SINGLETON DESIGN PATTERN

Wednesday, August 23, 2023 4:10 PM

Approach 1

```
public class Singleton {
   private static Singleton uniqueInstance;
   // other useful instance variables here
   private Singleton() {}
   public static Singleton getInstance() {
      if (uniqueInstance == null) {
         uniqueInstance = new Singleton();
      }
      return uniqueInstance;
   }
   // other useful methods here
}
```

Drawbacks

- Thread safety vulnerability.
  - So question arise How to make it thread Safe?
    - For thread safety we come up with Approach 2.

Approach 1 working

- we have created an instance of class but haven't initialized object to it.
- our objective is to only allow creation of one object of class.
- for that we would be making our constructor private so that no Object creation be done through default constructor.
- we would only allow object creation through getInstance() method.
  - In it we would check if class instance has not been initialised yet. then only we would allow object creation of it. else we would return old object itself.

Approach 2 Using Synchronised keyword

Approach 2 — Using Synchronised keyword

```
c
public class Singleton {
   private static Singleton uniqueInstance;

   // other useful instance variables here
   private Singleton() {
   }

   public static synchronized Singleton getInstance() {
      if (uniqueInstance == null) {
         uniqueInstance = new Singleton();
      }
      return uniqueInstance;
   }
// other useful methods here
}
```

Drawbacks

→ It would decrease throughput of our func / API / Service bcz only one thread would be able to access method at given point of time rest would be in waiting state hence decreasing throughput.

working of it

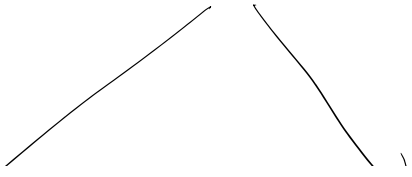
Synchronised

→ Allows only one thread to access that object at once keeping other thread in waiting state.

Since only one thread can access at given time so no case will arise where multiple thread have multiple instance at given time.

This feature is of critical use in Banking Systems & payment gateway.

Question arise : How to increase its throughput then ?



Double locking

Eager instantiation
[Resource intensive]

Approach 3 (Eager Instantiation)

```
public class Singleton {
  private static Singleton uniqueInstance = new Singleton();

  private Singleton() {
  }

  public static Singleton getInstance() {
    return uniqueInstance;
  }
}
```

Drawbacks

→ Resource intensive : Bcz obj get created even if its use case never arise in complete code execution.

working

→ we have created object while we are declaring it and we return same object whenever call getInstance() by any number of threads.

Approach 4 (Double locking)

The volatile keyword ensures that multiple threads handle the uniqueInstance variable correctly when it is being initialized to the Singleton instance.

```
public class Singleton {
  private volatile static Singleton uniqueInstance;

  private Singleton() {
  }

  public static Singleton getInstance() {
    if (uniqueInstance == null) {
      synchronized (Singleton.class) {
        if (uniqueInstance == null) {
          uniqueInstance = new Singleton();
        }
      }
    }
    return uniqueInstance;
  }
}
```

working

To check for if object is already being initialized or not. If not then only enters this block. If comes to getInstance()

```
    }
    return uniqueInstance;
  }
}
```

If Not then only ...

ex:- 10 threads comes to getInstance

At Time T1

→ 2 threads comes.

checks if Object is null.

yes it is.

then one thread Enters Synchronised block gets instantiated after second lock check.

now second thread will come to Synchronised Block.

Here 2nd lock will itself block it from object instantiation.

Hence doing work of singleton class.

But we now have 8 threads coming to hit getInstance() at different Time.

Here comes role of lock 1st (External lock).

Now External lock gets uniqueInstance variable as not null. Hence Block all 8 threads at once and hence not keeping 8 threads in waiting Stage as in approach 2.

But these 8 threads will not get chance to reach till synchronised Block.

...reasing our throughput ...t being

Hence increasing our throughput to large extend without being CPU / Resource intensive as in approach 3.

P.S: sorry for Handwriting :)

If you like my work plz make sure to Subscribe YT: Shubham Harikesh there I will bring video explanation of it.