Design Pattern Singleton

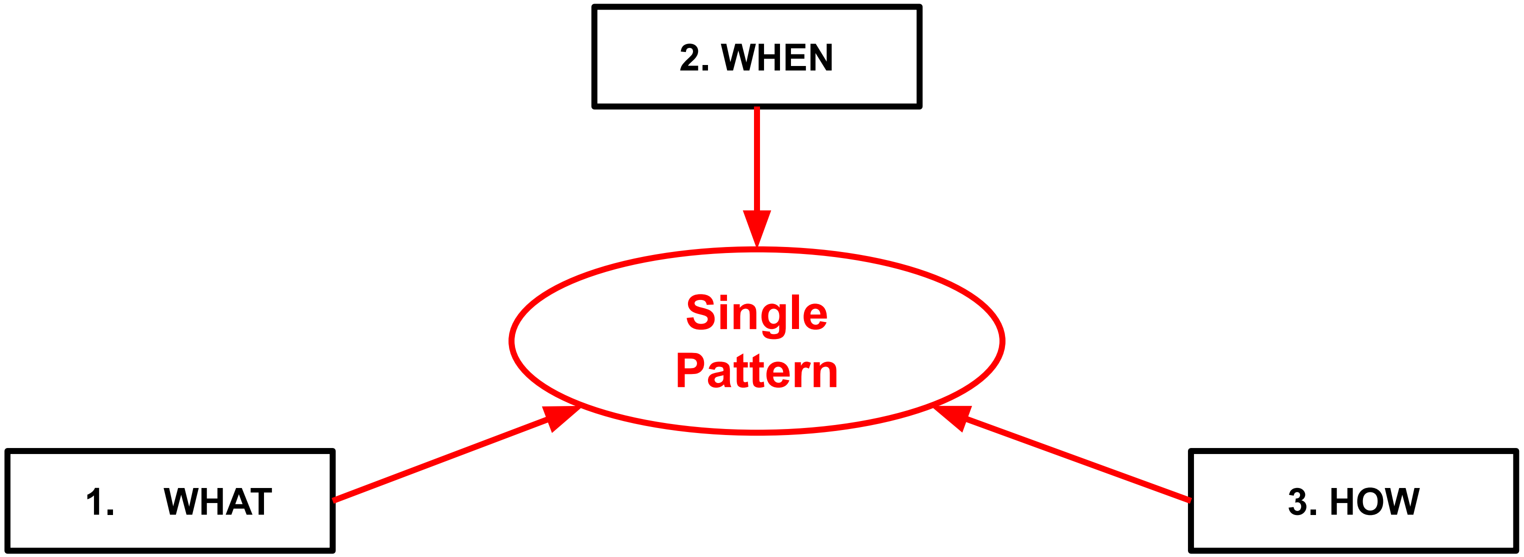
Outline

1. Intent
2. Problem
3. Solution
4. Structure
5. Applicability
6. Implement
7. Pros and Cons

Gang of Four patterns

Creational	Structure		Behavioral	
Abstract Factory	Adapter	Flyweight	Chain of responsibility	Observer
Builder	Bridge	Proxy	Command	State
Factory Method	Composite		Interpreter	Strategy
Prototype	Decorator		Iterator	Template method
Singleton	Facade		Mediator	Visitor
			Memento	

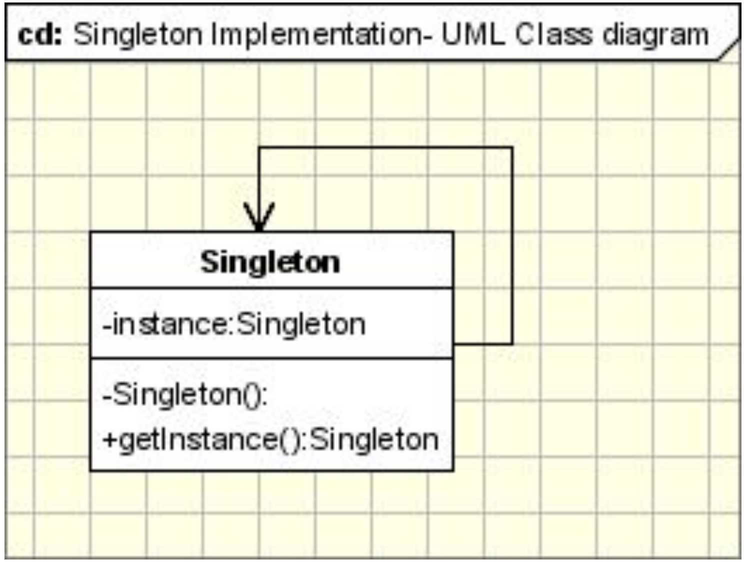
Outline



1. What is Singleton Pattern

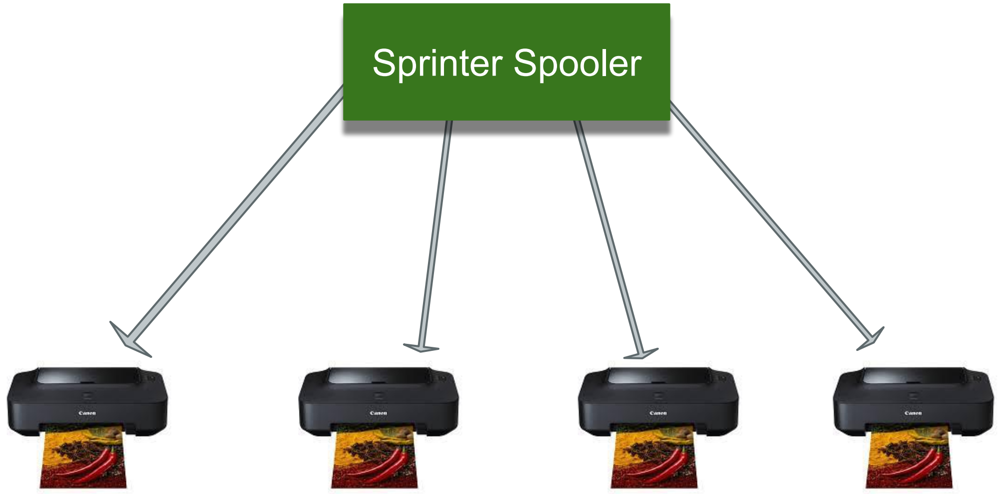
Singleton Patern is a Design Pattern which

- Ensure a class only **has one instance**
- Provide **global access** to that instance.



2. When to use Singleton

- When class has exactly only one instance



3. How to implement Singleton Pattern

1. How to Ensure a class only has one instance?
    - **Private constructor** to restrict instantiation of the class from other classes.
    - **Private static variable of the same class** that is the only instance of the class.
2. How to Provide global access to that instance?
    - **Public static method that returns the instance** of the class, this is the global access point for outer world to get the instance of the singleton class.

3.1 Eager initialization

```
public class EagerInitializedSingleton {

  private static final EagerInitializedSingleton instance = new EagerInitializedSingleton();

  //private constructor to avoid client applications to use constructor
  private EagerInitializedSingleton(){}

  public static EagerInitializedSingleton getInstance(){
     return instance;
  }
}
```

3.2 Lazy initialization

```
public class LazyInitializedSingleton {

    private static LazyInitializedSingleton instance;

    private LazyInitializedSingleton(){}

    public static LazyInitializedSingleton getInstance(){
        if(instance == null){
            instance = new LazyInitializedSingleton();
        }
        return instance;
        }
}
```

3.3 Thread Safe initialization

```
public class ThreadSafeSingleton {

    private static ThreadSafeSingleton instance;

    private ThreadSafeSingleton(){}

    public static synchronized ThreadSafeSingleton getInstance(){
        if(instance == null){
            instance = new ThreadSafeSingleton();
        }
        return instance;
    }
}
```

3.4 Thread Safe Upgrade initialization

```
public class ThreadSafeSingleton {
   private static ThreadSafeSingleton instance;
   private ThreadSafeSingleton(){}

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

Q& A