Using Inheritance

Jim Fawcett

CSE687 – Object Oriented Design

Spring 2014

Topics: Using Inheritance

- Protocol Classes
- Hooks
- Mixin Classes
- Sharing common code
- Reusing Implementation

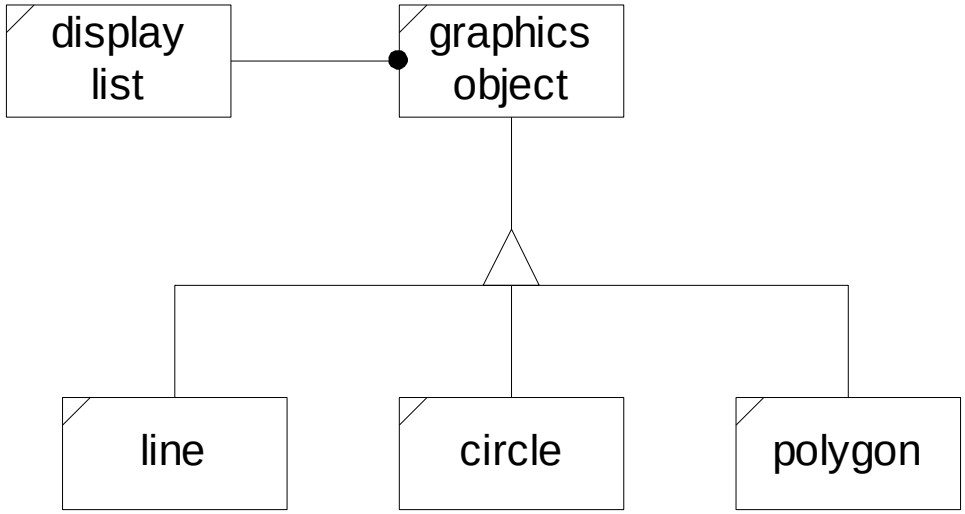
Liskov Substitutability (LS)

- Public inheritance is a contract for substitutability.
  - Any base class reference or pointer may be bound to an instance of any class derived from that base.
  - The compiler will ensure that derived public interfaces support LS syntactically:
    - Derived classes inherit all of the inheritable methods of the base class and so can receive its messages.
  - It is up to you to ensure semantic LS.
    - Your class must be a semantic specialization of the base.

Protocol Classes

- A protocol class (C++ interface) provides a language for clients to use when interacting with any of its derived class instances. It has:
  - No data
  - No declared constructors
  - A virtual destructor
  - A set of pure virtual functions that define the hierarchy's language.
- All classes are derived from the protocol base class using public inheritance.

Protocol Example



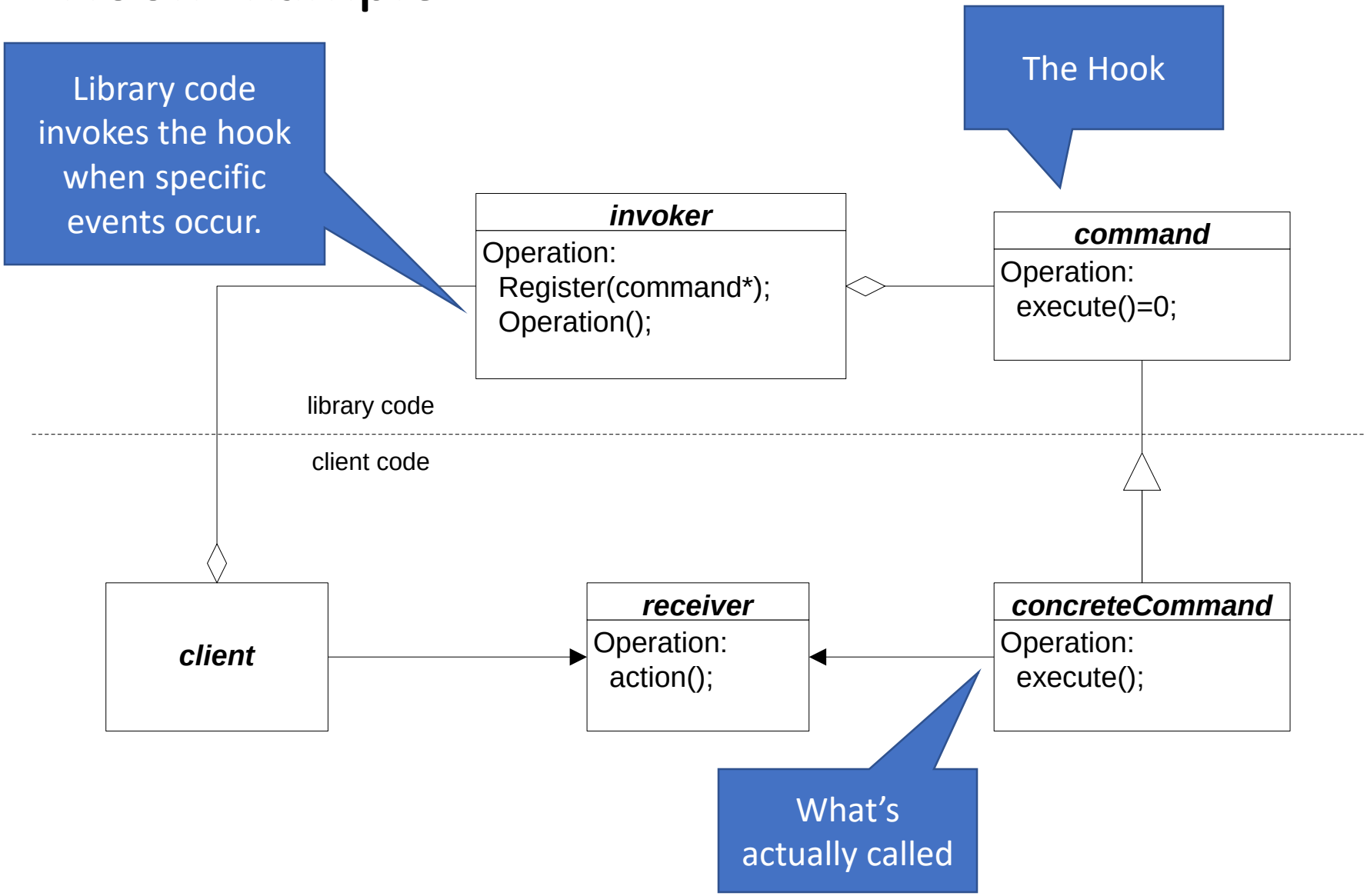
Hooks

- A hook is a base class that supports modification of a library's behavior by applications designed to use the library, without modifying any of the library's code.

- The Hook class provides a protocol for application designers to use and a set of virtual methods that are overridden to provide required application behavior.

- One very common usage provides a way for applications to respond to events that occur within the library.

Hook Example

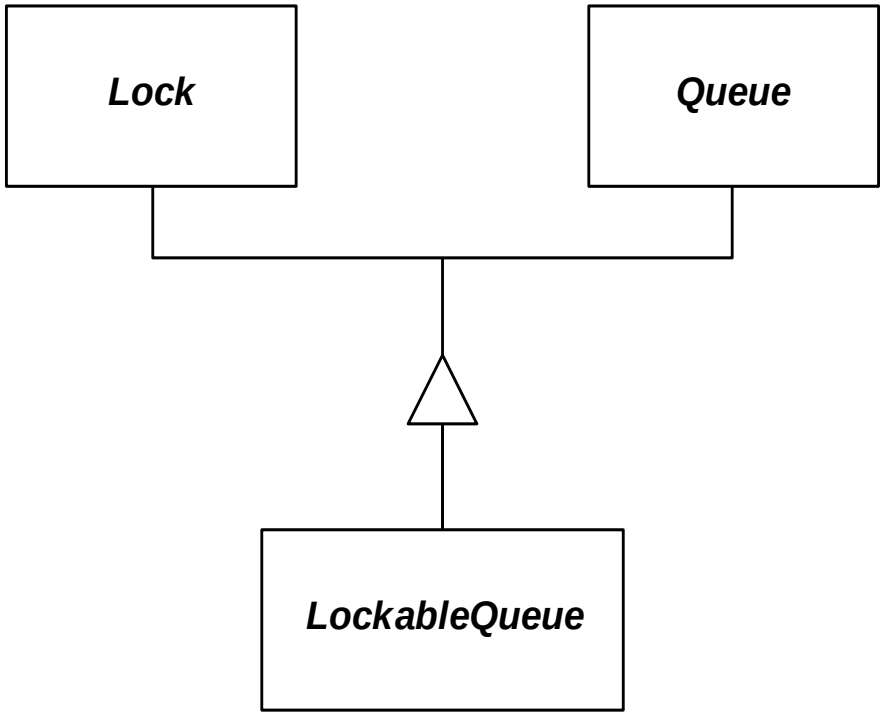


Mixins

- A mixin class provides a specialized set of behaviors intended to be used through multiple inheritance with other classes.

- Mixins can be used to provide reference counting, locking, memory management, and other specialized behaviors as services to any class that needs them.

Mixin Example



Share Common Code

- A base class may provide default implementations of some or all of the base class protocol. These, then, are shared by all derived classes that do not override the defaults.
- When using concrete base classes, however, you need to be careful that you support reasonable semantics:
  - Disallow assignment of derived instances to base instances or derived instances of one type to an instance of another type through base pointers or references.
    - You can make the base class abstract by including a pure virtual function and make the base assignment private.
    - Provide assignment operators for each of the derived classes.

Reusing Implementation

- C++ supports reuse of implementation with public, private, and protected inheritance.

- You should prefer composition unless:
  - You need access to protected data from the base class, or:
  - You want to support polymorphic operations in member functions of the derived class, perhaps to bind to any derived object passed to a member function by base reference or pointer without knowing the concrete class of the object.

End of Presentation