Singleton Demo: Problem Statement

In the quiet town of Singletonville, there's only one bank named "SingletonBank". The town's residents, who are all customers of this bank, perform regular banking activities such as depositing or withdrawing money. The bank needs a system that allows these customers to interact with it. Given the unique nature of there being only one bank in the town, the system should ensure that all operations, whether it's a deposit by one customer or a withdrawal by another, are all conducted with this one and only bank.

Singleton Demo: Relevance

Why is the Singleton Pattern Important for This Scenario?

- ***Consistency and Data Integrity***: Having one instance ensures consistent data management. Multiple instances might lead to discrepancies in the funds being tracked for the bank.
- ***Shared Resource Management***: The SingletonBank's funds are a shared resource amongst all customers. The Singleton pattern ensures that this resource is managed and accessed centrally.
- ***Avoiding Redundancy***: Creating multiple instances representing the same physical entity (the one bank in town) is redundant and could lead to inefficiencies in memory usage.
- ***Real-world Analogy***: Just as there is only one physical bank in Singletonville, our system should mirror this reality by having only one instance of the bank.