

A graphic of a spiral-bound notebook with a grey cover and a white page. The spiral binding is on the left side. The page contains the text "Patterns" and "Lecture 2".

Patterns

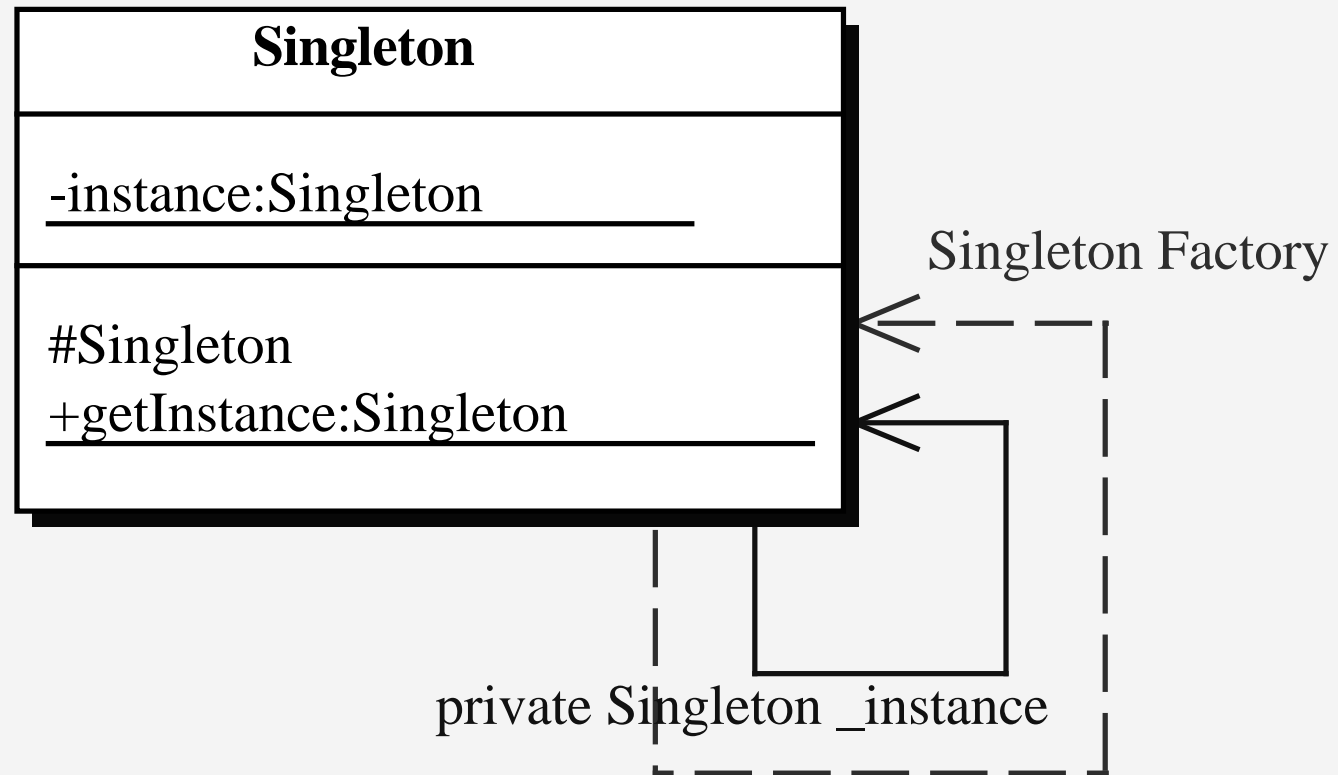
Lecture 2

A graphic of a spiral-bound notebook with a grey cover and a white page. The spiral binding is on the left side. The page contains the text for the Singleton design pattern.

Singleton

Ensure a class only has one instance, and provide a global point of access to it.

UML



Motivation

- Sometimes we need to ensure that there can only be one instance of an object within a given system
 - Filesystem access
 - Print Spooler
 - Thread manager
 - System Dictionary

Strategy

- The Singleton works its magic by accomplishing two things simultaneously:
 - hiding the class's constructor from public view
 - this disables the ability for others to create new instances of the class
 - maintaining a single reference to a single instance of the class, internally in the class

Benefits

- Controlled access to a single instance
- Enables an application to avoid the use of global references for single instances
- Provides a structure for the ability to have “Dual” or “Triad” objects (an object can control precisely two or three instances, and round-robin the incoming access to these objects)
- Avoids the use of static class operations, which cannot be used polymorphically, because there is not object in play (no self or this pointer)

Code

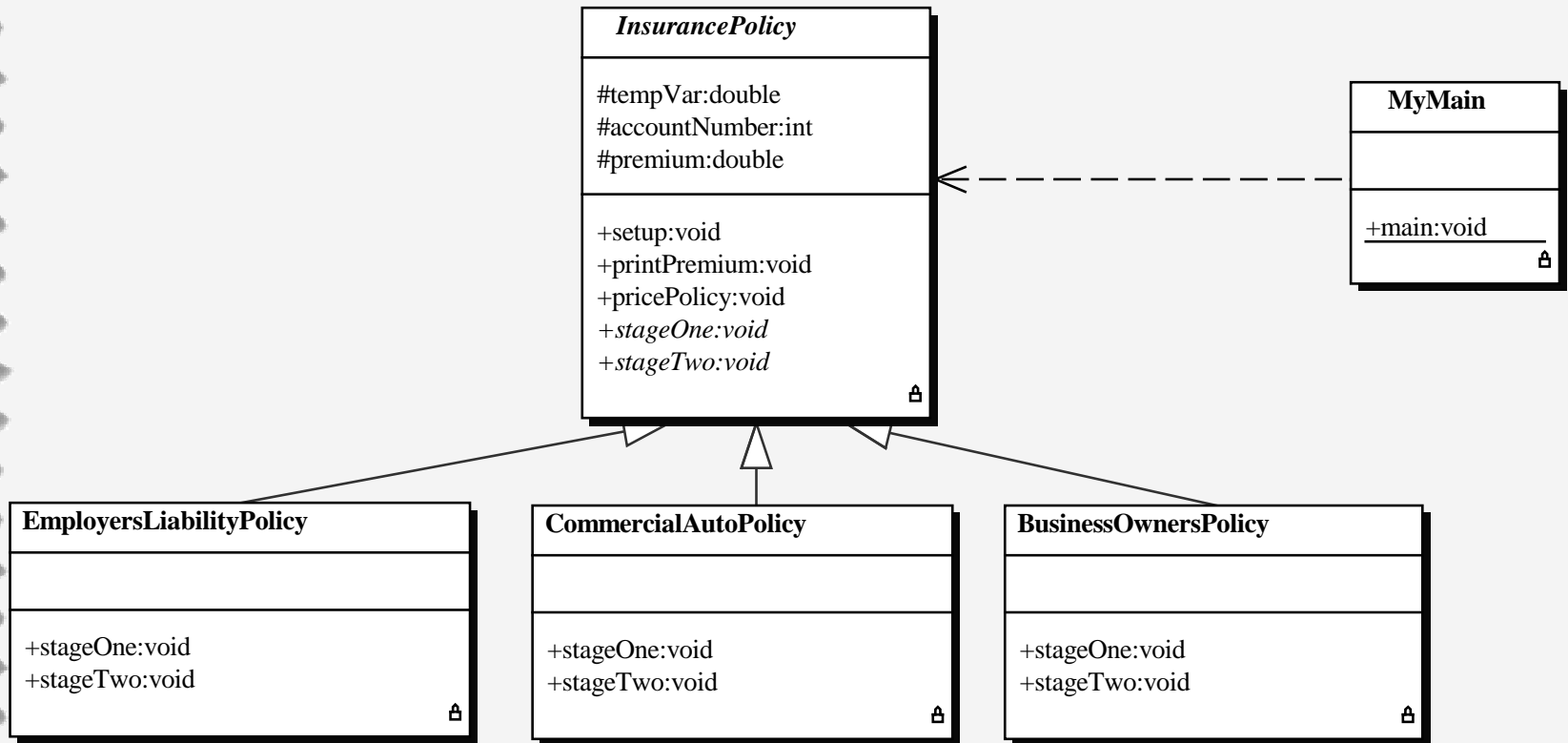
```
public class Singleton {  
    protected Singleton(){}  
    public static Singleton getInstance(){  
        if (instance == null) {  
            instance = new Singleton();  
        }  
        return instance;  
    }  
    private static Singleton instance = null;  
}
```



Template Method

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses, letting subclasses define certain steps of an overall algorithm without changing the algorithm's structure

UML



Motivation

- Sometimes, an algorithm is generic among multiple subtypes, with just a few exceptions
- These exceptions prevent the abstraction of the algorithm into a common base class
- Use the Template Method pattern to allow the invariant parts of the algorithm to exist in the common base class, and depend on derivatives to tailor the changeable parts of the otherwise common algorithm

Benefits

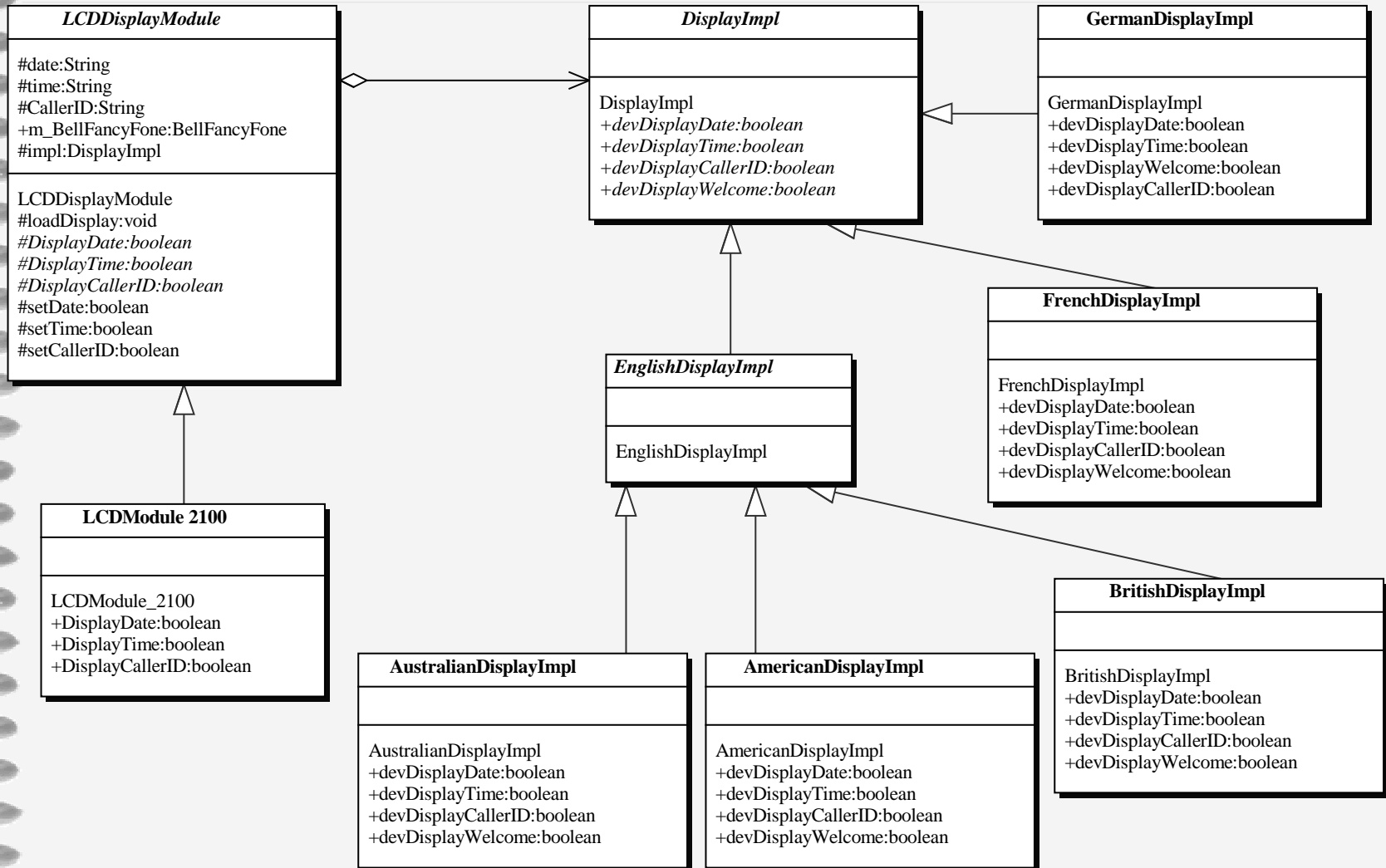
- Because much of the algorithm can be encapsulated in an abstract base class, the invariant parts of the algorithm are localized and non-redundant
- Individual behaviors are easily accomplished and themselves encapsulated in derivative classes
- Predefined “hook” callback methods can be assigned at defined points in a sequence of activities

A graphic of a spiral-bound notebook with a grey cover and a white page. The spiral binding is on the left side. The word "Bridge" is written in the center of the page in a black serif font. Below it, a definition of the Bridge design pattern is written in a smaller black serif font.

Bridge

Decouple an abstraction from its implementation so that the two can vary independently

UML



Motivation

- Inheritance statically binds a particular abstraction (interface) with a particular implementation
- A bridge is used when an abstraction can have one of several possible implementations, but you want to be able to choose dynamically *which* implementation is used

Benefits

- Using a bridge prevents a proliferation of inheritance-based classes (classic example of multiple window platform support)
- Using a bridge allows you to avoid a permanent binding between an abstraction and its implementation
- A bridge allows you to share a single implementation among multiple objects, so that the client is unaware of such sharing