FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture37

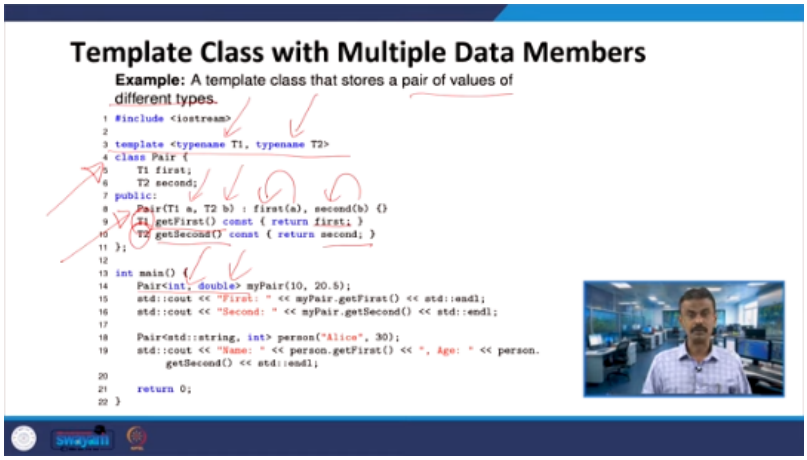
Lecture 37: Template Class in C++

Welcome to Lecture 37, Templates and Generics. So, we have seen some examples with respect to C++. What do you mean by a template? And we have written some programs, if you recall, in the last lecture, all right. So, continuing from there, here we can have a template class with multiple data members.

In fact, we have seen template functions and template classes as well. And here, I will continue with the template class with multiple data members. So, let us assume, in this example, this template class is storing a pair of values. Right, of different types—a pair of values of different types. So here, you see the template syntax, and I am going to use typename T1 and another one, typename T2. So, let us have a class Pair, all right? So, let us have a class Pair. Correct. So, T1 first, T2 second. Suppose I am using one as integer and another as double. So, T1 integer is equivalent to int first, double second. Okay.

So, now you have a constructor—a parameterized constructor—with one parameter, a, and another argument, b. One argument is a, another argument is b, and then you are copying a into first and b into second. Correct. So, this is a parameterized constructor, and then you have two functions. Member functions: getFirst, return type is T1; getSecond, return type is T2, all right. So, it is returning first and second—not a very difficult program. So, here you have—so let us say class Pair. This is the syntax; you have to mention one as int and another as double.

In the last class, we have done, let us say, a pair with only one template, right. So, only one type. So, in that case, what we have done is pair int and then close the angle bracket. So, here we are using two right pairs of values, right. So, in that case, you put int comma double, which means t1 is int and t2 is double in this case, right.



So, you are creating an object my pair and then passing 10 comma 20.5. So, therefore, it will call the parameterized constructor, right? A is assigned to first, which means first is equal to 10. Right, that is the meaning, and the second will be equal to 20.5, right? Second will be equal to 20.5. So, line number 15, you have first, so this will be printed. My pair is invoking get first, right? My pair is invoking get first. So, get first is returning first. What is first? 10. So, this will produce 10. This will give output 10. And second will be printed. My pair is invoking get second. Get second returns second. Second will be 20.5.

20.5 will be printed. So, this is the first case. And the second case, you have pair. You have string and integer. Now it is a string.

So now T1 is a string. T2 is an integer. So this is the advantage. And then you are creating an object person under pair. Name is Alice.

Age is 30. Right. So something like that. Two arguments. Right.

Two values you are passing. Right. So one is a string. Another one is an integer. Integer value.

Right. And then you are printing. Next line. Name you are printing. So person is invoking.

This is object. Is invoking get first. So get first when it is invoking. Here you have return first. So what is your first?

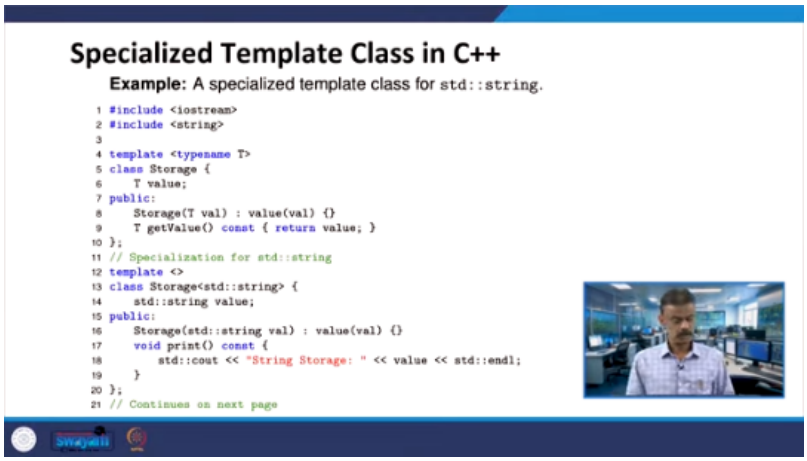
Allies. Similarly, right, you are printing age, person is invoking, get second. What is get second? 30, correct? So this is because you are passing this.

A is assigned to first, B is assigned to second, right? So, allies is assigned to first and 30 is assigned to second, right? So now, this is the advantage. So in one case, I am using it as an integer and a double. In another case, I am using a string and an integer.

You may have one more, right, as a double, comma double. Possible, right? A double comma integer, float comma integer. So, it is possible. So, that is the advantage of the template class and template functions, right? So, let us run the code.

So, template class with multiple data members, all right? Yes, my pair 10 comma 20.5, person allies in 30. If we run the code, yes, we are getting the output: first is 10 and second is 20.5. Name allies, age 30. Okay. So, this is the output we can get. Right.

So, this is the advantage of using the template class. So, let us see one more example. So, this is a specialized template class in C++. Right. So, one is the usual one.



Line numbers 4 to 10. Right. Template typename T. Class storage. Right. It is a template class.

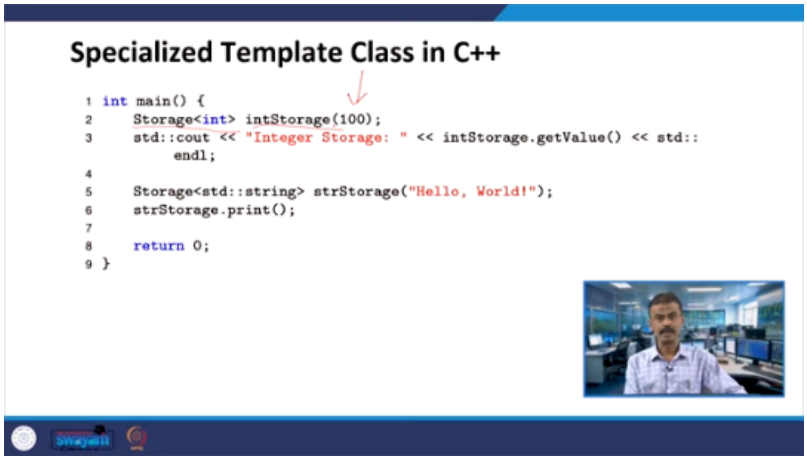
You have one member data value whose data type is T. And then you have member function storage. Right, val, passing one value. So, that val will be stored in value, right. So, this one, which is a constructor. Line number 8 is nothing but a constructor, parameter is constructor.

And line number 9, you have get value function, member function, whose return type is t and it is returning the value, right. So, this is a usual template class that we had seen. So, now you are seeing the specialized template class with the same name. So, but what you are doing when you look at line number 12 you are using this particular syntax template angle bracket only without anything right template angle bracket and then you are using class storage right. So, class storage is a specialization right or you call it as a specialized template class storage and then here you have string previously t. So, the t will be for any data type and here storage we have string.

Specialized one the specialization for string and then you have set of statement you have value which is a data type whose class is string all right whose data type is string value is a variable whose data type is string and here you have constructor storage you are passing one value whose data type is string and that will be copied val will be copied into value exactly what we have done previously. Only one member function, which is a print member function, which is not available over there. So, string storage. So, this will be printed. Okay.

So, this is what the difference between the usual template and the specialized template. So, in this particular case, the specialized template is for the string. So, now you go. All right. So, here assume that you have created an object in storage.

Right. Here, you are passing 100. And then, if you look at the class storage and then your type. So, the template type is integer. Type name is integer T. Capital T is becoming integer.



You are passing 100. When I am passing 100, now the value is equal to 100. And then you have a CO statement, integer storage. So, this will be printed. And in storage, you have created an object which is invoking getValue.

So, when it is invoking getValue, the value will be printed. What is the value? 100. 100 will be printed. Right.

Now, we go to line number 5. So, I have string storage. Right. STR storage. Another object.

Under storage STD string. That is a specialized template. Right. So, that means this will call this particular function. Right.

So, let us see. You are passing hello world. Right. So, that means the constructor will be called. And val will be copied into value.

It is a string. You are passing string. What you are passing? You are passing allow world. Right.

Now value is nothing but allow world. Correct. So when I am calling this print using the str storage. That means str storage is invoking the print function. Print member function.

So this will be printed. String storage. What is the value? Allow world. String storage will be printed.

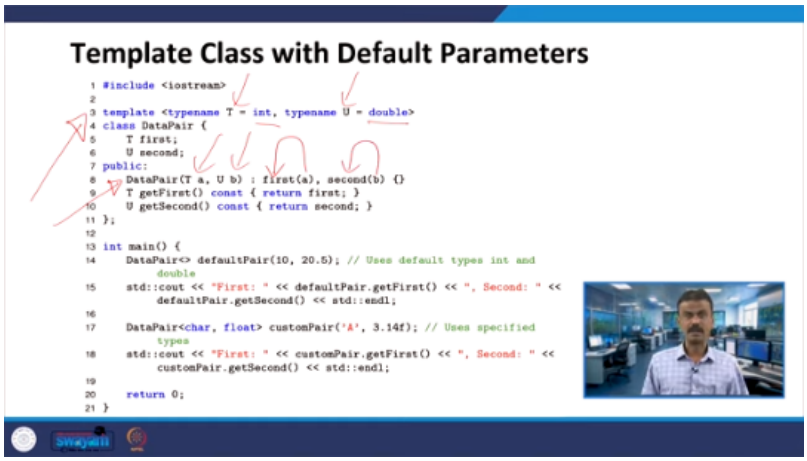
Allow world. One is 100 will be my output, integer storage 100, here string storage hello world, right. So, let us run the code in C++. Yeah. So, we have a specialized one, line number 11.

And when I run the code, compile and run the code, yes, I am getting the output. Integer storage 100 and string storage hello world. Okay. I hope it is clear for everyone. Also, we can have a template class with default parameters.

Right. So, we use T, U, T1, T2, something like that. Right. So, in that case, what we can do. So, assume that I put.

Right. So, assume that I put one type T. Type name 1 is integer. Another type name U is double. Right. Default.

So, that means T here. class data pair t first u second so initially without loss of generality your t is integer and u is double so that means is equivalent to in first and double second so now you have data pair which is a constructor two argument constructor you have two parameters a and b so a is copied into first and b is copied into second so whatever you are passing it will be copied Respectively first and second. So now you have a function. Let us say this is a member function.



Get first. Alright. So your return type is t. That means the integer. So it is returning first. Second one get second.

By default, it is u. That means a double. It is returning second. Okay. So now, let us go to the main program. So you have a data pair.

If I do not put anything in the angle bracket, it is considered as default. What is default? Integer and double. You are using two type names.

One is integer, another one is double. So, you have integer 10 and double 20.5, right? You have an integer 10 and a double 20.5. That means you have an object default pair, right? You are passing 10 and 25, correct?

So, you are passing 10 and 20.5, right? So, what will happen? So here you will, first will be 10, right? Here you have first and second as the variable, right? So now it is by default integer and double, correct?

So 10 you are passing, that is A. So first will be now 10 and second will be now 20.5, clear? And then let us assume, I mean it is printing C out. Default pair is an

object calling get first. So get first, when it is calling get first, it is returning first. What is first? 10.

So, I will get the output 10 and second case default pair is calling get second, get second you will get it will return second, second is 20.5. So, this is the case of default right. So, now you may ask the question. So, can I use this T and U for any other data types right. Yes, the answer is yes we are going to see line number 17.

So, here what we are doing data pair Correct. Same class name. Here you put character and float. If you do not put only the angle bracket without anything, it will be a default.

So here in this case, even though the default is t and u, integer and double, here in this case what I am doing, I am completely changing it. Right. So data pair, I put character and float. So when you do this, t is becoming character and u is becoming float. So, it is like in the procedural oriented programming you put int a equal to 10 initially right you can declare like this and then at one point of time what you are doing you can do a plus plus right.

So, similarly by default right. So, here it is integer and double. So, later I am changing it or a equal to a plus 2 same variable now it is a different value. So the same template class, now your T and U are different. T is character, U is float.

And then you are creating a custom pair object. Your character is A and your next float is 3.14F. F stands for float, correct? So now see out, first it will print. Custom pair is invoking get first, right?

Custom pair is invoking get first. So get first is returning first. So what is first now? A, right? This is being copied.

A is being copied into first. 3.14F is copied into second, correct? So custom pair, when it is invoking get first, A will be printed, correct? And the second will be printed. First also will be printed.

Second will be printed. And the custom pair is now invoking get second. Get second is returning second. What is second? Second is nothing but 3.14, right?

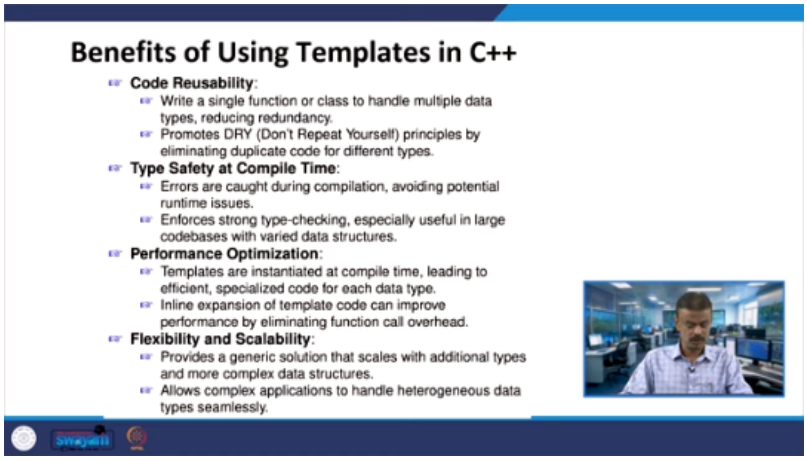
So, this is the output we will get when you run the code, right? So, this is the output we are getting. So, first is 10, second is 20.5, and in the second case, first

is A, second is 3.14, right? So, this is the output you are getting. First is 10, second is 20.5.

So, you run the code. It is not very difficult: A and second is 3.14, right? So, this is the explanation of the previous code. So, I have completely done that. So, when you are seeing for your nodes, what exactly have we done?

So, if you put only the angle bracket, it will be considered as default. So, in this case, default was integer and double. You may ask the question: can we use this T and U for other data types? The answer is yes, and we have seen it changing to character and float. Accordingly, your first and second are changing, and then we can get the output like this, right? So, this is called the template class with default parameters. So, we have so many advantages, so many benefits—code reusability. So, we have seen when you are using a function or class, there is no need to use it again and again, no need to write the code repeatedly—maximum of two numbers, integers, maximum of two doubles, maximum of two characters. So, no need to write

again and again. So, that is what reduces redundancy. Alright. Also, it promotes 'don’t repeat yourself.' No need to repeat.



Suppose I ask you to write a program to find the maximum of two integers and the maximum of two doubles. Right. So, before this concept, what did you have to do? You had to write them separately. Whereas now, you don’t repeat this.

That is called the 'Don't Repeat Yourself' principle. So, it is about eliminating duplicate code. Again, you have to use the same logic, right? When you are

writing the maximum of two numbers, whatever the numbers are, right? So, that is saving.

That is called code reusability. And type safety at compile time. So, any errors will be caught during compilation itself, right? So, it will avoid runtime issues. And when I am talking about type safety, it is strong type checking, right?

It is enforcing strong type checking, especially in large code bases with varied data structures, right? We talked about type safety when you are using, let's say, integers for the maximum of two numbers, and the two numbers are integers, and then you are using doubles. So, the type safety will be there; data type safety will be there. Performance optimization, of course, when you are using, let's say, the maximum of two integers, doubles, and characters.

Two characters are given which is the maximum right based on the SK value. So, we no need to repeat the code again. So, that will give you the optimized code all right and the compilation is also only for that particular. Suppose it is a function I am talking class also right. So, the code will be optimized one particularly when you want to efficient code all right or you are participating in some of the contest right.

It is advisable to use template. So, The performance will be optimized. And you have the additional features like the flexibility and scalability, right? So, these are all the major benefits of templates.

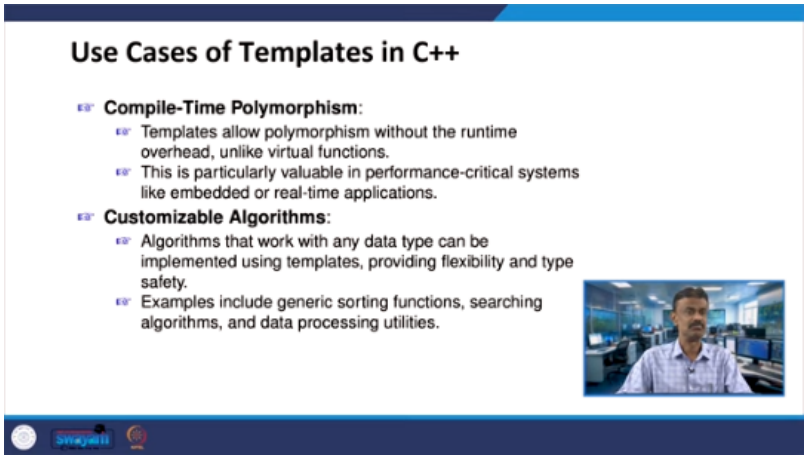
Also, when we talk about STL, standard template library, particularly when you are doing data structures, right? So, when you use, let us say, stack, queues, linked list. Right. So, you have standard template library. Right.

So, it is having the collection like a vector, map, set, list, link list, etc. Right. So, in fact, these data structures you can reuse and in fact, you can store any type. Right. So, which is improving the modularity.

Also, mathematical and algorithmic libraries. Assume that if you are using the maximum, minimum, many mathematical functions, square root, right, seal function, floor function. So, this can operate on any numeric type, right. So, the templates you can operate on any numerical types like you have libraries like a linear algebra or graph algorithms or numerical methods, right. So, the template is having high edge over your usual traditional approach.

Also, we have the concept of smart pointers. right, smart pointers and memory management, all right. So, you can cleverly do this, right. So, we have smart pointers like unique underscore PTR, you try, right. So, unique underscore PTR or shared underscore PTR.

So, this can handle the memory for any object type, all right. So, these are all the essential features. In fact, when I am talking about the benefits, these are all four important benefits and these are all nothing but the use cases right generic data structures you can use like a vector map set list and link list etc right so many you can go through that and mathematical and algorithmic libraries with the help of templates right also the memory management we can use the smart pointers also you have right some more use cases like a compile time polymorphism all right we talked about compile time polymorphism right.



Function overloading, operator overloading, right. So, these are all nothing but the compile time polymorphism. So, like this here if you get any error, right. So, that can be pointed out during your compilation time itself, right. So, the template when you are using a template that is allowing the polymorphism, right.

So, we no need to worry about the runtime overhead like virtual function. right and also the customizable algorithms. So, algorithms which are working with any data type right. So, which can be implemented using templates right. So, this is providing flexibility and type safety.

So, if I talk about any examples, any generic scoring functions or searching algorithms and data processing utilities, these are all customizable algorithms. So, the use cases of templates, the goal like this, you can have generic data

structures, mathematical and algorithmic libraries, smart pointers, right, compile-time polymorphism, and customizable algorithms. So, the next concept we are going to see is generics, right? I will have the basic introduction So, like template classes and template functions that we had seen in C++, in Java we have generics, all right.

So, the concept is generics, exactly like template classes and template functions that we had seen. So, here, this is enforcing type safety, same as in C++. So, that is at compile time, all right. So, when it is allowing, right, while allowing classes and methods, right? So here, in this case, classes and methods, we call to operate on any specified data type. So whatever we define as templates in C++, here we call it generics in Java, all right. So it is introduced in Java 5. So, generics, which enable a type or method to be a placeholder for types, so that means when you are using, assume that classes and interfaces

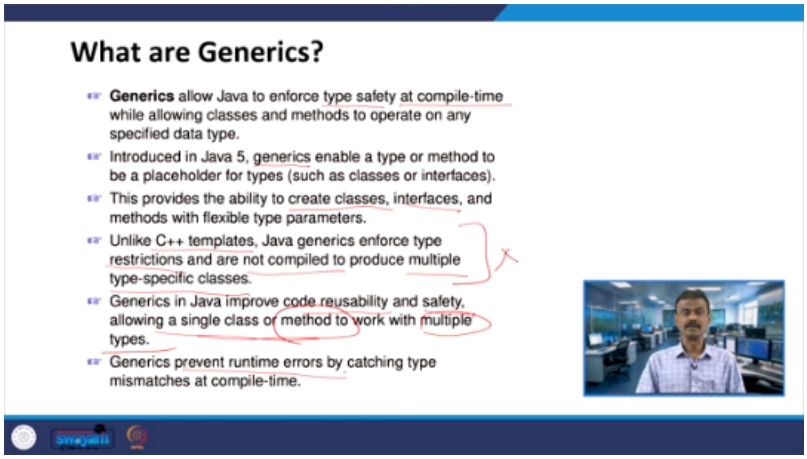
So, we talked about classes and interfaces. So, there, I mean, we can use generics. So, generics, which are able to, right, to create classes, interfaces, and methods, right. So, here, I mean, in C++, we talked about classes and member functions, right? Classes and, in fact, functions, template functions. And here, in this case, you can have a generic class, generic interface, and generic. When I am talking about C++ templates, so there, you can have multiple type-specific classes.

We had seen multiple type specific classes. So whereas in the case of a Java, we have some restrictions. So we cannot have the multiple type specific classes. So that is the difference. That is the major difference.

right so there we can use in fact we have seen the example right so multiple type specific classes whereas here that is not available that is not compiled like c plus plus so here also you can have the code reusability right and data safety etc right so when i am using a single class or method all right so that can work with the multiple types so exactly like what we had taken the example maximum of write two numbers integers all right so i can have one single method so the numbers may be integer maybe double maybe let us say character or even maybe string you are comparing two string and we can say right which one is greater all right so this you can have only one method and that method when you are talking about generic one method and that method can be useful for multiple types

exactly same what we have studied in c plus plus we call those are templates. So, here we are going to call it as generics right. So, only few things are not available for so far we had seen this is not available.

Whereas, the multiple type specific classes we had seen in C++, right. So, here we do not have. Obviously, so since everything is happening during compile time, it is preventing the runtime, right. So, if there is any exception, so that will be caught during the compile time itself, right. So, what we can do in the next lecture, we will talk about some examples of generics under Java.



Thank you.