Singleton Pattern

CSCI-4448 - Boese

University of Colorado Boulder

Objectives

- Problem
- Definition
- Why
- Examples
- How
- Comparisons

Problem

President

Problem

- We need to be able to turn on and off logging. All classes in our system need to be able to access the logger.
  - *There is no good association for a Logger class if every class needs to access it.*
  - *Would want concurrency controls to ensure only one instance accessing the logging file for writing.*
    - *There should only be one instance*

Definition

Definition

“Restrict the instantiation of a class to one object.”

Definition

Name “Singleton”

- Class has only one instance

Intent

- Ensure a class has only one instance, and provide a global point of access to it.
- Encapsulated "just-in-time initialization" or "initialization on first use” (lazy initialization)

Singleton Design Pattern

Problem:

- Only ever need one instance of a particular class
  - Examples: keyboard reader, bank data collection, logger
  - Make it illegal to have more than one, for safety's sake

Why we care:

- Creating lots of objects can take a lot of time
- Extra objects take up memory
- It is a pain to deal with different objects floating around if they are essentially the same
- Currency issues with multiple objects accessing shared resource

Why

Why use Singleton Pattern?

- Ensure that a class has at most one instance
- Provide a global access point to that instance
- Take responsibility of managing that instance away from the programmer
  (illegal to construct more instances)
- Provide accessor method that allows users to get the (one and only) instance
- Possibly the most known / popular design pattern!
  (this should tell you something)
  and the most incorrectly used pattern

Examples

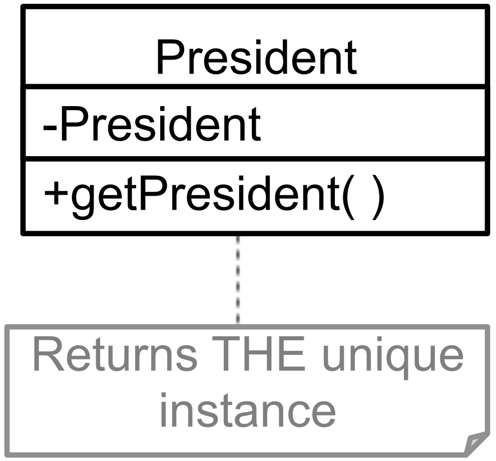
President

Problem

- We need to create a class for the President of the USA, but of course there can only be one president. We want to ensure that only instance can be created.

Example

```
public class President
{
    private static President thePresident;
    private President() {   }
    public static President getPresident()
    {
       if(thePresident == null)
          thePresident = new President();
       return thePresident;
    }
}
```



President
-President
+getPresident( )

Returns THE unique instance

Examples

Logger

Problem

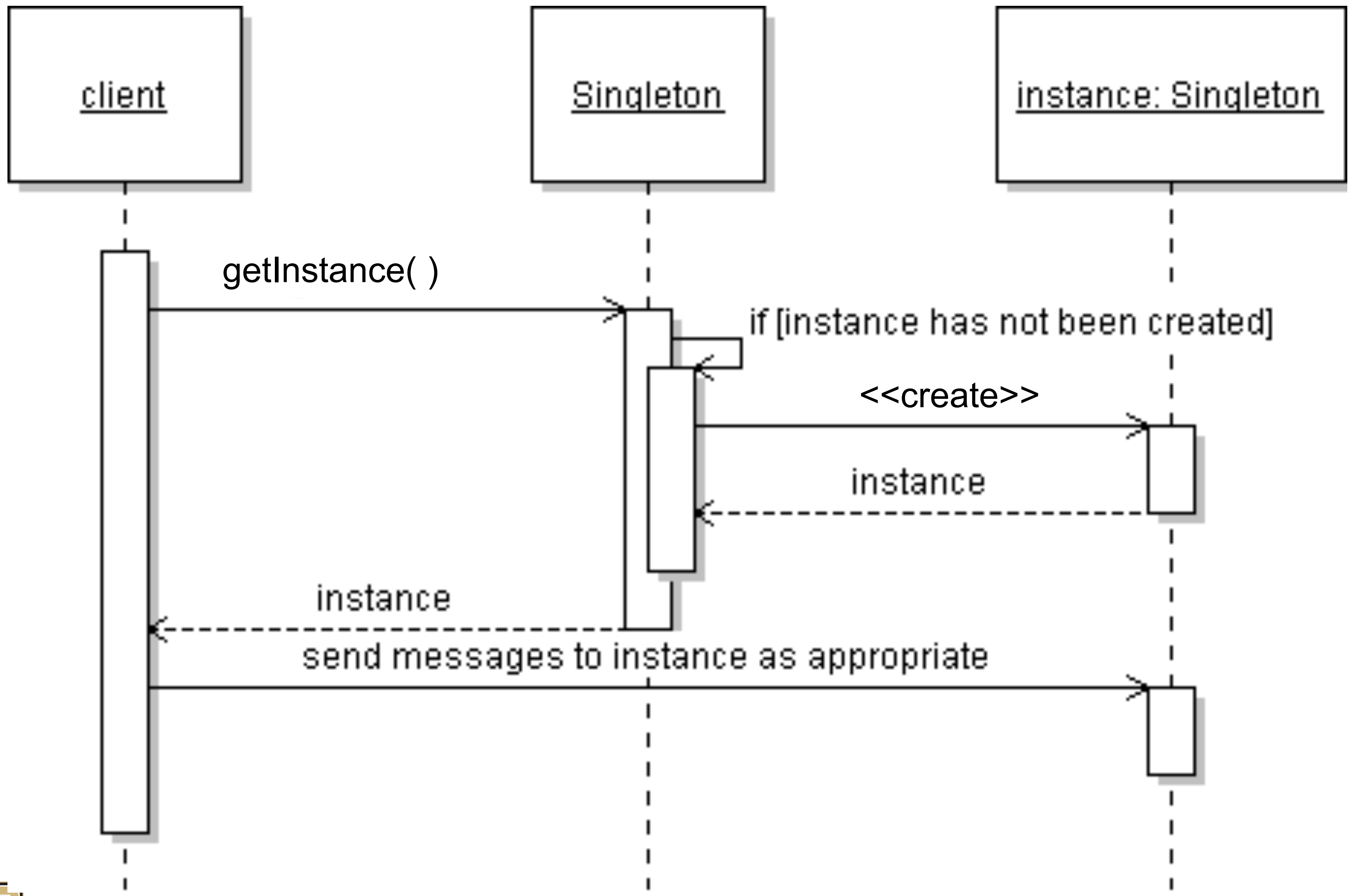
- How would you implement the Logger?
- How would you deal with concurrency?

```
public class Logger
{
    private static Logger logger = null;

    private Logger()
    {
    }
    private static Logger getInstance()
    {
        if (logger == null)
            logger = new Logger();
        return logger;
    }
}
```

```
public class Logger
{
    private static Logger logger = null;

    private Logger()
    {
    }
    private static synchronized Logger getInstance()
    {
        if (logger == null)
            logger = new Logger();
        return logger;
    }
}
```

How

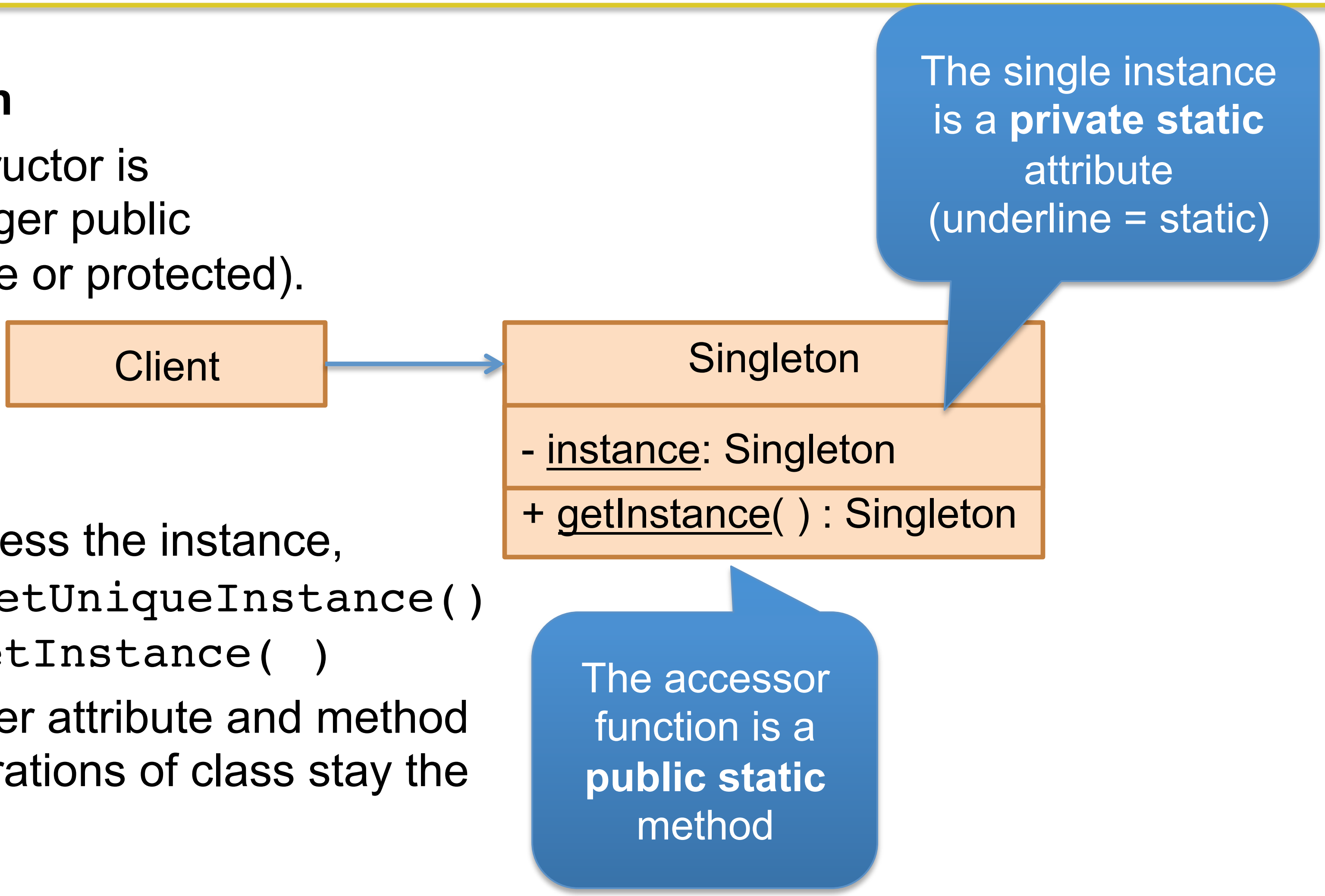
Singleton sequence diagram


client
Singleton
instance: Singleton
getInstance( )
if [instance has not been created]
<<create>>
instance
instance
send messages to instance as appropriate

Structure

Singleton

- Constructor is no longer public (private or protected).



The single instance is a **private static** attribute (underline = static)
Client
Singleton
- instance: Singleton
+ getInstance( ) : Singleton
The accessor function is a **public static** method

- To access the instance, use `getUniqueInstance()` or `getInstance( )`
- All other attribute and method declarations of class stay the same.

How

1. Define a private static attribute in the "single instance" class.
2. Define a public static accessor function in the class.
3. Do "lazy initialization" (creation on first use) in the accessor function.
4. Define all constructors to be protected or private.
5. Clients may only use the accessor function to manipulate the Singleton.

Comparisons

Comparisons

- Abstract Factory, Builder, and Prototype can use Singleton in their implementation.
- Facade objects are often Singletons because only one Facade object is required.
- State objects are often Singletons.
- The advantage of Singleton over global variables is that you are absolutely sure of the number of instances when you use Singleton, and, you can change your mind and manage any number of instances.

Summary

Singleton Summary Thoughts

- The Singleton design pattern is one of the most inappropriately used patterns.
  - Singletons are intended to be used when a class must have exactly one instance, no more, no less.
  - Designers frequently use Singletons in a misguided attempt to replace global variables.
  - A Singleton is, for intents and purposes, a global variable. The Singleton does not do away with the global; it merely renames it.

Singleton Summary Thoughts

When is Singleton **unnecessary**?

- Short answer: most of the time.
- Long answer: when it's simpler to pass an object resource as a reference to the objects that need it, rather than letting objects access the resource globally.
- The real problem with Singletons is that they give you such a good excuse not to think carefully about the appropriate visibility of an object. Finding the right balance of exposure and protection for an object is critical for maintaining flexibility.
- “Even though loggers are global state, since no information flows from loggers into your application, loggers are acceptable as Singletons.”