Department of Software Engineering
Mehran University of Engineering and Technology, Jamshoro

SWE411 – SOFTWARE DESIGN AND ARCHITECTURE			
Instructor	Mr. Mansoor Samo	**Practical/Lab No.**	06
Date		**CLOs**	3
Signature		**Assessment Score**	0.4 Marks

Topic	
Objectives	- To implement Singleton Pattern

Lab Discussion: Theoretical concepts and Procedural steps

Tool: - STAR UML

Theory:

SINGLETON DESIGN PATTERN

Intent and Motivation

- **Intent:** Ensure a class only has one instance, and provide a global point of access to it.
- **Motivation**: In many systems, there should often only be one object instance for a given class
- Print spooler
- File system
- Window manager
- one Input/Output socket
- How do we ensure that a class has only one instance and that the instance is easily accessible?
- Global variable makes an object accessible, but multiple instances
- Class itself responsible for sole instance.

CREATING A SINGLE INSTANCE

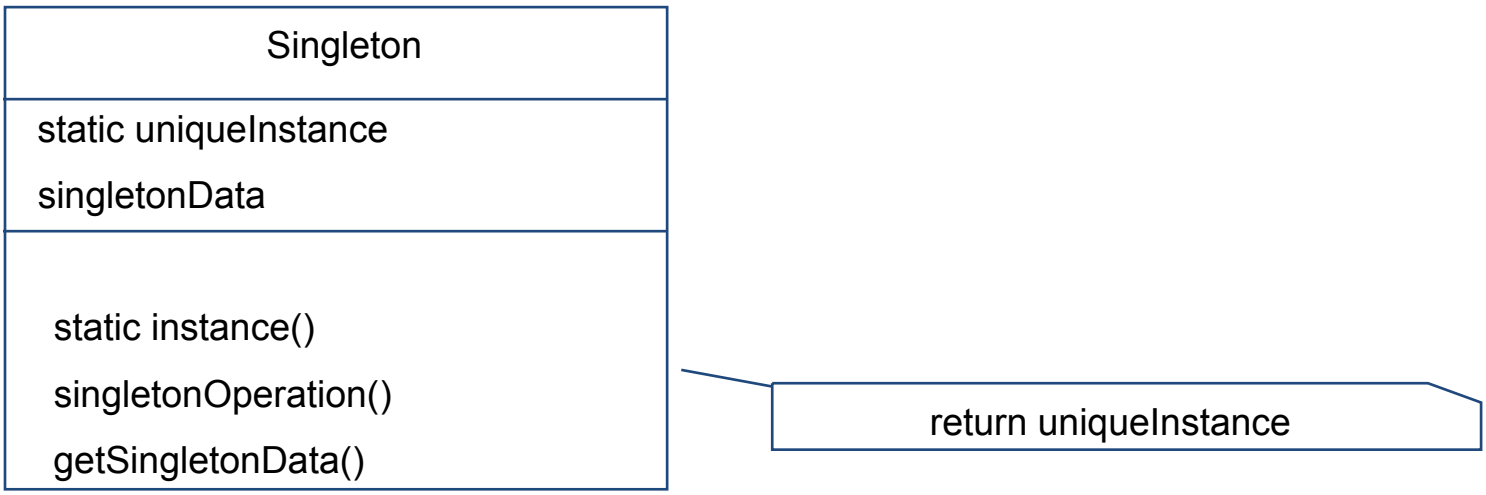
- This maybe necessary because:
  - More than one instance will result in incorrect program behavior
  - More than one instance will result in the overuse of resources
  - There is a need for a global point of access

SINGLETON Pattern: Applicability

Use the Singleton pattern when

- there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point

SINGLETON Pattern: Structure



SINGLETON Pattern: Participants & Collaboration

- Participants:
  - Singleton
    - defines an instance() operation that lets clients access its unique instance.
    - Instance is a class operation (i.e.: static)
    - may be responsible for creating its own unique instance
- Collaboration:
  - Clients access a Singleton instance solely through Singleton's Instance operation.

SINGLETON Pattern: Consequences

- Controlled access to sole instance
  - As the constructor is private, the class controls when an instance is created
- Reduced name space
  - Eliminates the need for global variables that store single instances
- Permits a variable number of instances
  - The class is easily modified to allow *n* instances when *n* is not 1

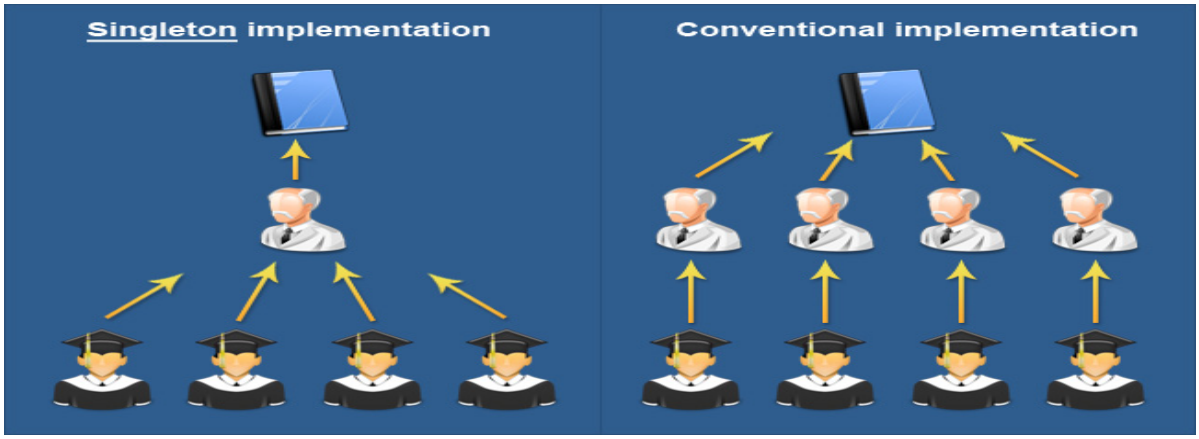
SINGLETON PATTERN: Implementation

```
class Singleton
{
        private static Singleton instance = null;
        private Singleton() {  }
                public static Singleton getInstance() {
                if(instance == null)
                {
                        instance = new Singleton();
```

```
            }
            return instance;
        }
        public static void main(String args[]){

            Singleton s=getInstance();
            System.out.println("one="+s);

            Singleton s2=getInstance();
            System.out.println("one="+s2);
        }
}
```



- **Known use**: A known use for singleton pattern is where a global resource is shared, Also used with meta classes.
- **Related Patterns**: Many patterns can be implemented using the Singleton pattern. Abstract Factory, Builder, Prototype.

EXAMPLES

- Example 1 - Logger Classes

The Singleton pattern is used in the design of logger classes. This classes are usually implemented as a singletons, and provides a global logging access point in all the application components without being necessary to create an object each time a logging operations is performed.

- Example 2 - Configuration Classes

The Singleton pattern is used to design the classes which provides the configuration settings for an application. By implementing configuration classes as Singleton not only that we provide a global access point, but we also keep the instance we use as a cache object. When the class is instantiated(or when a value is read) the singleton will keep the values in its internal structure. If the values are read from the database or from files this avoids the reloading the values each time the configuration parameters are used.

TASK

- **Implement the singleton pattern in a class which returns database connections to the clients.**