***WHAT IS SINGLETON DESIGN PATTERN****?s*

It is one of the simplest design pattern in terms of the modelling but on the other hand this is one of the most controversial pattern in terms of complexity of usage.

Class is defined in such a way that only one instance of class is created in the complete execution of program or project.

**Singleton Design Pattern**

https://cdncontribute.geeksforgeeks.org/wp-content/uploads/SINGLEton.png

An implementation of singleton class should have following properties:

1. **It should have only one instance :** This is done by providing instance of class from within the class. Outer classes or subclasses should be prevented to create the instance. This is done by making the constructor private in java so that no class can access the constructor and hence cannot instantiate it.
2. **Instance should be globally accessible :** Instance of singleton class should be globally accessible so that each class can use it. In java it is done by making the access-specifier of instance public.

***TYPES OF SINGLETON DESIGN PATTERN:***

Singleton class can be instantiated by two methods:

 **Early initialization :** In this method, class is initialized whether it is to be used or not. Main advantage of this method is its simplicity. You initiate the class at the time of class loading. Its drawback is that class is always initialized whether it is being used or not.

 **Lazy initialization :** In this method, class in initialized only when it is required. It can save you from instantiating the class when you don’t need it. Generally lazy initialization is used when we create a singleton class.

***HOW TO DO THAT(IMPLEMENTATION OF SINGLETON DESIGN PATTERN):***

**Eager initialization:**

Here we have created instance of singleton in static initializer. JVM executes static initializer when the class is loaded and hence this is guaranteed to be thread safe. Use this method only when your singleton class is light and is used throughout the execution of your program.

|  |
| --- |
| // Java code to create singleton class by  // Eager Initialization  public class GFG  {    // public instance initialized when loading the class    public static GFG instance = new GFG();      private GFG()    {      // private constructor    }  } |

 **Pros:**

1. Very simple to implement.
2. No need to implement getInstance() method. Instance can be accessed directly.
3. Thread safe

**Cons:**

1. May lead to resource wastage. Because instance of class is created always, whether it is required or not.
2. CPU time is also wasted in creation of instance if it is not required.
3. Exception handling is not possible.

 **Using static block:** This is also a sub part of Eager initialization. The only difference is object is created in a static block so that we can have access on its creation, like exception handling. In this way also, object is created at the time of class loading.  
It can be used when there is a chance of exceptions in creating object with eager initialization.

|  |
| --- |
| // Java code to create singleton class  // Using Static block  public class GFG  {    // public instance    public static GFG instance;      private GFG()    {      // private constructor    }      {      // static block to initialize instance      instance = new GFG();    }  } |

 **Pros:**

1. Very simple to implement.
2. No need to implement getInstance() method. Instance can be accessed directly.
3. Exceptions can be handled in static block.

**Cons:**

1. May lead to resource wastage. Because instance of class is created always, whether it is required or not.
2. CPU time is also wasted in creation of instance if it is not required.

 **Lazy initialization:** In this method, object is created only if it is needed. This may prevent resource wastage. An implementation of getInstance() method is required which return the instance. There is a null check that if object is not created then create, otherwise return previously created. To make sure that class cannot be instantiated in any other way, constructor is made final. As object is created with in a method, it ensures that object will not be created until and unless it is required. Instance is kept private so that no one can access it directly.  
It can be used in a single threaded environment because multiple threads can break singleton property because they can access get instance method simultaneously and create multiple objects.

|  |
| --- |
| //Java Code to create singleton class  // With Lazy initialization  public class GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method    private static GFG instance;      private GFG()    {      // private constructor    }      //method to return instance of class    public static GFG getInstance()    {      if (instance == null)      {        // if instance is null, initialize        instance = new GFG();      }      return instance;    }  } |

 **Pros:**

1. Object is created only if it is needed. It may overcome resource overcome and wastage of CPU time.
2. Exception handling is also possible in method.

**Cons:**

1. Every time a condition of null has to be checked.
2. instance can’t be accessed directly.
3. In multithreaded environment, it may break singleton property.

 **Thread Safe Singleton:** A thread safe singleton in created so that singleton property is maintained even in multithreaded environment. To make a singleton class thread-safe, getInstance() method is made synchronized so that multiple threads can’t access it simultaneously.

|  |
| --- |
| // Java program to create Thread Safe  // Singleton class  public class GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method    private static GFG instance;      private GFG()    {      // private constructor    }     //synchronized method to control simultaneous access    synchronized public static GFG getInstance()    {      if (instance == null)      {        // if instance is null, initialize        instance = new GFG();      }      return instance;    }  } |

 **Pros:**

1. Lazy initialization is possible.
2. It is also thread safe.

**Cons:**

1. getInstance() method is synchronized so it causes slow performance as multiple threads can’t access it simultaneously.

 **Lazy initialization with Double check locking:** In this mechanism, we overcome the overhead problem of synchronized code. In this method, getInstance is not synchronized but the block which creates instance is synchronized so that minimum number of threads have to wait and that’s only for first time.

|  |
| --- |
| // Java code to explain double check locking  public class GFG  {    // private instance, so that it can be    // accessed by only by getInstance() method    private static GFG instance;      private GFG()    {      // private constructor    }      public static GFG getInstance()    {      if (instance == null)      {        //synchronized block to remove overhead        synchronized (GFG.class)        {          if(instance==null)          {            // if instance is null, initialize            instance = new GFG();          }          }      }      return instance;    }  } |

 **Pros:**

1. Lazy initialization is possible.
2. It is also thread safe.
3. Performance reduced because of synchronized keyword is overcome.

**Cons:**

1. First time, it can affect performance.

As cons. of double check locking method is bearable so it can be used for high performance multi-threaded applications.

 **Bill Pugh Singleton Implementation:** Prior to Java5, memory model had a lot of issues and above methods caused failure in certain scenarios in multithreaded environment. So, Bill Pugh suggested a concept of inner static classes to use for singleton.

|  |
| --- |
| // Java code for Bill Pugh Singleton Implementaion  public class GFG  {      private GFG()    {      // private constructor    }      // Inner class to provide instance of class    private static class BillPughSingleton    {      private static final GFG INSTANCE = new GFG();    }      public static GFG getInstance()    {      return BillPughSingleton.INSTANCE;    }  } |

When the singleton class is loaded, inner class is not loaded and hence doesn’t create object when loading the class. Inner class is created only when getInstance() method is called. So it may seem like eager initialization but it is lazy initialization.  
This is the most widely used approach as it doesn’t use synchronization.

***WHEN WE NEED SINGLETON DESIGN PATTERN?***

1) It is used where only a single instance of class is required to control the action throughout the execution.

2) Singleton classes are used for logging, driver objects, caching and thread pool, database connections.

3) A single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly. Similarly, there can be a single configuration manager or error manager in an application that handles all problems instead of creating multiple managers.