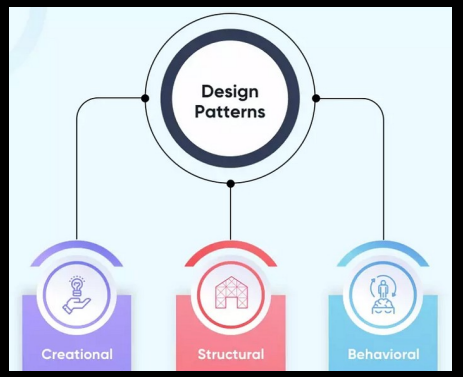


Creational Design Pattern << Singleton >>

Ajith de Silva @epita.fr | 2025

Creational Design Patterns

1. **Singleton**
   - *… One INSTANCE.*
   - *… Global access*
2. **Factory**
   - *… Interface to create instances in supper class.*
   - *… Subclass decides type of Instance`*
3. **Abstract Factory**
   - *… Group similar Factory based on Instances types*
4. **Builder**
   - *… Create complex instance step by step*
5. **Prototype**
   - *… Clone Existing instance*

Singleton Design Pattern

1. **In Brief….**

   One instance and a global point of access to it.

   Useful when to restrict the instance of a class to ONLY 1 instance

2. **HOW ??**

   Private Constructor

   … prevent other classes from instantiating it directly

   Static Instance

   … class contains static member variable to hold the single instance

   … It gets created when class is loaded into memory 1st TIME

   Static Method for instance access

   … provides a static method to access instance for other classes
   something like “getInstance()”

Singleton Design Pattern :: MORE

1. **Lazy Initialization (if needed)**

   … Created only when called “getInstance()” first time.

   … Improved performance by avoiding unwanted instance creation.

   (IF WE USE IT on very rare condition)

***** IF NO NEED --- DO NO USE IT**

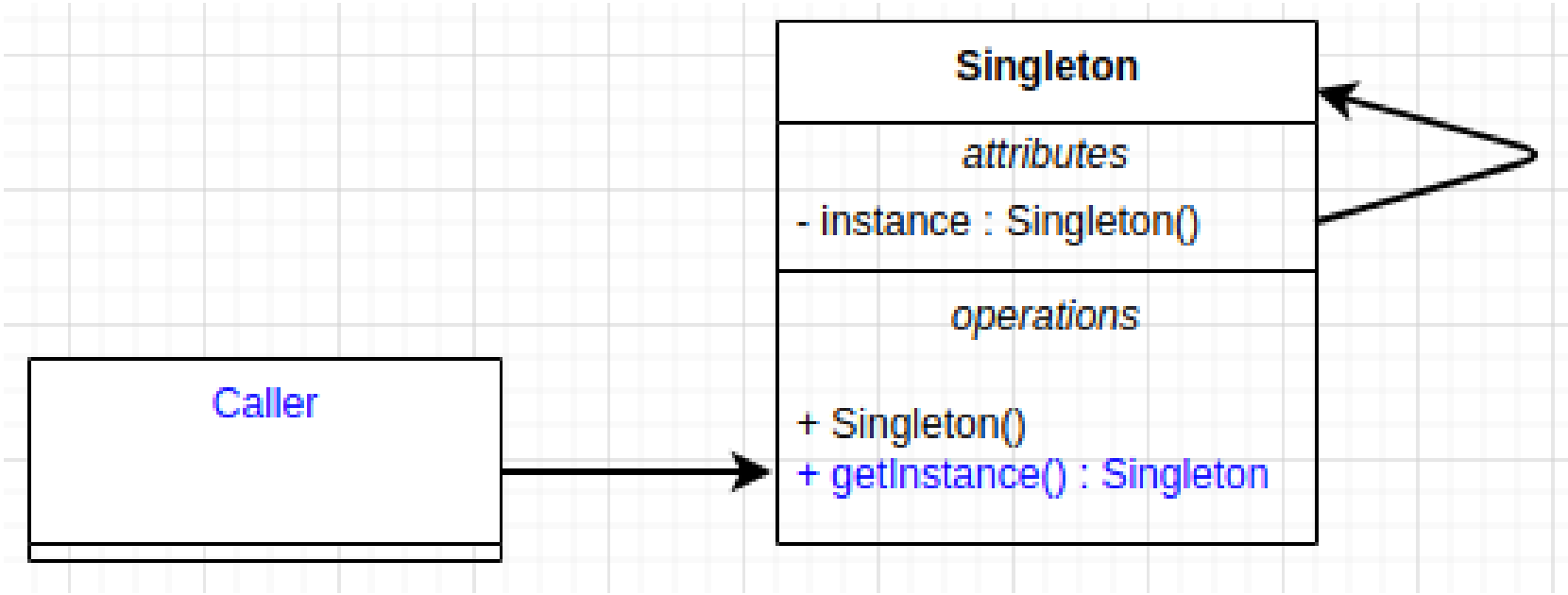
2. **Tread Safety??**

   … NO

   … Need extra treatments on multi-threaded environments

   ****** known as Synchronization/Locking***

Singleton Design Pattern :: UML



Singleton Design Pattern :: How to Implement

1. **Add private static field to store instance**

2. **Define public static method to retrieve instance**

   … implement “Lazy Initialization” and store the created instance in the private static field

3. **Make the constructor s private**

   … static method will ab able to call it but NOT by external

4. **Client/caller will use the Singleton static method “getInstance()” to retrieve it**

Singleton Design Pattern :: Practical

1. Check/study the singleton sample
2. Run/Debug and check how it works
3. Modify Constructor scope to public and check the behavior