can see that if type mismatches, both of them are 3 but this person is trying to fix one shape of 3 into another. so that's what happens when your types mismatch between variables and when we try to do something with that.

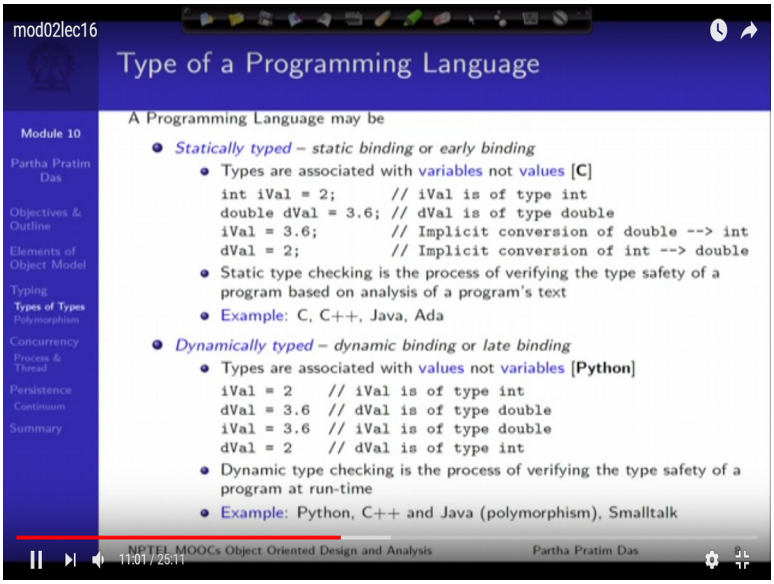
(Refer Slide Time: 09:37)



Certainly, several different kinds of types we have been exposed with significantly if we are coming from C and as you move on to C++ or in java, there are a number of built-in types or primitives types which are given in the language and then we have certain builders given in the language like struct in C or struct or union in C or in C++ you have struct union, class these different kinds which will allow us to define new user defined types these are called udts

And these are some examples of udts for example in C++ you can make a complex class of complex numbers which will certainly have different kind of an algorithms for addition than the algorithm you use for adding 2 integers. It will have also new operations like taking the modulus of a complex number and so on, we can vector types, we can have string types and if we refer back to our LMS example these are these basically are possible from our LMS example that we could define types or class for executives, for employee, for lead, for leave, for causal leave and so on. so, these are different kinds of types that we will read out.

(Refer Slide Time: 10:56)



Now we need, we would like to take a look into the type of a programming language. This particular heading has a little bit tone what use of the word type means kind of programming languages or the data type of programming language you can inter predict either way so typically programming languages are classified into 2 categories as strongly statically typed or dynamically typed. A statically typed language use static binding or early binding.

A dynamically typed binding uses dynamic binding or late binding. In exact simple terms, what it means is just focus into this these 2 statements. When we are talking about type, certainly we have the context is of a variable and a value that resides in the variable. If my language is statically typed, then the type is associated with the variable. So, you say that the variable has a type and that you define at the time of writing the program and the compiler while compiling determines THAT this is the type for that variable and every context will enforce that type.

So, the type is associated with the variable, it happens at the compile time that is the reason you call it the so these are the process of verify types of your program is based on the analysis of the program's text that is the compiler is working, no data is there and that therefore it is statically typed. So, if I look into this simple example of C, you all would know if I define 2 variables ival of integer type, dval of the double type then the type is associated with these variables.

So, when I start doing certain things that I should probably not doing, I take a double value and try to put to ival, the type of the variable is still hold on to be of integer. So, what it does the compiler to make this assignment possible, it could have done 2 things. 1 is it could have disallowed it. Some

language will exactly disallow that or some languages like C or C++ will allow implicit conversion. It will take 3.6 as a double value knowing that ival is int will make a conversion which in this case will be a truncation

3.6 the point 6 part will be just thrown away and 3 will get assigned to the ival. Similar thing will happen if I do the other thing 2 will become 2.0 when assign to dval. In contrast if use a dynamically typed language the nature of the code is very similar. Its only that the code is written in python and the important commission is all these preceding predefined types for the variable names. Now we do not provide any. So, what do we say is the type now is associated with the value?

So, it is not determined at the time of compilation. It is determined at the time of execution at the dynamic timing. So, you look at the value the value being 2, you infer that the value is of an int type and since you are assigning that to ival, so you infer that this is of int type and since you are making this assignment you infer this int type on this ival. So, you say ival is int. but at a later point of time, say in here these are these are all sequence of instructions in python.

So, at a later point of time, I choose to assign 3.6, 3.6 naturally has a type which is double or floating point. So, what will happen when I make this assignment this type is put to ival. So ival becomes a double type variable and this is not restricted to primitive types alone. This will also extend to the different user defined types that python allows that different classes. So, the type of a variable in python is the type of the value that was last assigned to that variable and that type is maintained.

So, the that is the difference in in these 2 statements here in static case, it is associated with the variable; in the dynamic case, it is associated with the values. So that's the basic difference between dynamic and static and that gives rise to a lot of consequences because if you can statically type a language, then the advantage that you get is a you can add compile time, you can say that well this probable is wrong this probably is a wrong assignment or this probably is a wrong invocation.

In contrast if its dynamically typed you cannot say that but when we are going to use different kinds of objects moving around, different kinds of objects on different hierarchies if you recall the vehicle hierarchy we talked of earlier, a vehicle could be land, water, air. A land vehicle could be car or bus. Now if I just get an instance of a vehicle I may not exactly know whether it is a car or a ship or it is an aero plane or tin what it actually is. So, it may be very difficult to conclude at the static time to decide

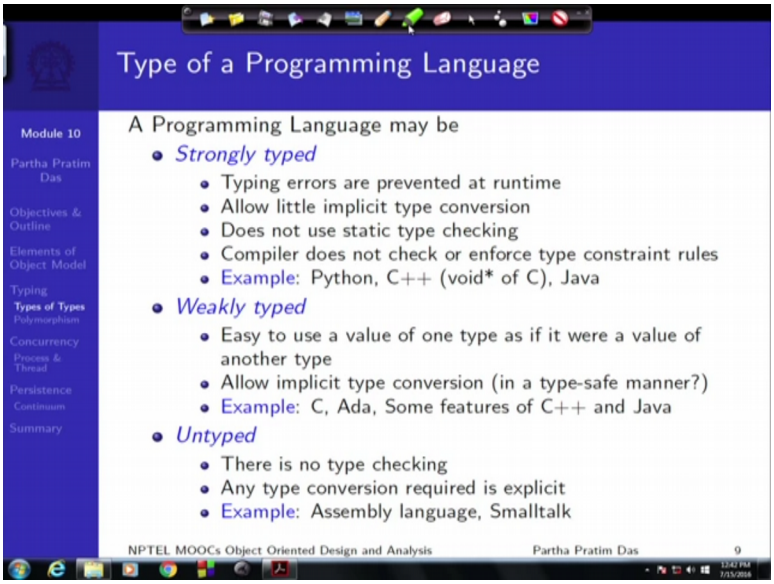
what kind of operations I should allow.

If it is a car, I should ask for how many wheels it has. But if it is a ship I should allow how many different parts does its propeller have and so on. so, there is always a trade of between whether I want to do static typing for better program safety or I want to do dynamic typing better flexibility and more object orientation. So, what happens if we try to look into the more common languages that we intend to u use, C is certainly strong statically typed.

Everything is done as a static time so in C++ and java but C++ and java also support part of dynamic typing through a mechanism called polymorphism and as will see that will become a very strong handle to support object orientation. So, C++ java as a language is crossly statically typed but they also support some dynamic typing whereas if you look at languages like python they are only dynamically typed. There is no static typing in python.

So, this is one way to look at the programming languages and am trying to relate this minor typing element of object models to the language characteristic because when you finally at the end of the design when you choose the programming language, the OOP programming language and want to convert your design your classes, the hierarchies, encapsulation into that language code, you will need to understand what kind of support you can get from the language.

(Refer Slide Time: 18:19)



There is another kind of typing or classification of programming languages that's commonly studied thesis between strongly typed, weakly typed and un typed. A strongly typed language say that anything

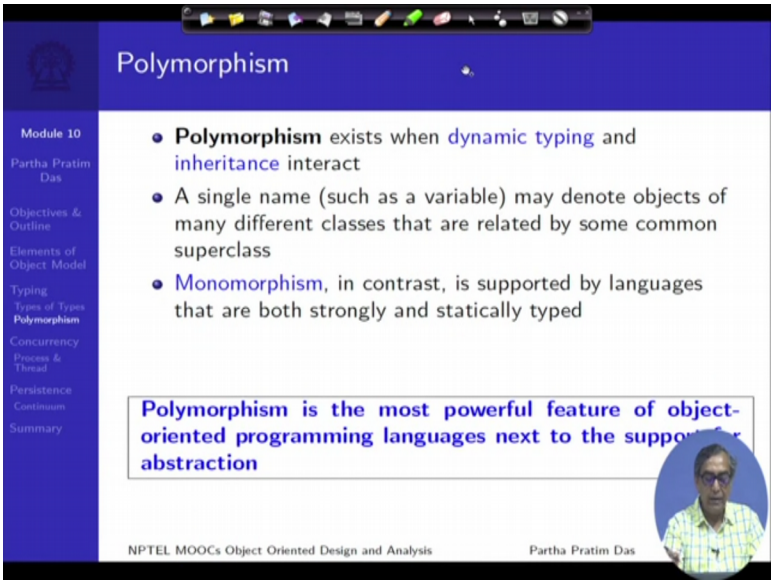
that you use that must have a type and that type must be a very strict one. So, typing errors will get prevented at runtime because you only allow things to happen if the type is completely known and can be completely reduced. It does not allow or allow very little implicit conversion

So, you would not end up allowing double to be used in place of an int and so on. thereby making sure that you are always using the value of a correct type in the place of a correct usage and so certainly strong typing often cannot use the static typing mechanisms because statically you may not know the type of a value all the time. So, as it turns out that languages like python have very strong type. C also is a principle after strong typing but as you understand C subjoins am sorry C++ have strong typing and but C++ also subjoins the whole of C which is weakly typed.

It has a plenty of scope for implicit conversion, it explicitly has provided a world star kind of pointer that is pointer with no type to take care of different specific programming situations. So those actually make C++ also weakly typed where you can do things that you could do in C. but if you make a guideline if you make a make a policy at your personal level or at an organizational level that you will only use the pure form of C++, the object oriented and strongly typed part of C++.

You will still be able to do all the programming but of course you will need some skill as to use that and you will have to drop all the weakly typed paradigms that C offers you by default. So, these are the 2 basic different categories and of course there is a third category of programming languages which are un typed that is where you do not need to you do not specify the type of a value ever till you use it, at the time of using it you will say that this is the type

And so, the type significantly does not remain in the program. It remains in the mind of the programmer. So, these are more difficult languages to program in and we will not expect typically an object-oriented system to be coded in this. For example, is any kind of assembly language if for all those who know assembly languages will understand if you are working in assembly languages you are directly dealing with bit patterns and therefore there is no specific type which is known.
(Refer Slide Time: 21:44)



So, but this is am just including these notes. Un typed languages as a for completeness we will typically not make a reference to this. So, given these different kinds of classification of programming languages so just we had strongly typed, weakly typed languages and we have statically typed and dynamically typed languages and there are different combinations of that, the language can be statically typed and strongly typed, statically typed and weakly typed and so on.

And C++ falls in kind of a mixed boundary where as I said mostly it is statically typed, for polymorphism it gets it is dynamically typed. Mostly it is strongly typed but if you use pure C construct within C++ then you will trade into the weakly typed territory of C++. Another concept which you just need to start getting aware of in con context of typing is known as polymorphism. polymorphism as the name suggest basically they has two parts here,

Poly is one part one word it says many, morph means change so polymorphism is basically the mechanism of working with change. So, it exists when dynamic typing interplay, interact with inheritance. It talked about inheritance as a part of hierarchy just recall we talked about single inheritance, we talked about multilevel inheritance and so on. so, when you would like to code those kind of object models into language, we will need the inheritance to be presenting the object language.

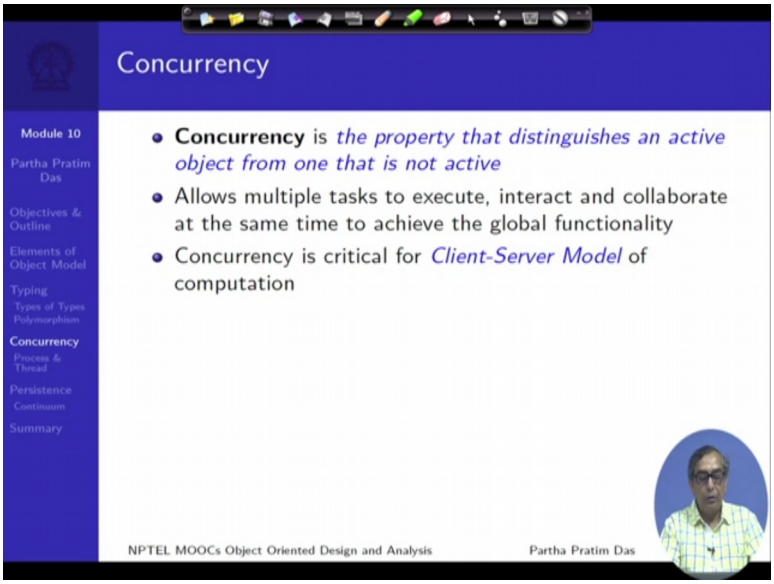
But along with that if you also have dynamic typing that is on that inheritance model which you do not know on the time of compiling what a specific object is, which class does it belong to, then I need to resolve that dynamically then you will need to make use of dynamic typing and when you combine these 2, you have polymorphism. C++ naturally gives a very strong framework for polymorphism and

so does java so that's become very good vehicles for object oriented programming.

There are languages which are monomorphic in nature which does not allow such behavior to be encouraged. They are strongly and statically typed languages. So, to conclude on this part, please remember that polymorphism is the most powerful feature of object oriented programming next to abstraction. So, if we if you are now faced with the question of what are the 2 top features that an object programming language must provide so that you can realize all the object models easily in them.

It will be the number 1, could be abstraction which you have already studied and now you are getting introduced to polymorphism which is typing at the runtime on an inheritance hierarchy. We will all of you may not fully understand what am talking of here. But do not get worried as we go along this course you will talk about these polymorphism, polymorphic behavior and the and its use in the object-oriented analysis and design.

(Refer Slide Time: 25:07)



Further you want to talk about this concurrency. we will take a brake and talk about that.