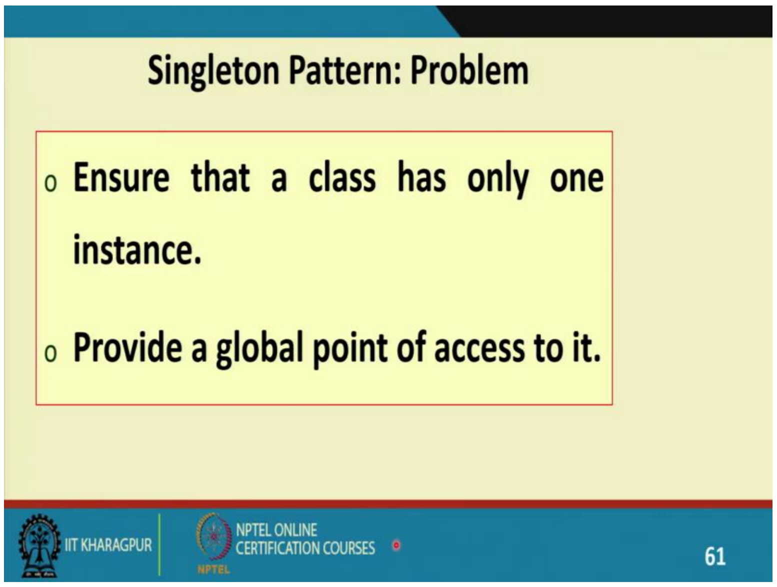
Object Oriented System Development Using UML, Java and Patterns

Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Management, Kharagpur
Lecture 46
Singleton Pattern- II

Welcome to this session! In the last session we were discussing about the singleton pattern, very simple pattern but very useful pattern, let us just review what we had discussed and we will continue from that point onwards.

(Refer Slide Time: 00:34)



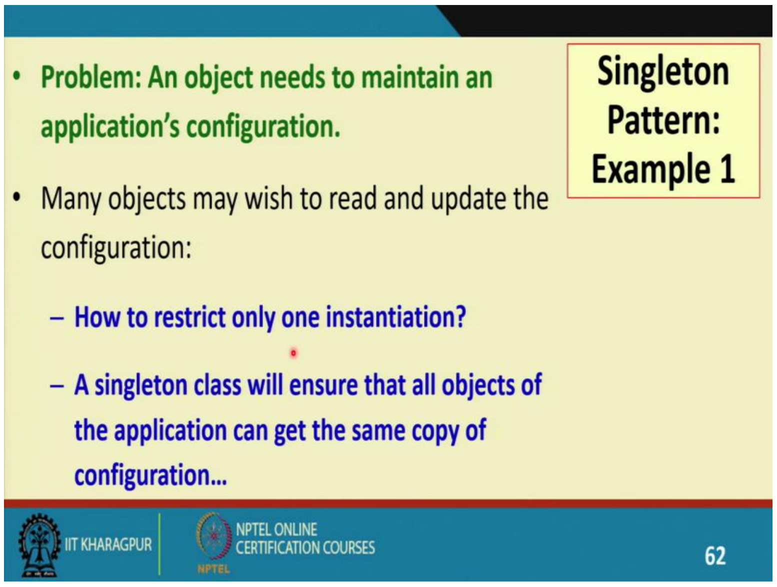
Singleton Pattern: Problem

- Ensure that a class has only one instance.
- Provide a global point of access to it.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 61

The singleton pattern problem occurs in many situations that we want exactly one object of a class, exactly one object of a class and also all other classes need to access this single object. So that is the context in which the problem occurs that we need a single object somehow we must restrict creation to exactly one object and prevent instantiation of multiple objects and provide global point of reference, so that all classes can access this object.

(Refer Slide Time: 01:21)



- **Problem: An object needs to maintain an application's configuration.**
- Many objects may wish to read and update the configuration:
  - **How to restrict only one instantiation?**
  - **A singleton class will ensure that all objects of the application can get the same copy of configuration...**

Singleton Pattern: Example 1

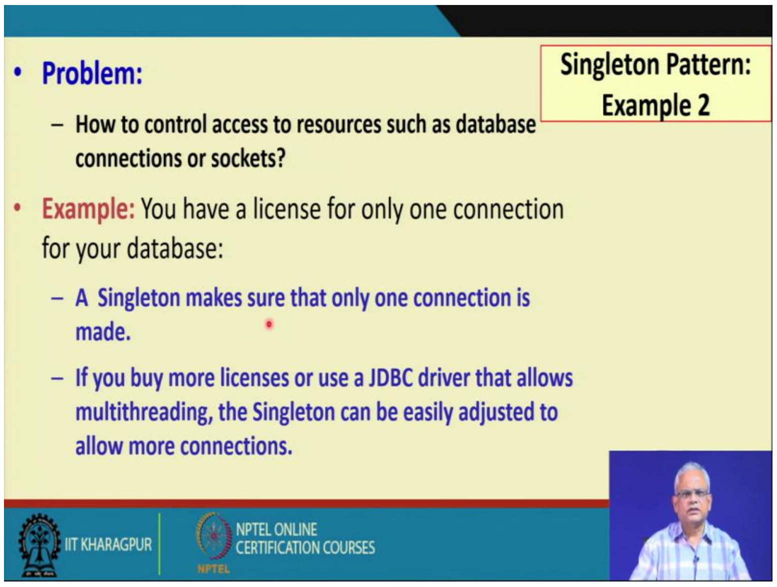
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 62

We will give few examples where the singleton pattern occurs, so that in a problem that we are trying to solve we can determine when to use the singleton pattern. Let us look at this problem that configuration of a gaming application. The gaming application can be played at various levels preliminary level, intermediate level, expert level and so on and the features of the game, the rules of the game are different for different level that is set in the configuration.

The current configuration is saved in a single object. The single object has all the rules that for a given level of the game what are the things that needs to be done, and therefore various objects of the game need to access this configuration object. The problem now is that we want to restrict to exactly one configuration object because multiple configuration object will be a problem; the game will become inconsistent and so on and also the global point of access that all other objects in the gaming application should be able to access it, how do we go about?

We will use the singleton pattern in this situation, it will restrict creation of a single object and all other objects can access this object, the exact solution, how to create this object? How is the singleton pattern structure instantiation and so on, that we will see a little later. First we are looking at example applications where this pattern is useful or applicable.

(Refer Slide Time: 03:38)


Singleton Pattern: Example 2

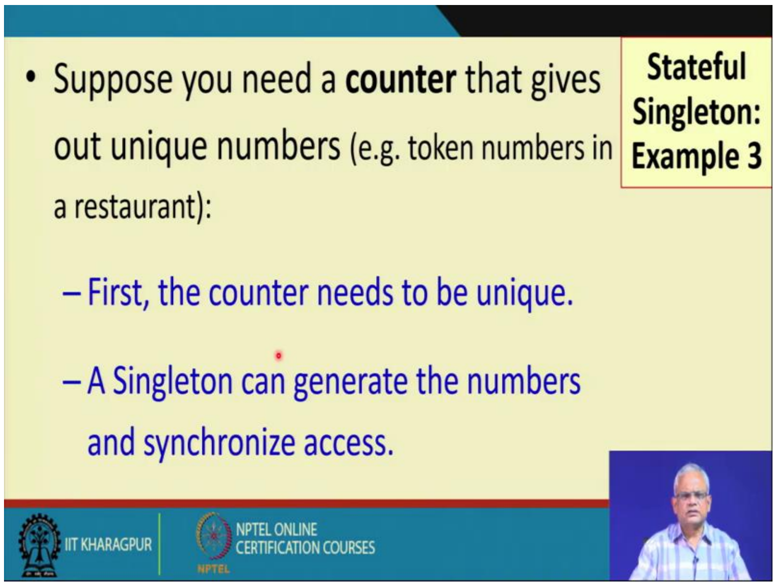
- **Problem:**
  - How to control access to resources such as database connections or sockets?
- **Example:** You have a license for only one connection for your database:
  - A Singleton makes sure that only one connection is made.
  - If you buy more licenses or use a JDBC driver that allows multithreading, the Singleton can be easily adjusted to allow more connections.

Another example of a singleton pattern is that we have an application and the application needs to make a connection to the database. The database is a paid database, it is a commercial database and we have license of one connection and there are many objects in the application who will like to have open a connection to the database and get data from there. But because of the License Agreement, we should allow exactly one connection otherwise the connection may be refused or maybe there will be a heavy penalty if we have more than one connection to the database. And it is from the application point of view that we want to restrict the connection to be exactly 1.

In this situation we use the singleton pattern which will allow a single connection. All the application objects who need a connection to the database will first request the singleton for the connection. In other words the singleton manages the connection to the database and if the connection is not taken by another object it will give the connection on request otherwise it will refuse and ask the object to it.

But then the problem is that we must have exactly one singleton object here, otherwise again there will be multiple connections that might be occurring.

(Refer Slide Time: 05:31)



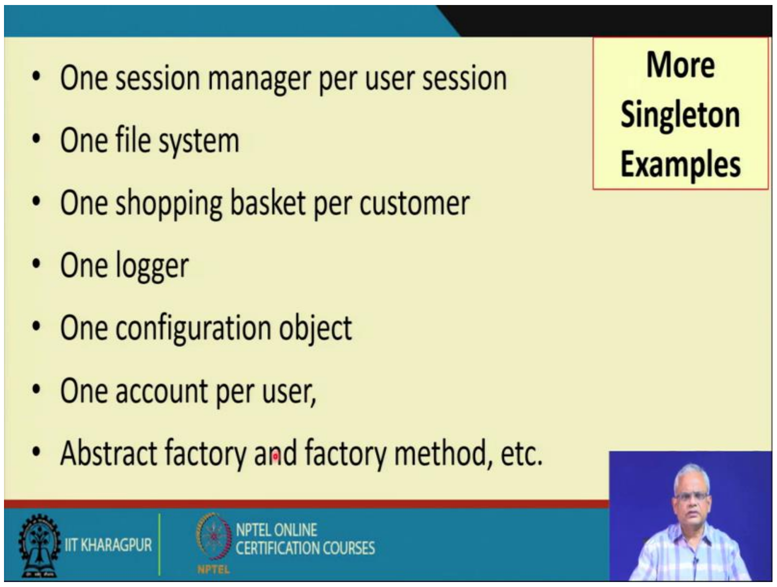
Stateful Singleton: Example 3

- Suppose you need a **counter** that gives out unique numbers (e.g. token numbers in a restaurant):
  - First, the counter needs to be unique.
  - A Singleton can generate the numbers and synchronize access.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Another example, let us say we have a restaurant where there are different sections of the restaurant, let us say there is a section on Indian menu, there is another one European, another one just fast-food and so on. Now, different clients and different sections of the restaurant, they order but then the token number given to these different sections is to be unique because finally the kitchen will prepare the order and the order number has to be unique. Therefore, there has to be exactly one counter object which will generate the token numbers based on the token number it should be possible to identify exactly which customer and what is the order and so on. If there is a duplication of the counter object, then there will be going to be real chaos and we can use the singleton pattern here. The counter object is to be just a single unique object and then this singleton object the responsibility will be to generate the numbers and all objects in the restaurant application can access the singleton object.

(Refer Slide Time: 07:14)



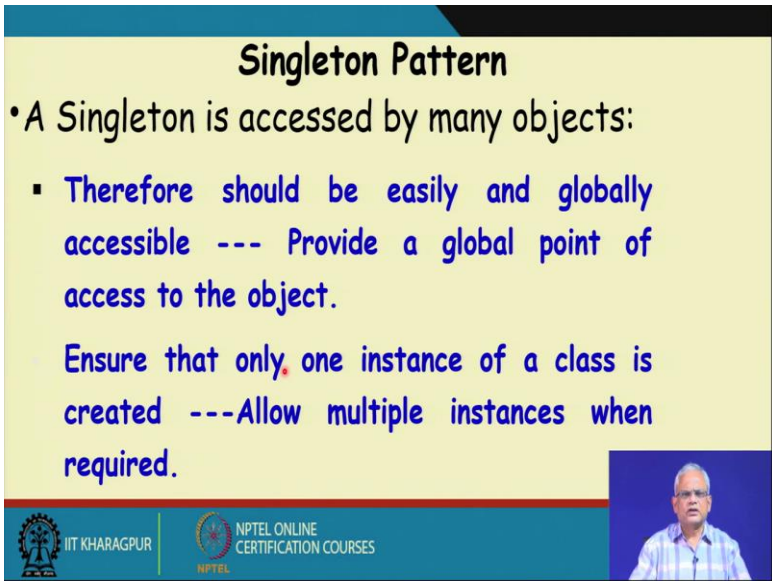
More Singleton Examples

- One session manager per user session
- One file system
- One shopping basket per customer
- One logger
- One configuration object
- One account per user,
- Abstract factory and factory method, etc.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

There are many more applications where such singletons occur, just a few written here but then in application that you are likely to develop, you might also very likely get a situation where you can use the singleton pattern. One session manager per session, one file system, one shopping basket per customer in an E-commerce application, one logger, one configuration object, one account per user. Abstract factory and factory method these two patterns we will see little later and these two also make use of the singleton pattern.

(Refer Slide Time: 08:01)



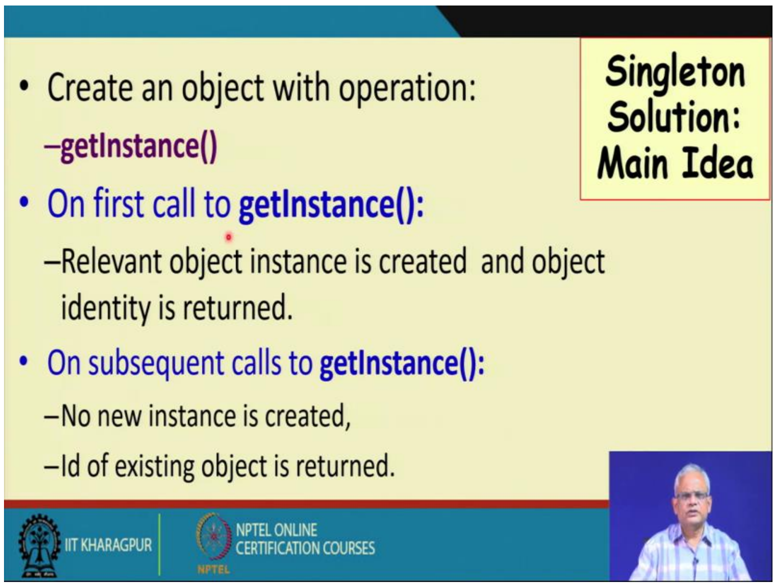
Singleton Pattern

- A Singleton is accessed by many objects:
  - Therefore should be easily and globally accessible --- Provide a global point of access to the object.
  - Ensure that only one instance of a class is created ---Allow multiple instances when required.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Now let us look at the overall idea here, that we have a singleton object which will be accessed by many other objects, therefore the singleton should be accessible globally by various objects and also once they access this object it has to be unique, in no case multiple instances of singleton should be created, ofcourse we will see a situation where exactly two or exactly three singleton objects may be created. We call it as multi-ton and that is a variation of the singleton pattern. The singleton as it is guarantees that there will be exactly one object. But then we will see multi-ton where we relax this and say that there will be exactly two objects or maybe exactly three objects no more than that and so on.

(Refer Slide Time: 09:04)



Singleton Solution: Main Idea

- Create an object with operation:
  - getInstance()
- On first call to **getInstance():**
  - Relevant object instance is created and object identity is returned.
- On subsequent calls to **getInstance():**
  - No new instance is created,
  - Id of existing object is returned.

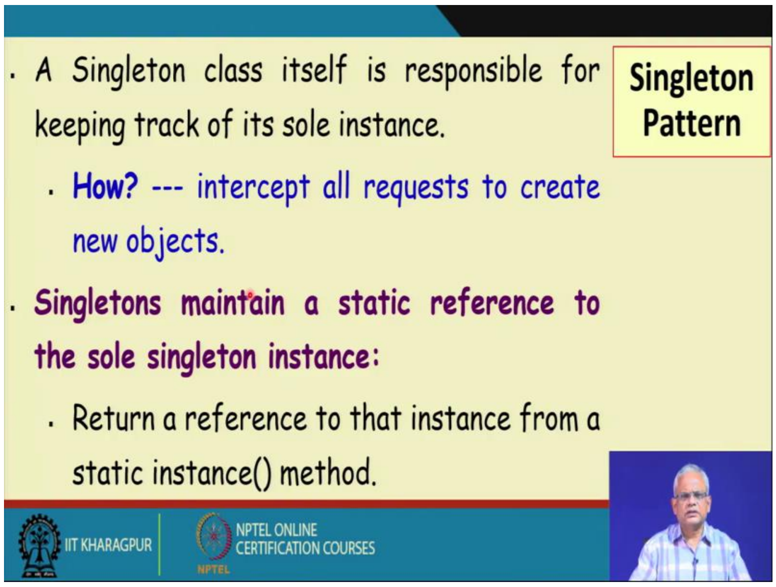
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Let us look at the main idea here, the main idea here is that the constructor of the singleton we make it private, so that no other object can instantiate a singleton the only way an instance can be created or access can be obtained is by the method of the singleton getInstance. All other objects who need the instance of the singleton are reference to the singleton they call the getInstance method.

The getInstance method on the first call to the getInstance, it creates the single object and keeps track of the object itself, we had so far seen that when objects are created it is the aggregator who keeps track of those objects but here in the singleton the singleton itself the getInstance method of the class itself keeps track of the singleton.

On the subsequent call to the getInstance, the first call the singleton gets created on subsequent calls it just returns the ID or the reference to the singleton that was created and no new instance is created and because all the creation requests and reference requests are routed through the getInstance method therefore it is possible to maintain a single instance of this singleton.

(Refer Slide Time: 10:52)



Singleton Pattern

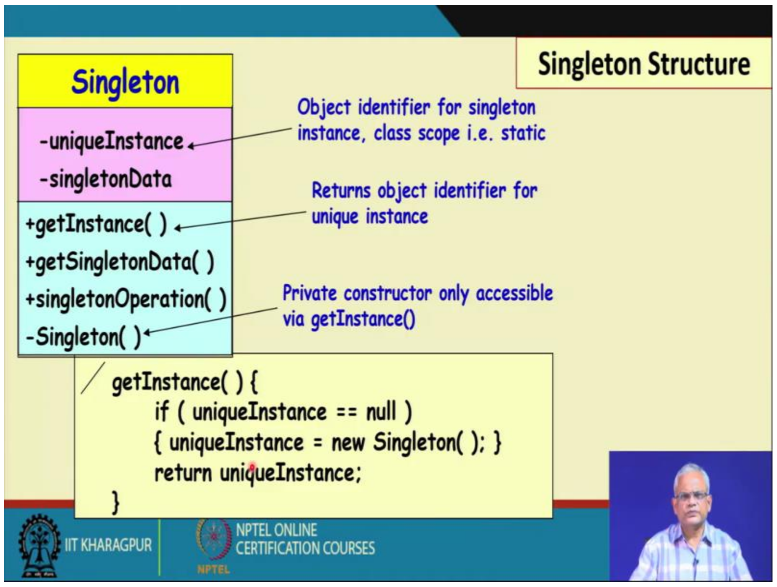
- A Singleton class itself is responsible for keeping track of its sole instance.
  - How? --- intercept all requests to create new objects.
- Singletons maintain a static reference to the sole singleton instance:
  - Return a reference to that instance from a static instance() method.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

As we were mentioning that in contrast to other classes here the singleton itself creates and maintains its own ID. In the example of a book aggregation the book register used to create the book and keep the ID with itself but then here in the singleton objects the singleton itself creates it and keeps the ID itself. It intercepts all requests to get the reference and to create through the getInstance method, no one can call the create new cannot be called on singleton because the constructor is private.

And as we will see, that the reference is maintained within the singleton as a static variable. Static variable maintains the reference to the instance once it is created and if there are subsequent calls to the getInstance it is returned based on what is stored in the static variable.

(Refer Slide Time: 12:11)



Singleton Structure

Singleton

-uniqueInstance
-singletonData

+getInstance( )
+getSingletonData( )
+singletonOperation( )
-Singleton( )

Object identifier for singleton instance, class scope i.e. static

Returns object identifier for unique instance

Private constructor only accessible via getInstance()

```
getInstance( ) {
      if ( uniqueInstance == null )
      { uniqueInstance = new Singleton( ); }
      return uniqueInstance;
}
```

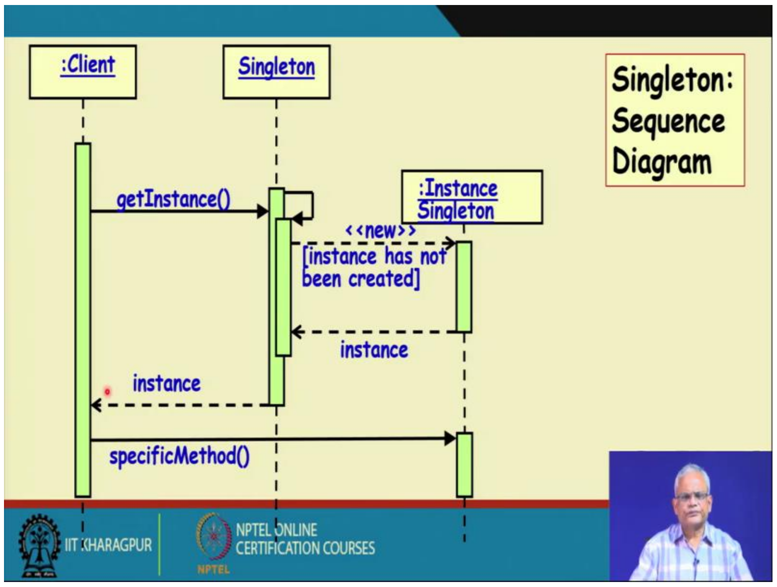
Now let us look at the class structure of a singleton, one is that we have a private variable unique instance this is the static private variable which maintains the instance of the singleton object and getInstance is the one which all clients of the singleton call and observe that the constructor is a private here.

And then the singleton might have some responsibilities, it may offer some operations or data. For example, for a game configuration singleton it might provide data about the exact configuration of the game or maybe it will allow the connection to a database and so on. These are the data and operations these are the role of the singleton.

But then two main methods here which ensured that there is a single object created is by making the constructor private and all requests to the singleton is through the getInstance method. The gate instance method is static and the gate instance returns the unique ID of the singleton object and the constructor is private and it is accessible only within the class by the gate instance.

The gate instance can only call singleton, the outside objects cannot call the new on the singleton object because the constructor is private. Now let us look at the gate instance method. In the gate instance method, if the instance has not been created that is unique instance is null, we call the constructor here and get create the instance and store that in the unique instance and in all other cases if the unique instance is not null, it just returns the unique instance, that is the code for singleton getInstance of the singleton and a very simple code, very simple ideas but then very useful.

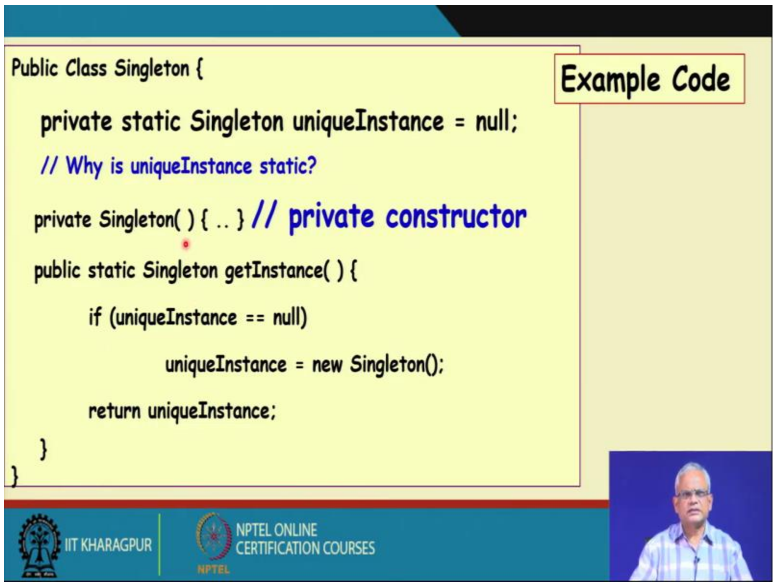
(Refer Slide Time: 14:58)



Singleton: Sequence Diagram

If we look at the sequence diagram, the client of the singleton calls getInstance there may be many clients calls this getInstance. The getInstance checks if the unique ID is null. If it is null, then it creates the instance through the new operation and gets the new instance created and the instance is written and the instance is given to the client and then the client can call specific method. On further requests, if the instance already exists then it just returns the instance that is the sequence diagram for the singleton.

(Refer Slide Time: 15:53)



Example Code

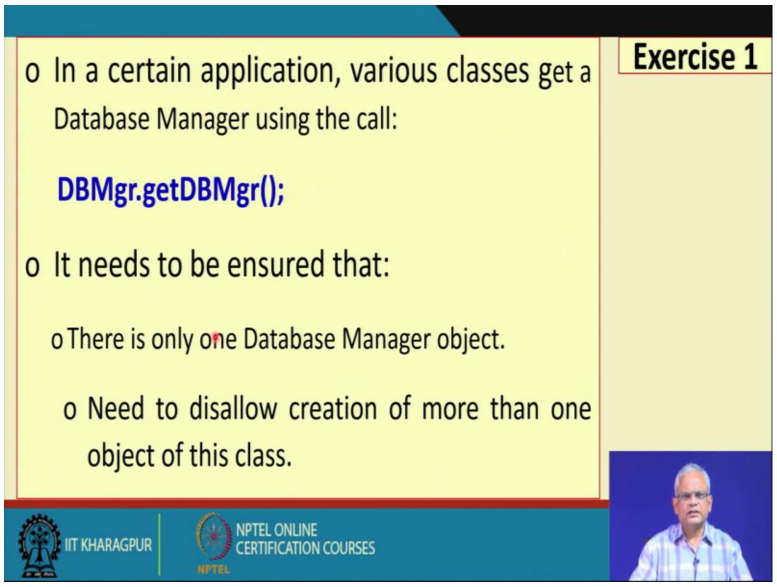
```
Public Class Singleton {
    private static Singleton uniqueInstance = null;
    // Why is uniqueInstance static?
   private Singleton( ) { .. } // private constructor
   public static Singleton getInstance( ) {
        if (uniqueInstance == null)
                uniqueInstance = new Singleton();
        return uniqueInstance;
    }
}
```

Now this is an example code, this is the singleton class and then the unique instance we make it null initially, this is a private static singleton, but then the question is that why is it that we need to make the unique instance static? Let us try to answer this question as we go through the other methods.

The singleton constructor is private no other outside object external objects can instantiate the singleton and then the gate instance returns a reference it is a static because even when no object has been created using the static method, the method can be accessed even when there is no instance of the singleton object. And then check the unique instance is null. If it is null, create the singleton and return unique instance and if it is not null, do not create it and we just return the unique instance, but then the question still remains that why is that the unique instance need to be static? For the getInstance, it need to be static because the method will be called even when there is no instance of the singleton object exists.

Now that answers the question that why the unique instance need to be static, because a static method can only access the static variables and since it is accessing the unique instance and therefore unique instance must be static. A static method cannot access non-static variables because those ones get created after an instance is created. The static exists even when there is no instance created.

(Refer Slide Time: 18:27)



Exercise 1

- In a certain application, various classes get a Database Manager using the call:

  DBMgr.getDBMgr();

- It needs to be ensured that:
  - There is only one Database Manager object.
    - Need to disallow creation of more than one object of this class.

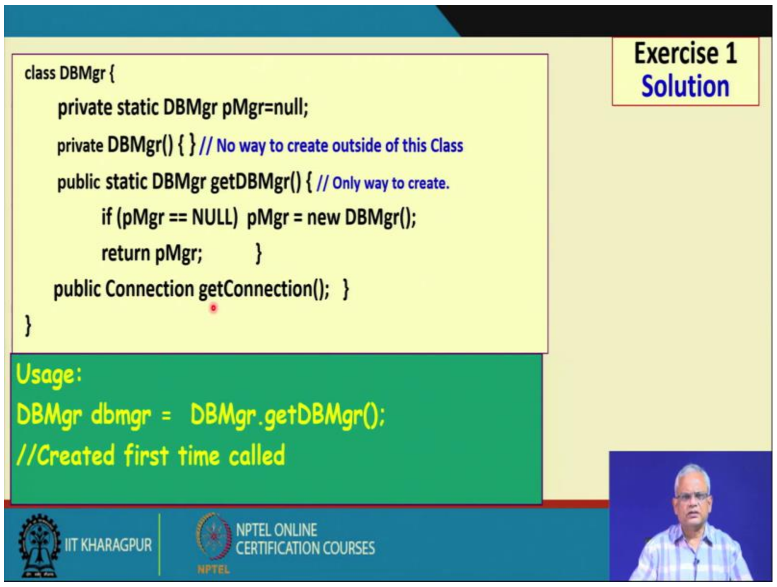
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Now let us try to do one exercise on the singleton. Now let us say for a certain application, we want a singleton object which we will call as DBMgr, it manages the database connection and the clients can instantiate using get DBMgr and then it allows the connections, the requestor connection to the DBMgr object.

But then we need to have exactly one instance of the DBMgr object because multiple instances of DBMgr if by chance they get created it will defeat the whole purpose of having a singleton and there can be multiple connections open because different DBMgrs will allow connections. So here, we need only one instance of the DBMgr which we will get by calling DBMgr.getDBMgr and then we can request for connection.

Now, how do we write the code for this DBMgr class which is basically a singleton? Based on the code that we have the pattern code we can adapt here and write the DBMgr code can first attempt to do this based on the code that we had written for an example singleton pattern but if you try, then compare with the solution that we give here.

(Refer Slide Time: 20:21)



Exercise 1
Solution

```
class DBMgr {
      private static DBMgr pMgr=null;
      private DBMgr() { } // No way to create outside of this Class
      public static DBMgr getDBMgr() { // Only way to create.
            if (pMgr == NULL)  pMgr = new DBMgr();
            return pMgr;         }
      public Connection getConnection();  }
}
```

Usage:

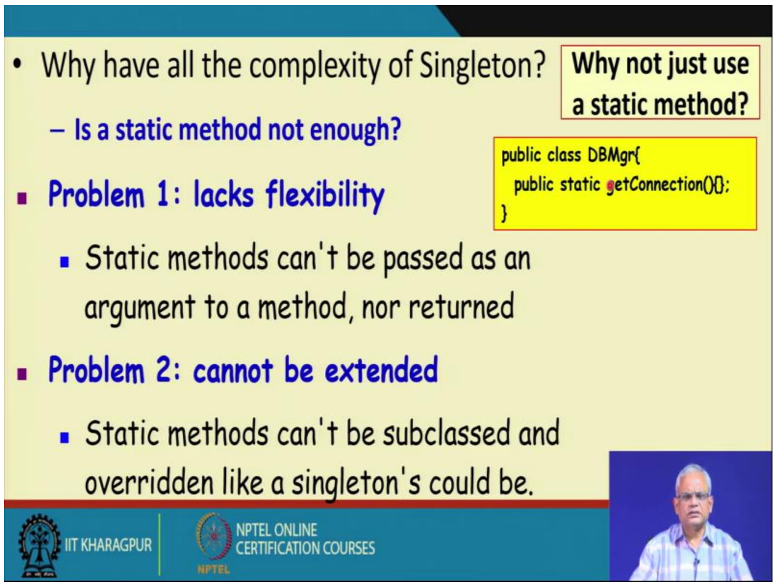
```
DBMgr dbmgr =  DBMgr.getDBMgr();
//Created first time called
```

The uses of this will be that the clients getInstance of DBMgr by calling DBMgr dot get DBMgr, DBMgr is the singleton class and then based on the DBMgr reference they can request for connection. Now this is the code for the class DBMgr we have private static instance of the DBMgr which we name is P manager private manager which is null to start with.

And then the constructor we make it private and then the important method here is the get DBMgr all other objects they call the gate DBMgr. And in the DBMgr first we check if the P manager is null, then we create new DBMgr and keep the reference in the P manager.

Otherwise, just return the P manager and then it has the method get connection and the single DBMgr keeps track of the connection to get connection and return connection and if the connection is not taken by another object, it will approve the connection otherwise it will reject the connection. So that is the code for the DBMgr singleton application.

(Refer Slide Time: 22:03)



- Why have all the complexity of Singleton?
  - Is a static method not enough?
- **Problem 1: lacks flexibility**
  - Static methods can't be passed as an argument to a method, nor returned
- **Problem 2: cannot be extended**
  - Static methods can't be subclassed and overridden like a singleton's could be.

Why not just use a static method?

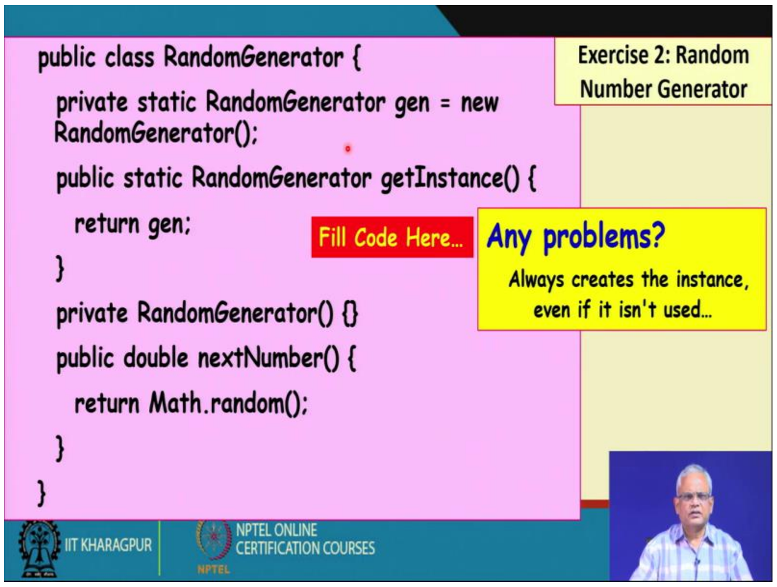
```
public class DBMgr{
  public static getConnection(){};
}
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

But then one insight into the singleton cannot we just have a class with a static method which is just does whatever necessary like get connection etc, why do we need a singleton object? We need not even create an instance of the object and the different clients just call the get static connection, the get connection. Just a static method, cannot it be sufficient? The answer is that there will be several problems if you try to do this. For some situation it may work that you have like a very simple situation you have a class with a static method and then that manages the connection. So no objects are created here but then the problems will be lack of flexibility, we cannot really pass a class like this as argument into another class.

Whereas object reference singleton reference can be passed and also if this is the situation that we have a class and just a static method we cannot extend it and add new functionality. So in many situations we need to extend the singleton to do slightly different work and if we just have a static method we cannot really extend and override the methods of the singleton.

(Refer Slide Time: 24:00)



Exercise 2: Random Number Generator

```
public class RandomGenerator {
  private static RandomGenerator gen = new
  RandomGenerator();
  public static RandomGenerator getInstance() {
    return gen;
  }
  private RandomGenerator() {}
  public double nextNumber() {
    return Math.random();
  }
}
```

Fill Code Here...

Any problems?

Always creates the instance, even if it isn't used...

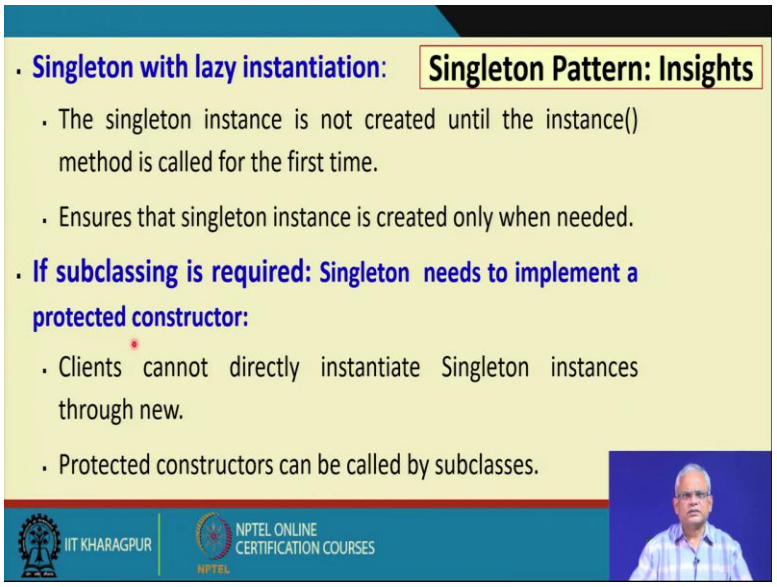
Now let us try to solve this problem of the random number generator. For a restaurant we have this random number generator class which generates a token number and then we want to write the singleton class definition here we have filled some part of the class definition of written public class RandomGenerator and then we have made the constructor private.

Now, there are other methods of the RandomGenerator like next number etc., which also you have written but now we need to write the code here for the getInstance, already the constructor is private new we cannot generate an object using the new. Now we need to have a getInstance using which all clients can try to get the reference of this if the reference already exists it will be written otherwise it will be created and written.

So please try to write the code here and here the code that is required is that we have the private static random number generator. So here we generate new random number generator so in the reference in this code this is slight variation of the code that we have seen here seen earlier here during instantiation itself during the class loading we are already creating an instance and then we just return the instance.

Here even before the first request comes the instance is generated and then each time only the reference to the singleton object created but then this code sometimes is not really very good because the instance is getting created even when we do not have any use for the singleton object always the instance is getting created.

(Refer Slide Time: 26:44)



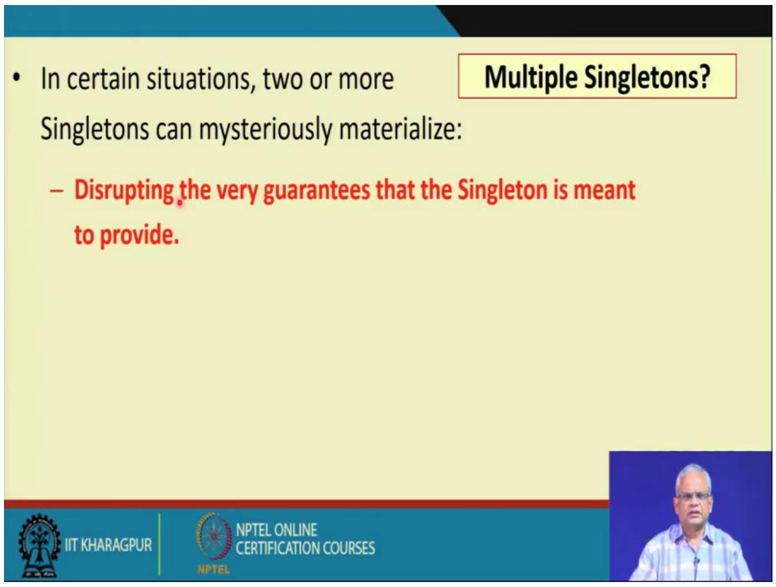
Singleton Pattern: Insights

- **Singleton with lazy instantiation:**
  - The singleton instance is not created until the instance() method is called for the first time.
  - Ensures that singleton instance is created only when needed.
- **If subclassing is required: Singleton needs to implement a protected constructor:**
  - Clients cannot directly instantiate Singleton instances through new.
  - Protected constructors can be called by subclasses.

But maybe it is a good idea to create the object only when the first request comes and that we call as lazy instantiation. In the lazy instantiation, only when the first request comes the singleton object is created and for the subsequent requests the reference is returned. The other insight that we need to consider is that when sub classing of the singleton is required.

In this case, we cannot make the constructor private because then the derived classes it will be difficult to instantiate the singleton and therefore we need to make it protected and in this situation after sub-classing the derived classes can create the singleton instance.

(Refer Slide Time: 27:46)



Multiple Singletons?

- In certain situations, two or more Singletons can mysteriously materialize:
  - Disrupting the very guarantees that the Singleton is meant to provide.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

But what about multiple singletons? Is it possible that the code that we had discussed so far is it possible that by any chance multiple singletons will appear we had taken all precaution and made the constructor private all the access was through the gate instance and so on but is it possible that multiple singletons can materialize?

Yes, in some situations the code that we have so far shown or discussed multiple singletons can appear and therefore the very guarantee that the singleton object class provides will be disrupted. Now, we are almost at the end of this class.

In the next session, we will look at the situation when the code that we had shown is insufficient to guarantee a singleton object, multiple singletons can materialize and we will refine the code that had discussed so far to prevent the occurrence of multiple singletons. Thank you.