SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT, NAGPUR - 440013

DESIGN PATTERNS (CST324)

V SEMESTER SECTION A, B

COURSE COORDINATOR: RINA DAMDOO

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

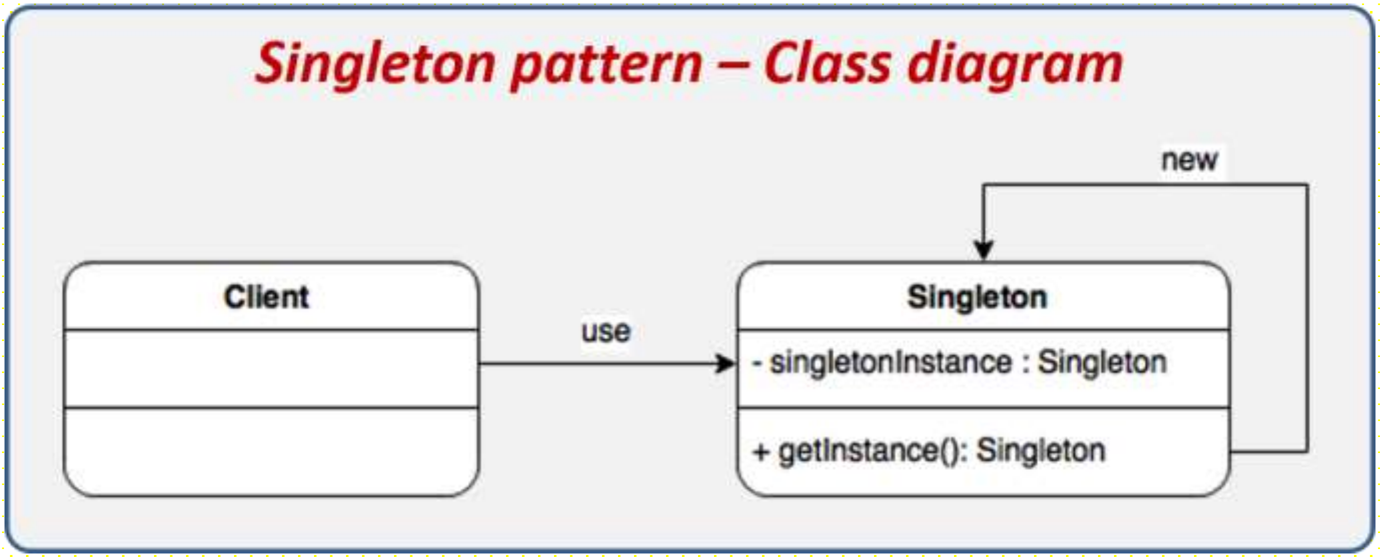
SINGLETON DESIGN PATTERNS

Intent

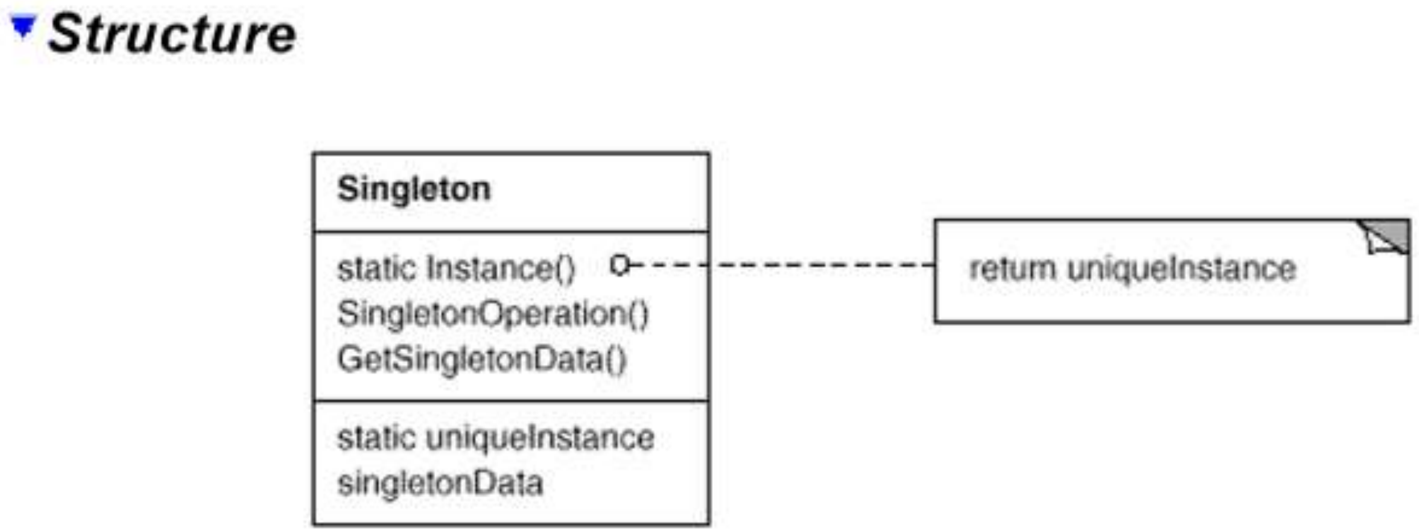
- Ensure a class only has one instance, and provide a global point of access to it.

SINGLETON DESIGN PATTERNS

Problem Statement : develop a Java program to implement Singleton Design Pattern.


Singleton pattern – Class diagram
new
Client
use
Singleton
- singletonInstance : Singleton
+ getInstance(): Singleton

Singleton Design Patterns



MOTIVATION

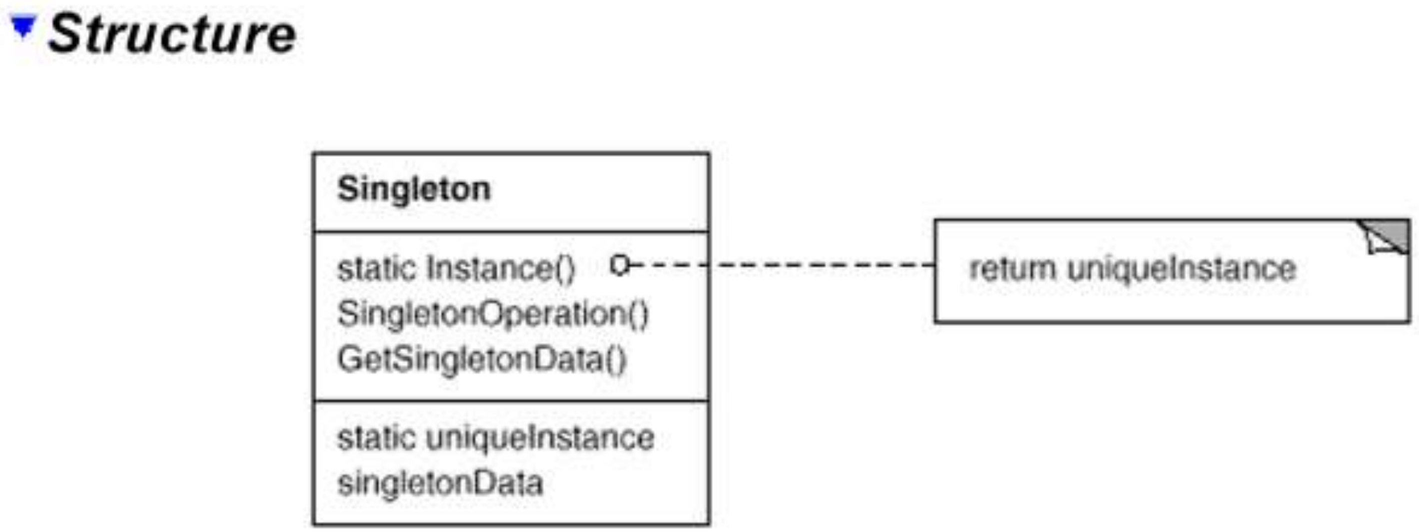
- There can be many printers in a system, there should be only one printer spooler.
- An accounting system will be dedicated to serving one company.
- Only one configuration file a complete project, so that as and when Master configures it, others will get updated configuration.

APPLICABILITY

Use the Singleton pattern when:

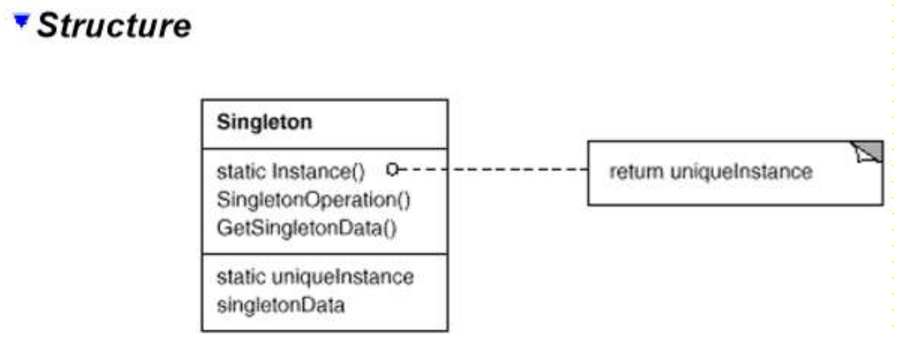
- There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
- When the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

Singleton Design Patterns



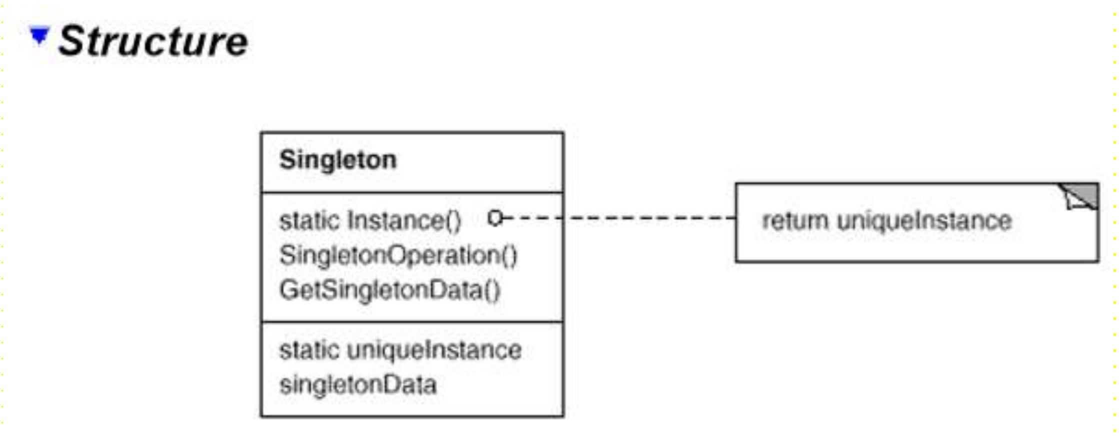
PARTICIPANTS

Participant	Responsibility
Singleton	• defines an Instance operation that lets clients access its unique instance. Instance() is a class operation (a static member function in C++/Java) may be responsible for creating its own unique instance.



COLLABORATIONS

- Clients access a Singleton instance solely through Singleton's Instance operation.



CONSEQUENCES

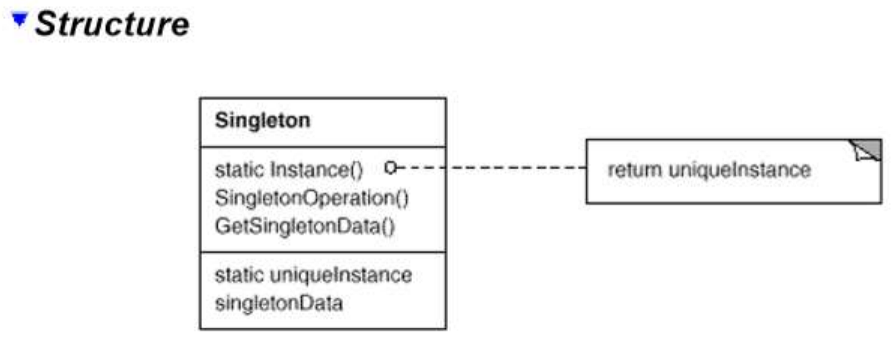
The Singleton pattern has several benefits:

1. **Controlled access to sole instance.** Because the Singleton class encapsulates its sole instance, it can have strict control over how and when clients access it.
2. **Reduced name space.** The Singleton pattern is an improvement over global variables. It avoids polluting the name space with global variables that store sole instances.
3. **Permits refinement of operations and representation.** The Singleton class may be subclassed, and it's easy to configure an application with an instance of this extended class. You can configure the application with an instance of the class you need at run-time.

CONSEQUENCES

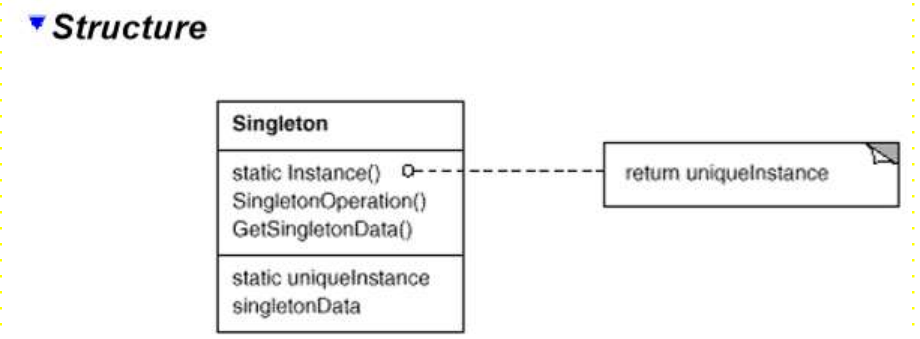
4. **Permits a variable number of instances.** The pattern makes it easy to change your mind and allow more than one instance of the Singleton class.

5. **More flexible than class operations.** Another way to package a singleton's functionality is to use class operations (that is, static member functions in C++/Java).



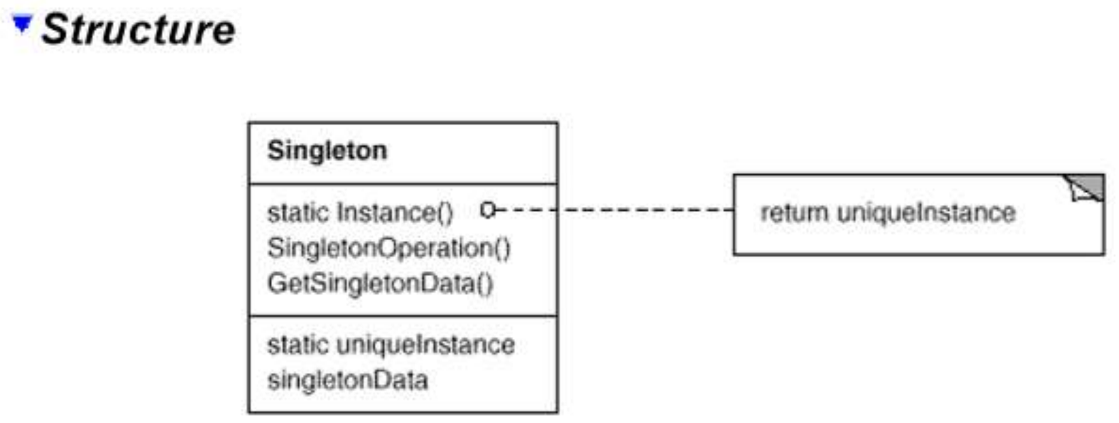
IMPLEMENTATION

1. **Ensuring a unique instance.** The Singleton pattern makes the sole instance a normal instance of a class, but that class is written so that only one instance can ever be created. A common way to do this is to hide the operation that creates the instance behind a class operation (that is, either a static member function or a class method) that guarantees only one instance is created. This operation has access to the variable that holds the unique instance, and it ensures the variable is initialized with the unique instance before returning its value. This approach ensures that a singleton is created and initialized before its first use.



IMPLEMENTATION

2. **Subclassing the Singleton class.** Here the main issue is defining the subclass and installing its unique instance so that clients will be able to use it. The variable that refers to the singleton instance must get initialized with an instance of the subclass. The simplest technique is to determine which singleton you want to use in the Singleton's Instance operation.



KNOWN USES & RELATED PATTERNS

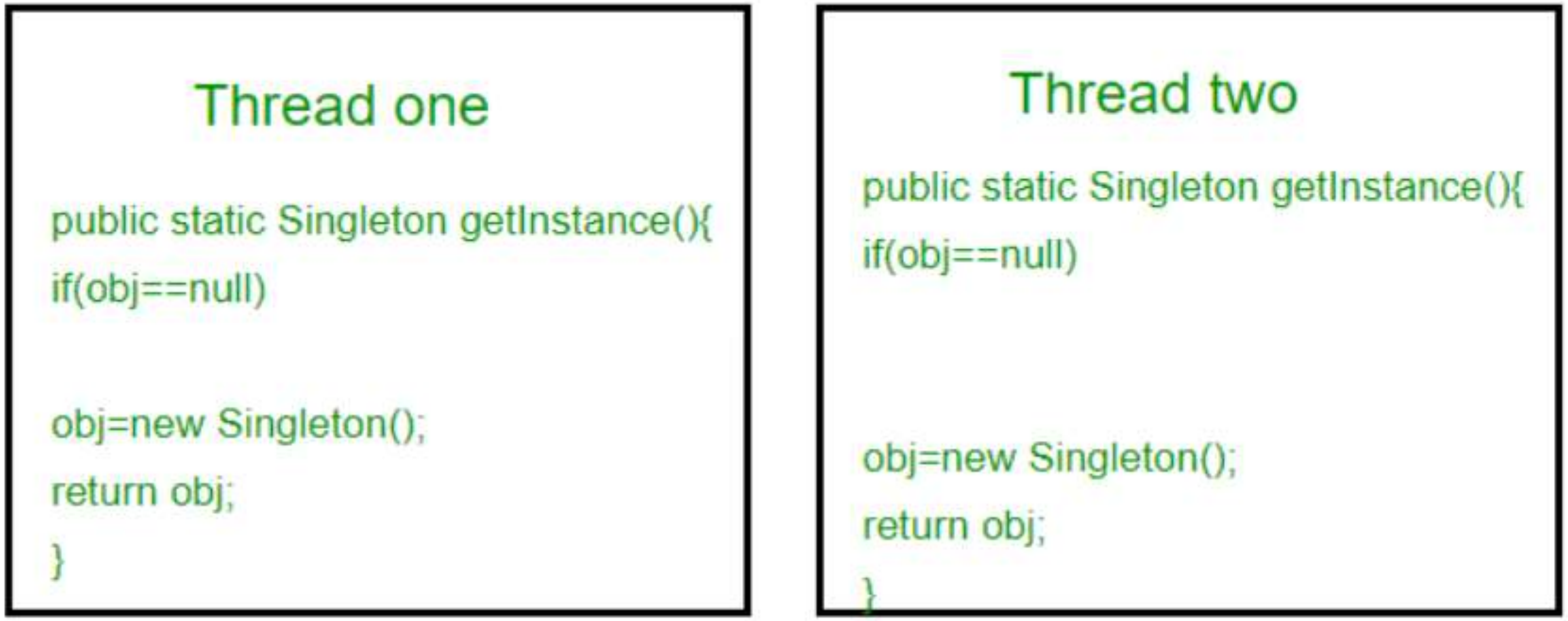
- Logging
- Load balancing
- Configuration
- Communication (to avoid poor performance) and DB connection-pooling.
- Singleton in the Java API is the Runtime class.

- **Abstract Factory**
- **Builder**
- **Prototype**

Few important points

Lazy instantiation:

- We have declared getInstance() static so that we can call it without instantiating the class. The first time getInstance() is called it creates a new singleton object and after that it just returns the same object. Note that Singleton obj is not created until we need it and call getInstance() method. This is called lazy instantiation.
- The main problem with above method is that it is not thread safe. Consider the following execution sequence.



Thread one

```
public static Singleton getInstance(){
if(obj==null)


obj=new Singleton();
return obj;
}
```

Thread two

```
public static Singleton getInstance(){
if(obj==null)



obj=new Singleton();
return obj;
}
```

This execution sequence creates two objects for singleton. Therefore this classic implementation is not thread safe.

Solution is make getInstance() synchronized

Eager Instantiation

```
class Singleton
{
  private static Singleton obj = new Singleton();

  private Singleton() {}

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

- 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.

- Important Note: Use this method only when your singleton class is light and is used throughout the execution of your program.