FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture46

Lecture 46: Design Patterns

Welcome to lecture number 46 on design patterns. So, what are design patterns? So, we talked about templates, generics, and reusability. Similarly, when I talk about design patterns, they are also reusable solutions to common problems in software design, right? In fact, this provides a blueprint for solving issues related to object creation.

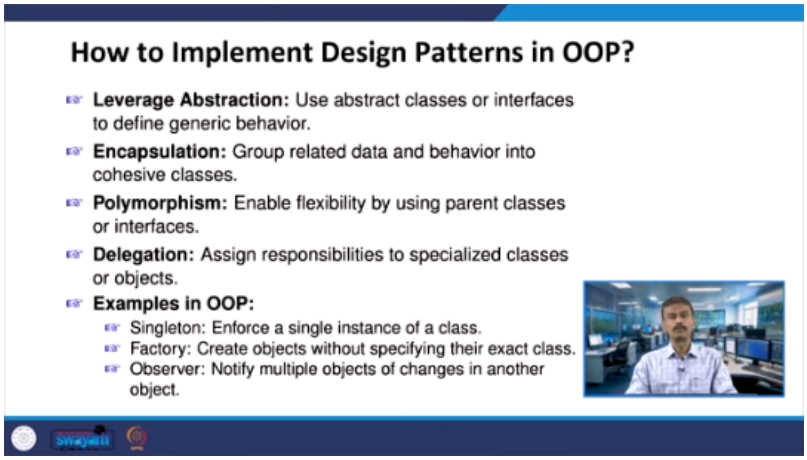
So, when you are creating or instantiating an object, right, and then the communication and then for behavior, we categorize this based on the purpose and usage. So here, we have design patterns like Singleton, Factory, and Observer. They are not specific implementations but serve as a guide for structuring our code very efficiently, right, and effectively as well. So, what are all the advantages of using design patterns? So, we can use this for code reusability, ease of maintenance, and improved communication. That means the patterns provide a common vocabulary for developers and flexibility.

So, we have, right, these patterns are useful for extensible and scalable software architecture. And they reduce development time, right? So patterns help solve recurring design problems very efficiently. So what are all the types of design patterns? We have creational patterns, structural patterns, and behavioral patterns. Under creational patterns, we are going to see what you mean by Singleton, what you mean by Factory, all right? And what you mean by Builder. These deal with object creation mechanisms. Similarly, when I am talking about Adapter Decorator and Composite, these focus on class and object composition, all right? So this is under structural patterns. And when I'm talking about behavioral patterns, we are going to have Observer, Strategy, and Command. These patterns handle object interactions and communication. So why are design patterns important in software design, right? Mainly, we have consistency, efficiency, scalability, robustness, and a problem-solving framework. So, the

meaning of consistency. So, it promotes standardized solutions across teams and projects.

Efficiency, it reduces development time by providing proven solutions. Scalability, which ensures the software can handle growing requirements seamlessly. Robustness, improves code quality by addressing common design pitfalls. And the problem-solving framework, which provides templates to tackle complex design challenges. So if you have any, right, a very hardcore problem or very hard problems, in that case, these all provide templates to tackle complex design challenges.

So now, how to implement design patterns in object-oriented programming? Abstraction, leverage abstraction. So we talked about abstraction. So, these design patterns use abstract classes or interfaces to define generic behavior. Then we have encapsulation, polymorphism.



So, these are all the cases where we can implement design patterns in OOP and delegation. So, this assigns responsibilities to specialized classes or objects. So, examples in OOP. So, we are going to see a singleton, factory, and observer as I already said. So, singleton, which enforces a single instance of a class, right.

So, we are going to talk about the single instance of a class. Factory, so this is helpful to create objects without specifying their exact class, right? And then, Observer. So, Observer notifies multiple objects of changes in another object, right? So, this is what we are going to see in the design patterns: Singleton, Factory, and Observer. I will start with the Singleton pattern.

So, what do you mean by the Singleton pattern? So, the Singleton pattern ensures that a class has only one instance globally, right? So, it will have a single instance, which is why it is named Singleton. I mean, in mathematics, you have a singleton set. So, it is a single instance, right?

So, the class which has only one instance. So, this instance is accessible through a global point of access. So, it is commonly used in scenarios where shared resources, configurations, or logging are needed. So, we are going to talk about shared resources, configurations, and logging. So, what are all the main features?

So, as I already said, it has a single global instance. So, controlled instantiation. So, when you are creating an object, you have controlled instantiation. And the singleton pattern, which simplifies access to shared resources, right? So, I will give an example: the singleton pattern with the analogy of a coffee machine, right?

So, in an office or any workplace, you can find a coffee machine, right? So, what is the purpose of a coffee machine? It saves resources and ensures consistency, right? So, the main idea is, right? So, instead of having multiple coffee machines,

All right. So, the singleton is acting as a single shared coffee machine. So, one coffee machine you can find. And then, this is providing functionality for everyone. In fact, it is providing the same functionality to everyone.

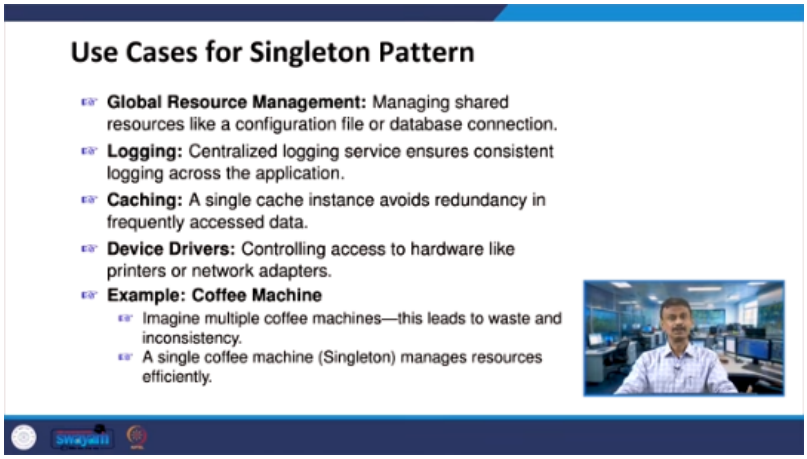
Suppose in the department. Right. I want to have a coffee machine. I'll buy one coffee machine. Right.

So, this coffee machine will provide functionality to everybody. So, in a similar way. So, in a class, we will have only one instance. Right. So, like the coffee machine in the office, only one coffee machine.

So, it serves the purpose for everybody. So, what are all the use cases for the singleton pattern? So, we have global resource management. That means the singleton pattern manages shared resources like a configuration file, database connection, logging, centralized logging service, ensuring consistent logging across the application, and caching. So, a single cache instance avoids redundancy in frequently accessed data.

The next one is the device drivers. So, this is controlling access to hardware like printers or network adapters. So, we already saw this example, the coffee machine. So, assume that imagine you are having multiple coffee machines. Assume that in a department, I am having four or five coffee machines.

So, one is a waste of money. And the person has to do a lot of work. So, before putting all the ingredients right. So, if it is only one I mean I no need to waste money and only one time I can put the ingredient in the morning. So, my work is also very less right.



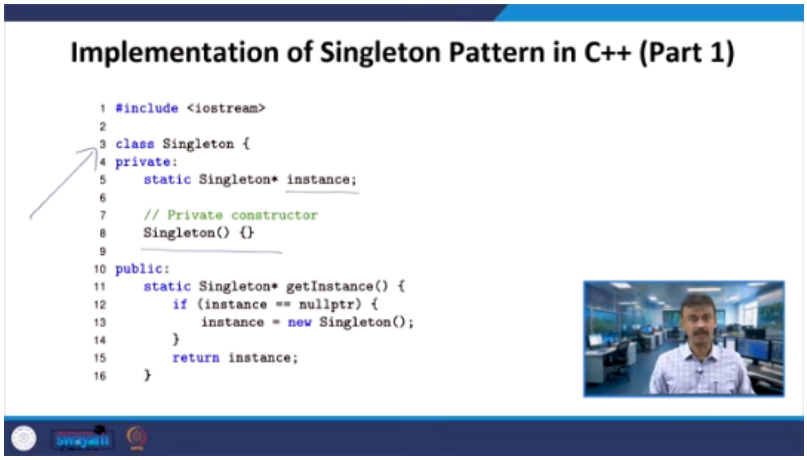
So, when I have a multiple coffee machines this leads to the waste and inconsistency right. So, that is why the single coffee machine now we call it as a single turn. That is managing resources very efficiently. So, now we will talk about the implementation of singleton pattern in C++. So, this is very important.

So, whatever we had seen the theory so far and then you know what is the meaning of singleton and you can have the analogy of coffee machine. That means a class will have one instance, one global instance. So, let us consider the class singleton. In line number 3, we have class singleton. Right.

So, you have one private instance under singleton. Right. It is a pointer variable instance. Right. This is a pointer object.

Correct. So, under singleton which is static and then we have one private constructor. Right. So, here you have the private constructor. Right.

Both are private. Right. Singleton's the instance. Right. The object.

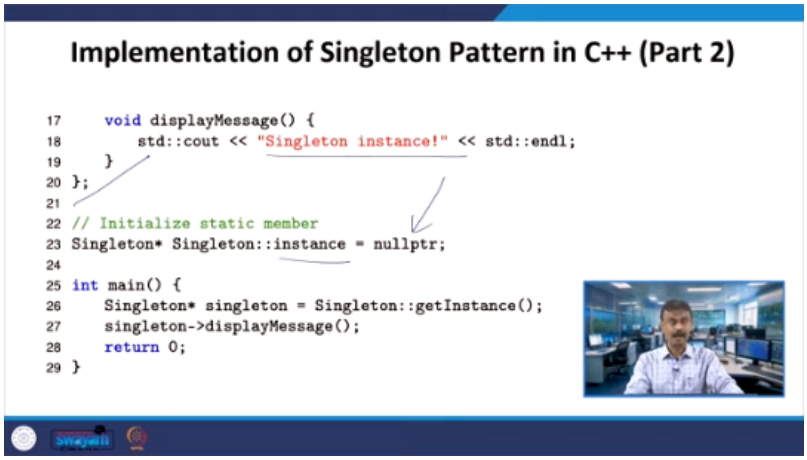


In fact, it is a pointer object. whose class is singleton and then you are having the constructor which is a private constructor right. So, here you have a public you have a function get instance the pointer function right under singleton. So, now this is checking if instance is equal to null pointer right. So, if the instance is pointing to the null pointer that means there is no instance if there is no instance you create the instance there is a meaning line number 13 you are creating an instance new singleton right this is a constructor and then

if it is not required already instance is there you are returning instance right already instance is there you are returning the instance so now you have one more member function display message so this display message is displaying singleton instance right and then your class is over here correct and line number 23 you have singleton all right so you have a singleton in fact the instance star instance We are initializing the static member, all right. So, we are initializing the static member. So, which is null pointer, which is pointing to null pointer. So, this is a pointer.

The instance is the pointer variable, right. Pointer object under singleton pointing to null pointer. That means we are initializing the static member. So, now go to the main program. It is a global in fact, right.

So, before main you are finding is a global in fact. And in the int main, so here we have singleton. Let us call the function get instance, right? The pointer function get instance. So get instance will call this, right?

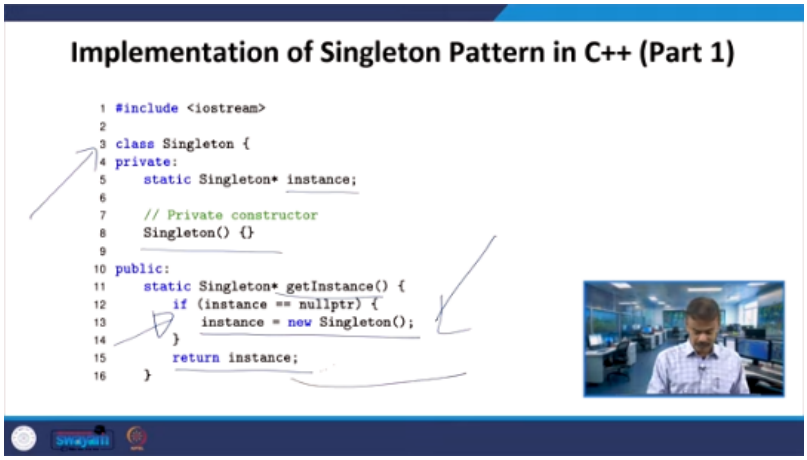


So here you have the singleton object, right? Singleton object. So when I am creating this pointer object, right? So the get instance function will be called. So, the get instance function will be called that is equal to get in the right hand side you have singleton get instance.

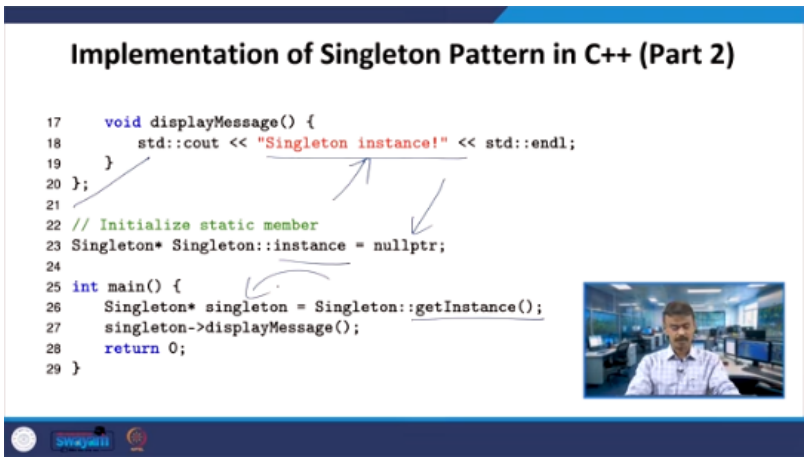
So, when this function is being called if there are no instance the instance will be created from line number 12 and 13. Otherwise if the instance occur so we know the singleton instance only one instance will be there the global instance will be there. So, that will be returned. So, return instance. So, whatever be the case the if there is no instance, instance will be created in line number 13.

If there already instance occur that will be returned in line number 15 right. So, then line number 27 the singleton small s right. So, that is invoking display message. So, the display message you have singleton instance right that is a cout statement. So, this cout statement will be executed right.

So, when I run the code we will get the output singleton instance. So, the main idea here is the class which has a singleton instance right one instance one global instance in fact right. So, like exactly office as one coffee machine. So, if the instance does not exist we are creating otherwise whatever the instance is being created. So, that will be returned right and then when I am displaying this message because the instance either created or already exists.



So, in that case, when the singleton in line number 27 invokes display message arrow display, because singleton is a pointer object, all right. So, the get instance is returning the instance. So, the instance will be here, right? So, the right-hand side will be return instance. So, singleton is a pointer, a pointer object.

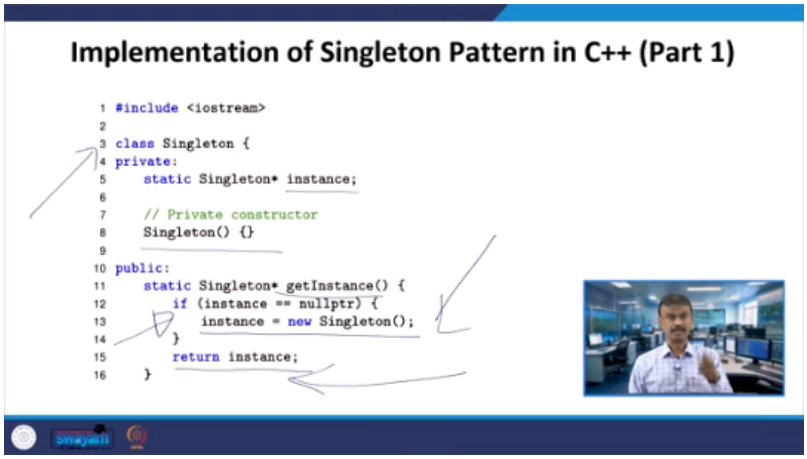


So, whatever you have returned, So, that will be pointing to singleton small s, and singleton, when it is invoking display message. So, the singleton instance will be printed, right? So, this is what exactly is happening, and this is an example of the implementation of the singleton pattern in C++, right? So, here we have seen the private constructor.

Line number 7 and 8, in fact, line number 8, you have a private constructor. So, this private constructor ensures that the class cannot be instantiated directly. So, that is what the private constructor is ensuring. And then you have a static instance, correct? So, here you have the static instance, right?

So, static singleton star instance. So, this is static instance. So, the static instance this is a pointer right which is holding the single instance of the class on coffee machine right in the office. So, in a class you are having the single instance. So, that is why you have static instance.

And then you have a member function called get instance. So, the get instance here you go. So, that is what I said if there is no instance it will create the instance if it is already there. So, definitely line number 12 and 13 will not be executed and if the instance is already existing. So, that will be returned in line number 15.



So, create the instance if it does not exist returns the existing instance otherwise. So this singleton implementation simplifies resources management by ensuring a single shared instance. Exactly you have to think about the single coffee machine in the office. So this class has a single shared instance. So now in this program so what we do the implementation of singleton pattern in Java we will see the last program we had seen in terms of C++ right.

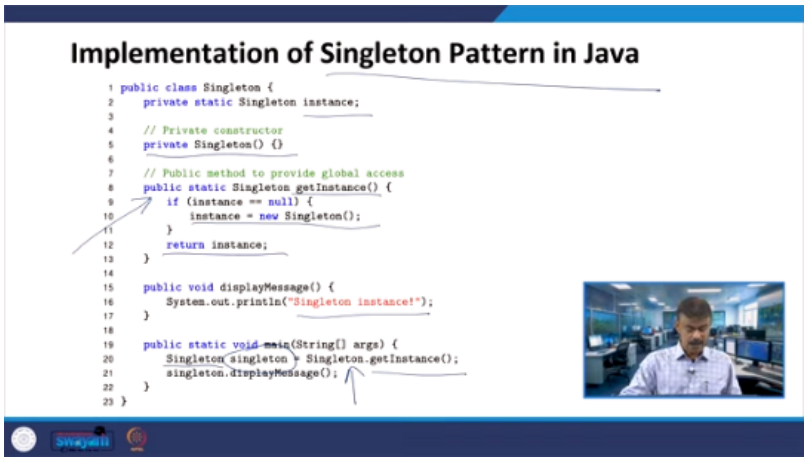
The similar or almost same program right of course the syntax will be different so we will see in the case of java all right so let us have public class singleton all right so here you have a private instance under singleton right so i am not worrying about star or address here so singleton instance so here i have a private constructor correct so the private constructor and here you go the public static alright the public static. So, under singleton class you have a member function get instance if instance is equal to null alright. So, you are creating a instance.

So, instance is equal to new of singleton which is a constructor in fact the private constructor. If the instance already exists you are returning the instance correct exactly same. whatever we have seen in C++ and here you have the singleton instance in Java, right. If it does not exist, we are creating the instance in line number 9 and 10. Otherwise, if it is already existing, you are returning it in line number 12.

So, the next one is public void display message. So, the display message, it is printing singleton instance. So, now if you go to the main program, I have an object singleton, right, under the class singleton, correct. So, the singleton, the class, right, so which is invoking get instance, right.

So, slightly modified version we had seen in C++, right. So, here this is invoking, right. I hope there the object is invoking. So, otherwise it is equivalent. The singleton class in the left hand side dot operator and here you have get instance.

So, when it is calling get instance, if the instance is not there, instance will be created and then it will be returned. Otherwise, if the instance already exists, it is like if you do not have a coffee machine, you are buying a coffee machine. Suppose I have a coffee machine in the office, I do not need to buy. So, if the instance in the class does not exist, You are creating the instance in line number 9 and 10.

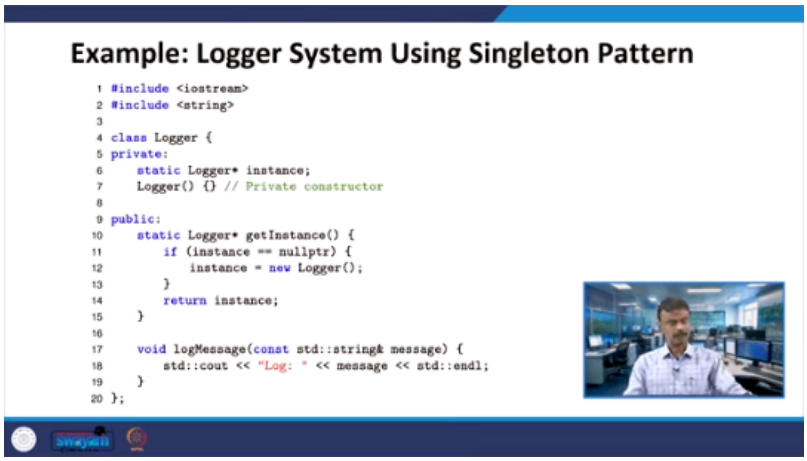


Otherwise, it will return the instance in line number 12. And line number 21, the singleton, the smallest, the object, which is invoking display message. So, when it is invoking display message, the singleton instance will be displayed. So, when

I run the code, I will get the output, the singleton instance. So, this is the case of Java.

I hope it is clear for everyone. So, here similar. So, I had the private constructor in the case of Java. So, this prevents direct instantiation of the class. So, I cannot directly instantiate an object.

So, that is why line number 5 you have the private constructor exactly same what we had seen in the case of C++. And here you have a static instance. So, that variable holds a single object and get instance which is creating and returning the instance if it does not exist otherwise it is providing the existing instance we had seen all right. So, it is very easy to implement and also it is ensuring consistent resource management. So, now we will see in fact we had seen all right there are certain applications.



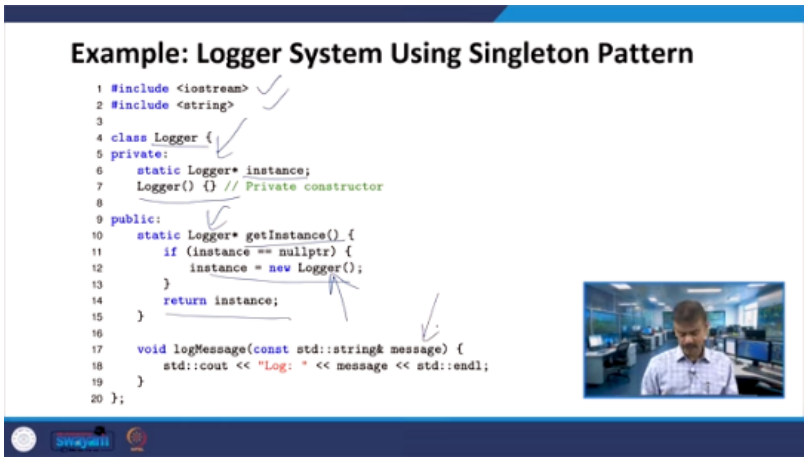
The logger system using singleton pattern all right. So, let us have the class logger and here you have a private right the object pointer object instance under logger. So, we have included IO stream and string right and then you are having the private constructor exactly like what we have seen in the case of singleton right almost similar. So, now if instance is equal to null pointer right. So, here you have the get instance member function under right.

So, this is a pointer function under logger, a pointer function under logger. If the instance is equal to a null pointer, if it is not pointing to anything, or in the case of Java, just now we had seen instance is equal to null, right. So, in that case, you are creating the instance, all right, where logger is the constructor. If it If the

instance already exists, right, it will return. If it does not exist, we are creating it exactly like what we have done in the case of a singleton class, right.

So, here we are creating an instance if it does not exist. So, that is happening in line number 11 and 12. If it is already existing, you are returning the instance, right. So, line number 17, you have a member function called log message. All right. So, under log message, you have a message.

All right. The reference message under string. Right. String address message. See out.



So log will be printed and whatever message you are going to pass. So that will be printing. So now I have. Right. So before main, the global bond.

So I have instance, it is a logger instance, pointer variable, okay. So now I am creating over here inside the main, I have logger, right, small l, right, creating a pointer object, all right, under logger. So in the right-hand side, get instance, right, this function, this member function is being called, all right. So when it is calling get instance, so if the instance does not exist, we are creating the instance. If it is already existing, we are returning it, all right.

So, otherwise the return, whatever be the case. The new instance, right, occurring, new instance is, right, you are getting, creating, it will be returned in line number 13, right, line number 14. And if it is already existing, it will return instance. So, instance will be returned, right. And then logger is invoking the, right, log message function, right.

You are pausing singleton logger. initiated all right singleton logger initiated yes it is being initiated here all right because it is returning so now this will call log message all right so log will be displayed log colon will be displayed and the message what is the message you have passed singleton logger initialized right singleton logger initialized will be printed and again line number 27 logger is invoking log message So logging and event will be printed, correct? So that means log colon logging and event will be printed, right? So this is what exactly happening.

So when I run the code, I will get the output like this. So singleton logger initialized and the next case logging and event, all right? So this will be printed. So this is all about the logger system. So the logger system using singleton pattern.

So So what it is saying, so it is ensuring, the logger system is ensuring a single instance for logging events throughout the application, right? Only one instance. And we are using the global access, right? So which is a get instance function, member function.

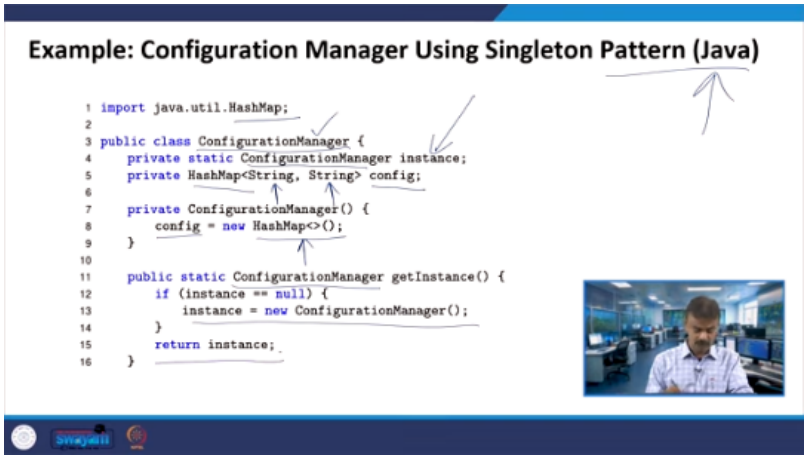
So, this is providing a unified entry point. And any component in the system can use the logger instance to record messages. That means you have flexibility. And the main advantage is it reduces redundant logger instantiations and also ensures consistent behavior across the application. So, now let us have the configuration manager using the singleton pattern.

So, this program we will see using Java, right? So, this program we will see using Java. So, Java utility HashMap, and if you look at the public class which is configuration manager, all right? So, private static, you have a configuration manager instance, all right? So here you are having The class configuration manager, and you have an object instance, right, exactly like what we had seen, right, in the Java program for the singleton, right.

And then you have the private class HashMap, right. So, you are having data type string and string. So, configuration, config, right, which is an object under HashMap. So, instance is an object under configuration manager, and config is an object under HashMap. So, now you have the private constructor configuration manager, right.

So, here you have it. So, config is correct. So, it is already the hashmap, right, which is equal to new hashmap, right? So, here you have the constructor, right, for the class hashmap. And here you see we have one getInstance member function.

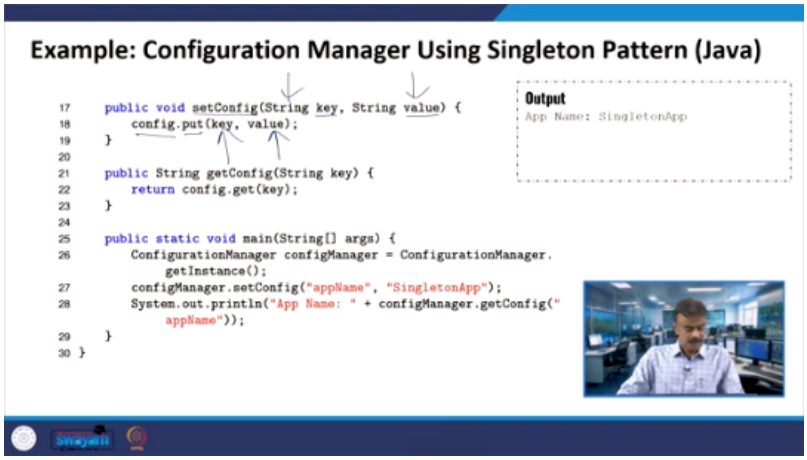
So, the getInstance member function under the ConfigurationManager class, right, under the ConfigurationManager class again, if the instance does not exist. So, we are creating an instance, right, otherwise return, right. So, even after creating, this will return the instance, as we had seen in the previous cases, a logger and singleton. So, now if you look. So, we have the setConfig method.



Right. So, here you have key and two attributes, key and value. Right. Two parameters, key and value, whose return type is string. Right.

So, two argument function whose data type is string. So, now your config object is invoking put. Right. Put. So, you are passing key and value.

So, key and value will be passed. Right. So, inbuilt function. like a puts and gets that we had studied in string, right. So, line number 21, you have get configuration method.



Here you are pausing, right, one argument key, right, one parameter key and it is returning. Here it is returning, right, the config will invoke get, right, and the key will be passed. So, now let us go to the main program. So, the main program in fact, it is a in the driver class is nothing, but the configuration manager itself right. So, driver class we do not have any other class.

So, it is under this driver class configuration manager. So, here in the configuration manager, I am creating one object: config manager. And on the right-hand side. So, this configuration manager is invoking get instance. Alright.

So, when get instance is invoked, we know that. If there is no instance. An instance will be created. Otherwise, if the existing instance. Line number 15.

It will be returned. Even after creating, line number 15 will work. It will return the instance. So now, the config manager object. Right.

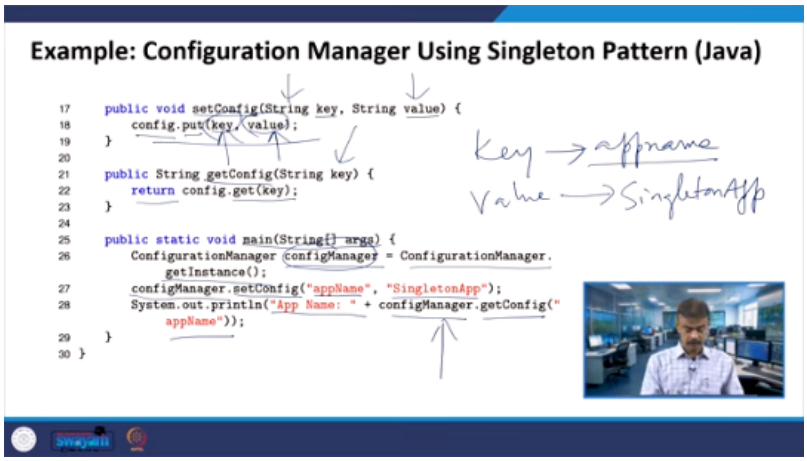
So this object is invoking set config. So set config. You are passing app name and singleton app, right? So that means the key is app name and singleton app is the value, right? This is what the put function will do.

And then you have one print statement: System.out.println. So this will print app name and config manager object. So which is invoking get config, right? So get config, you have app name. So that means your key is app name.

Key is what? App name. And what about the value? The value is nothing but what we already put. Can you tell me what the value is?

Singleton app. Right. So this is your value. Right. So both will be printed.

So app name, configuration manager. Right. So this will be printed like app name. In fact, this key will not be printed because config manager is invoking get config. Right. So get config, you are having get of key, right? So what is your key? Key is nothing but Singleton app, right? So when I run the code, I have to get the output like this, right? App name Singleton app. This is what we are expecting, and we get the output, right? So this is how the configuration manager using



Singleton pattern in Java is working, right? So the main idea is the configuration manager, which is maintaining the application-wide settings in a single instance. Yeah, I keep on talking about the single instance, and there are several features: the centralized storage for the configuration values and similarly consistent access to settings throughout the application. So the main usages are Developers use set config function to define settings and use get config to retrieve values from anywhere in the application, all right. So, in the next class, we will talk about the alternative singleton implementation in C++, all right. So, in today's lecture, So, we were talking about the single instance, all right, with the help of coffee machines, and then we had seen three different programs, all right, the mixture of Java and C++, all right.

So, in the next class, I will start with the alternative singleton implementation in C++. Thank you.