Creational Pattern: Singleton

Kevin Dockx
Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Describing the singleton pattern

Structure of the singleton pattern

Implementation

- Real-life sample: Logger
- Making the implementation thread-safe

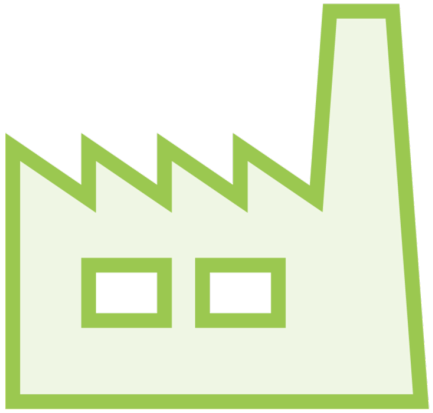
Coming Up

Use cases for this pattern

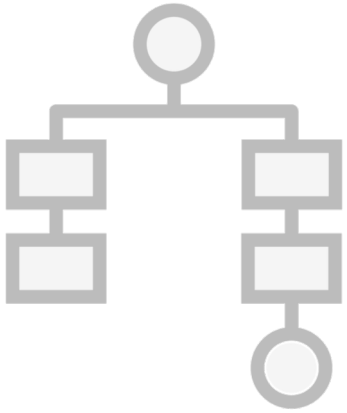
Pattern consequences

Related patterns

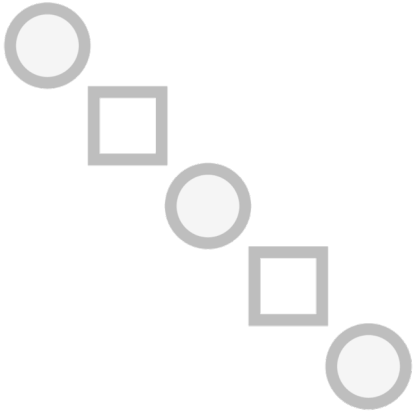
Describing the Singleton Pattern



Creational



Structural



Behavioral

Singleton

The intent of the singleton pattern is to ensure that a class only has one instance, and to provide a global point of access to it

Describing the Singleton Pattern

Real-life example:

- Logger
- One instance is preferred to avoid unintended consequences

Describing the Singleton Pattern

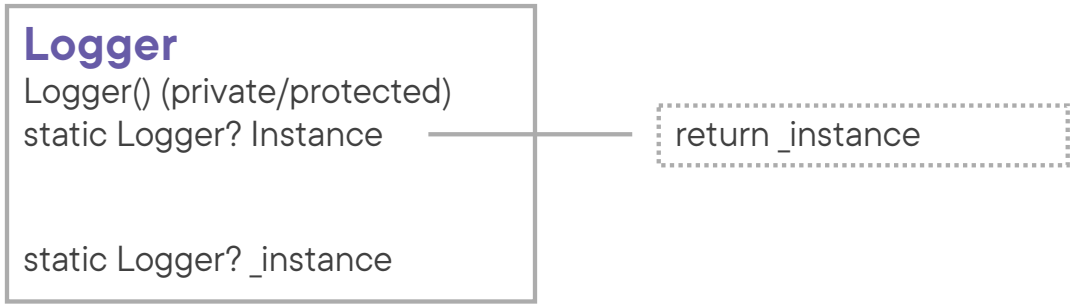
Holding the class instance in a global variable doesn't prevent clients from creating other instances of the class

Make the class responsible for ensuring only one instance of itself exists

Describing the Singleton Pattern

Describing the Singleton Pattern

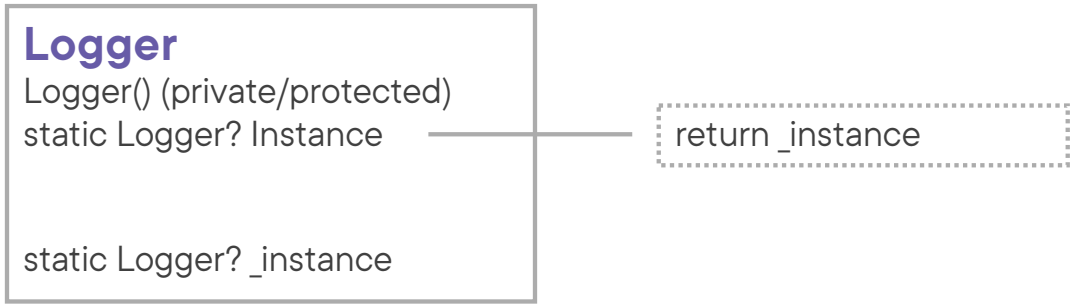
Describing the Singleton Pattern



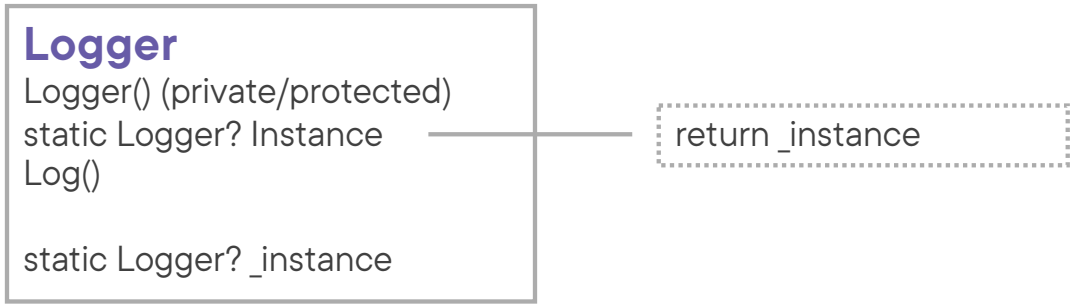
Prefer lazy instantiation

Create and store the instance when it's requested for the first time, and return that instance on subsequent requests

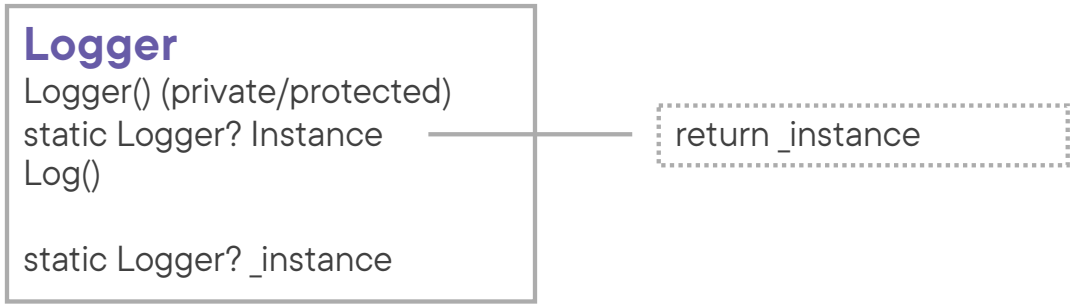
Describing the Singleton Pattern



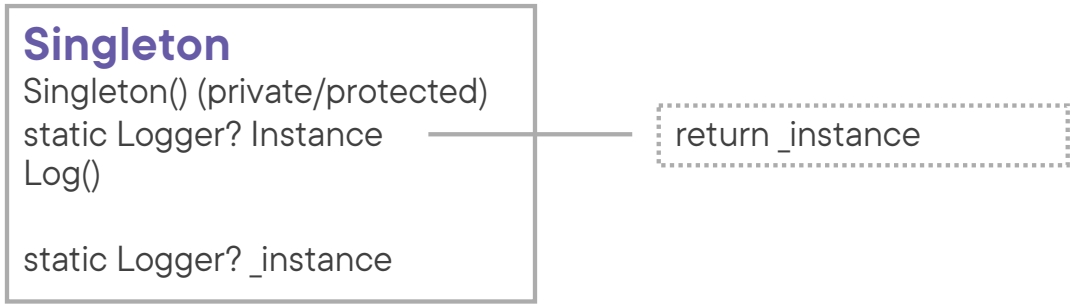
Describing the Singleton Pattern



Singleton Pattern Structure

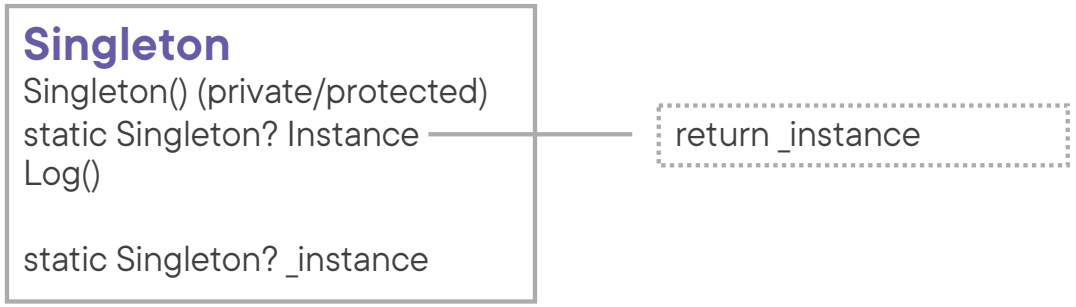


Singleton Pattern Structure

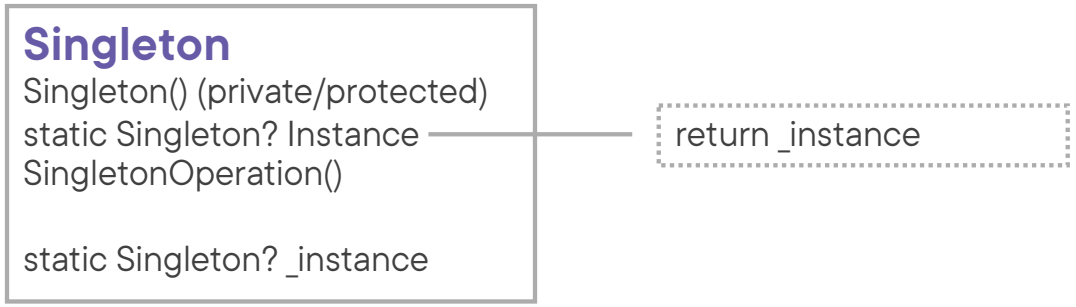


A **Singleton** defines an instance operation that lets clients access its unique instance

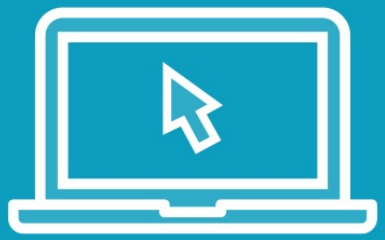
Singleton Pattern Structure



Singleton Pattern Structure

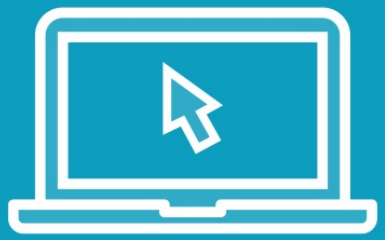


Demo



Implementing the singleton pattern

Demo



Making the implementation thread-safe with Lazy<T>

Use Cases for 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

Pattern Consequences

Strict control over how and when clients access it

Avoids polluting the namespace with global variables

Subclassing allows configuring the application with an instance of the class you need at runtime

Multiple instances can be allowed without having to alter the client

Violates the single responsibility principle

Related Patterns

Abstract Factory
Can be implemented as a singleton

Builder
Can be implemented as a singleton

Prototype
Can be implemented as a singleton

State
State objects are often implemented as singletons

Summary

Intent of the singleton pattern:

- Ensure that a class only has one instance
- Provide a global point of access to it

Summary

Implementation:

- Static `Instance` property (+ backing field)
- Private or protected constructor
- `Lazy<T>`

Up Next:
Creational Pattern: Factory Method