iOS Design Patterns

Jackie Myrose

CSCI 5448

Fall 2012

Design Patterns

- A design pattern is a common solution to a software problem
- They are helpful for speeding up problem solving, ensuring that a developer doesn't have to re-invent the wheel for every situation
- They also give developers a common vocabulary with which to get across high-level ideas with minimal explanation and full understanding

Why iOS?

- Design patterns are everywhere in iOS
- Because iOS is a fairly specific platform, developers often face similar problems over and over, so there are a few design patterns that are extremely common in iOS

In this presentation

- Singleton
- Delegate
- Model View Controller
- Observer
- Façade
- Command
- Template Method

Singleton

- The singleton pattern is very simple but extremely powerful
- It is a very common pattern, but developers have to be careful not to overuse it
- Because abuse of the singleton pattern is common, some developers avoid the pattern altogether

Singleton

- When a class is restricted to just one instantiation, that one object is called a singleton
- In some situations it can be problematic to have two instances of a class running, this should be the only reason to use the singleton pattern
- The next slide contains a basic example of creating a singleton in objective-c, however keep in mind that this code is not thread safe

Singleton - code

```
+(ExClass *) singleton{
	static ExClass *sharedInstance = nil;

	if ( sharedInstance == nil){
		sharedInstance = [[ExClass alloc] init];
	}

	return sharedInstance;
}
```

Delegate

- The delegate pattern is another simple, yet powerful design pattern
- As the name indicates, the delegation pattern is used to have one object delegate control to another object to act on its behalf
- This is used to keep implementation specific behavior out of the generic class

Delegate

Image from: https://developer.apple.com/library/mac/#documentation/cocoa/conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html

Delegate

- Many UI elements in iOS use delegates to control their behavior
- One example of this is UIScrollView
- The UIScrollView class has no knowledge of what it should be scrolling as that is an application specific task
- So to notify the application of scrolling events, UIScrollView uses the UIScrollViewDelegate
- The application can implement the delegate and then intercept the scrolling events sent to it by the UIScrollView
- The next slide has examples of methods the UIScrollViewDelegate could implement

Delegate - code

```
UIScrollViewDelegate
- scrollViewDidScroll:
- scrollViewWillBeginDragging:
- scrollViewWillBeginDecelerating:
- scrollViewDidEndDecelerating:
- scrollViewDidZoom:
```

Model View Controller

- All iOS apps use Model View Controllers (MVCs)
- MVCs are the link between an app's data and its UI
- The MVC is broken up as such:
  - Model – Underlying data
  - View – The view the user sees, the UI
  - Controller – Determines the interactions between the model and views
- This keeps the program modularized, allowing for high cohesion and loose coupling

Model View Controller

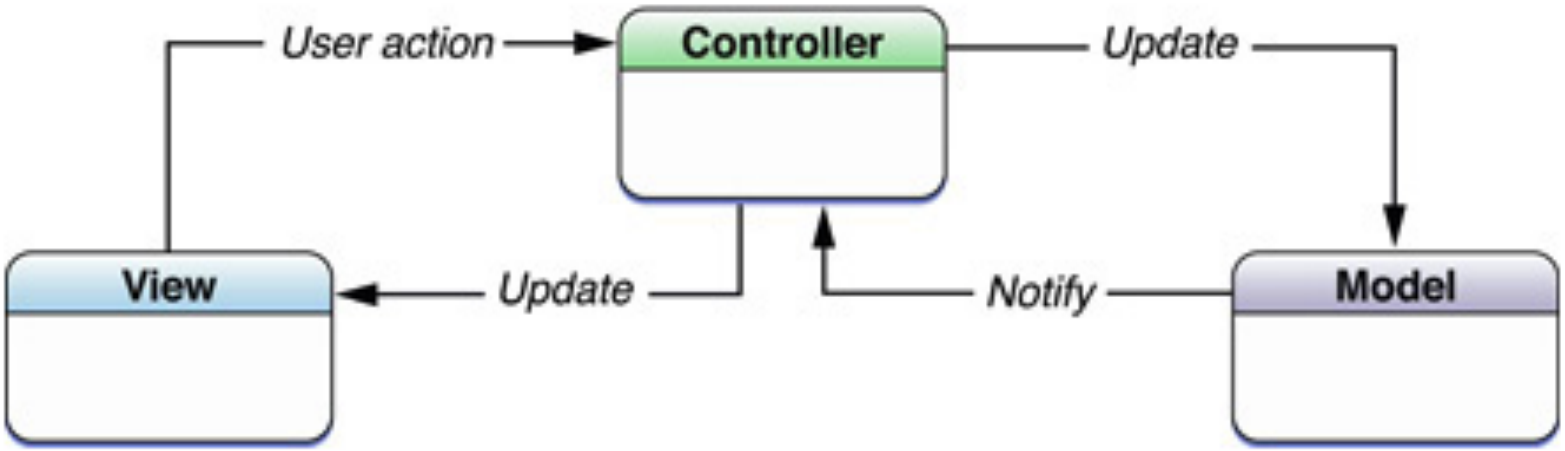

Image from: http://developer.apple.com/library/ios/#documentation/general/conceptual/DevPedia-CocoaCore/MVC.html

Observer

- The observer pattern is used when one object wants to know when another object changes
- This pattern is build into every NSObject vi Key-Value Observing
- This is also often used with MVCs because when a model changes you often will want to update the views

Observer

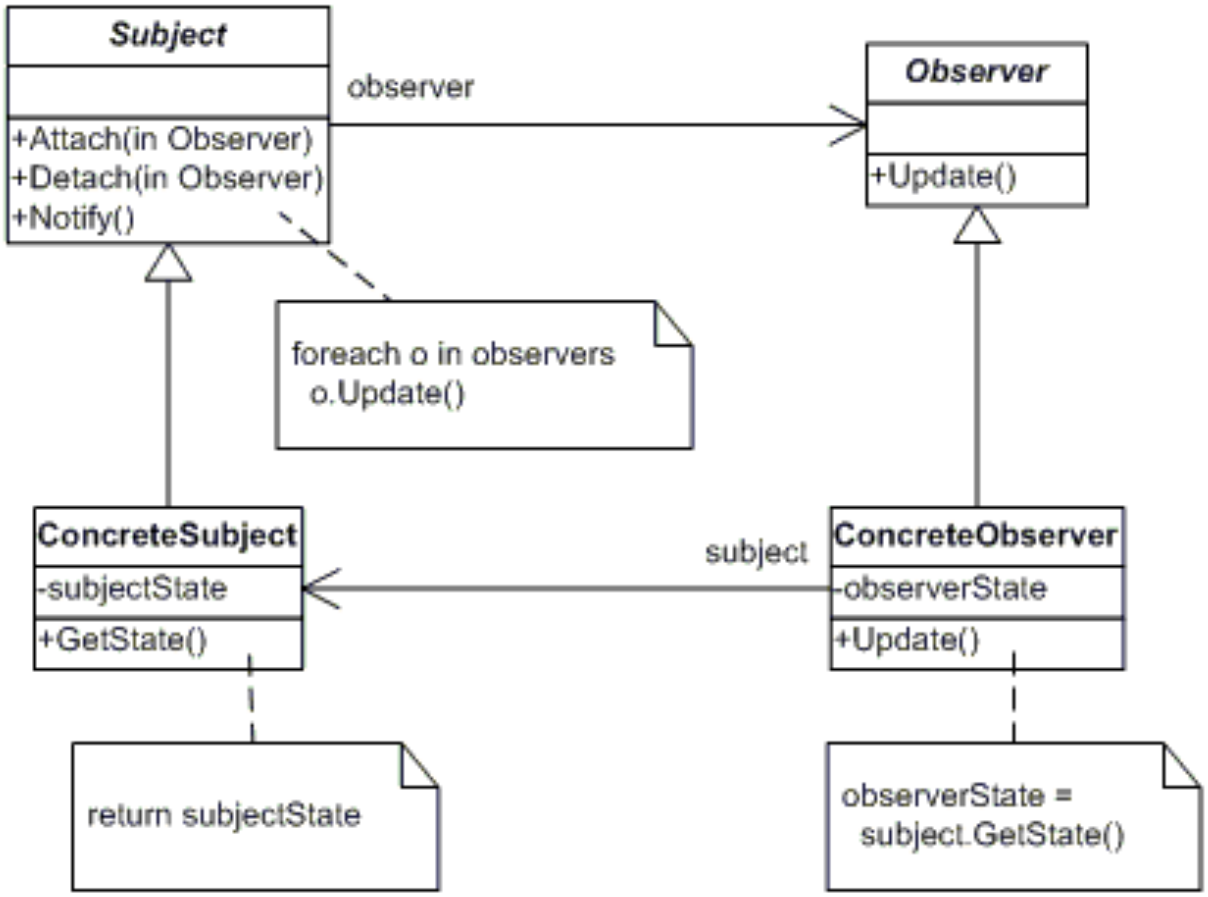

Image from: http://www.dofactory.com/Patterns/PatternObserver.aspx

Observer

- The observer pattern is similar to the delegate pattern, however one key difference is that observable objects support multiple observers, while a delegate is just one object
- However, with this expanded possibility comes one big pitfall: you must remember to remove an object as an observer when that object is deallocated, otherwise there will be a code leak
- The following slide contains a code sample of what the Observable class looks like

Observer - code

```
@interface Observable: NSObject
- (void)addObserver:
  (id<NSObject>)observer;
- (void)removeObserver:
  (id<NSObject>)observer;
- (void)notifyObservers:
  (NSInvocation*)invocation;
@end
```

Façade

- The façade pattern is used for simplifying a complex system
- It allows for a subsystem to be accessed through one entry point, allowing the systems using it to be unaware of the classes in the subsystem
- This is also useful if the classes under the façade are likely to change, as we can still have the façade class have the same API

Façade

- One example of this in iOS is the NSImage class
- This class is a façade which provides an interface for using and loading images that can be vector-based or bitmap-based
- So no matter what type of image the application is using, it can use NSImage and have no knowledge of what's happening underneath the class

Façade

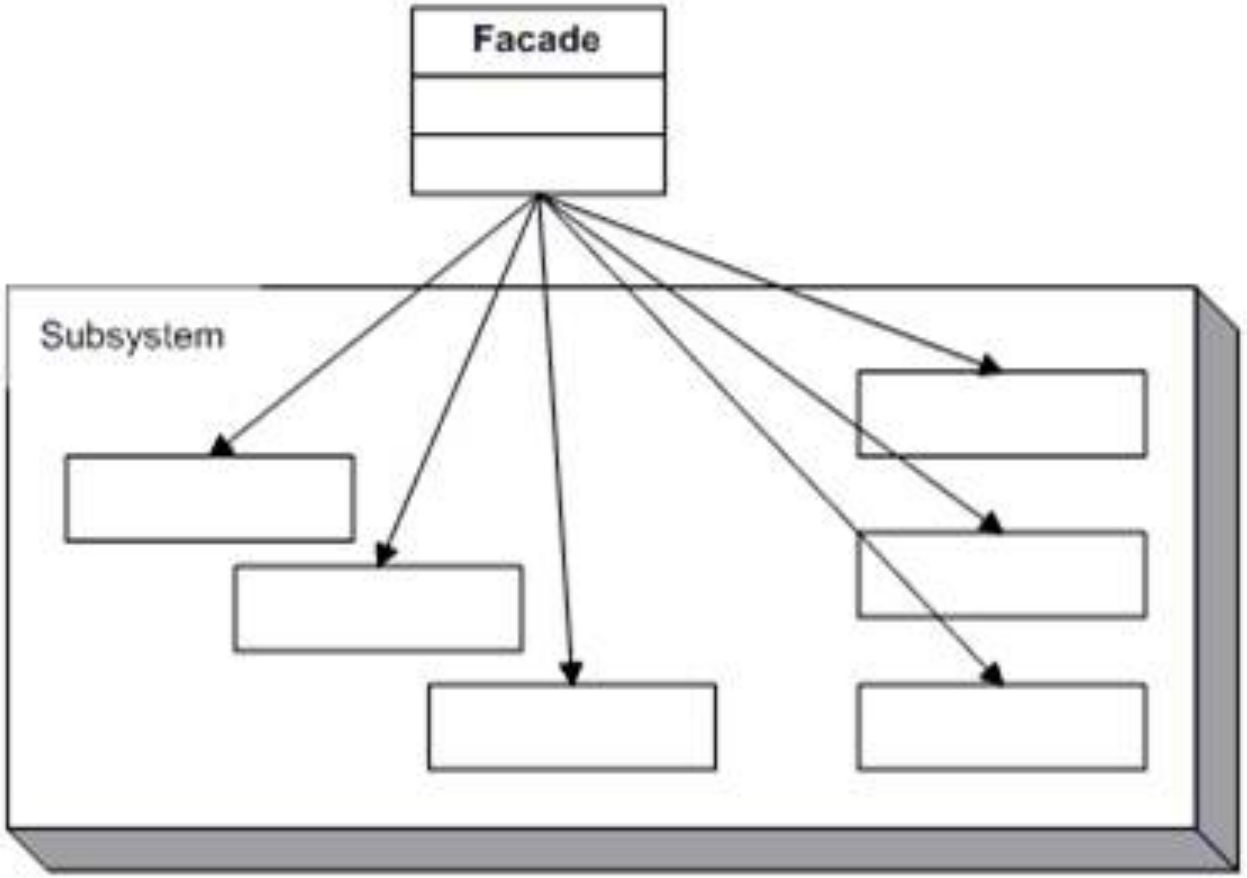

Image from: http://www.tutebox.com/2066/computers/programming/design-patterns-facade-pattern/

Command

- The command pattern is used for request encapsulation
- It allows for the separation of an object sending a message from the objects receiving the message
- The encapsulated request/message is then much more flexible and can be passed between objects, stored for later, dynamically modified, or placed on a queue

Command

- In iOS an example class that is used to encapsulate messages is NSInvocation
- These objects are used for undo management
- They contain a target, selector, arguments, and the return value
- These elements can be set directly and the return value is set automatically when the object is dispatched

Command

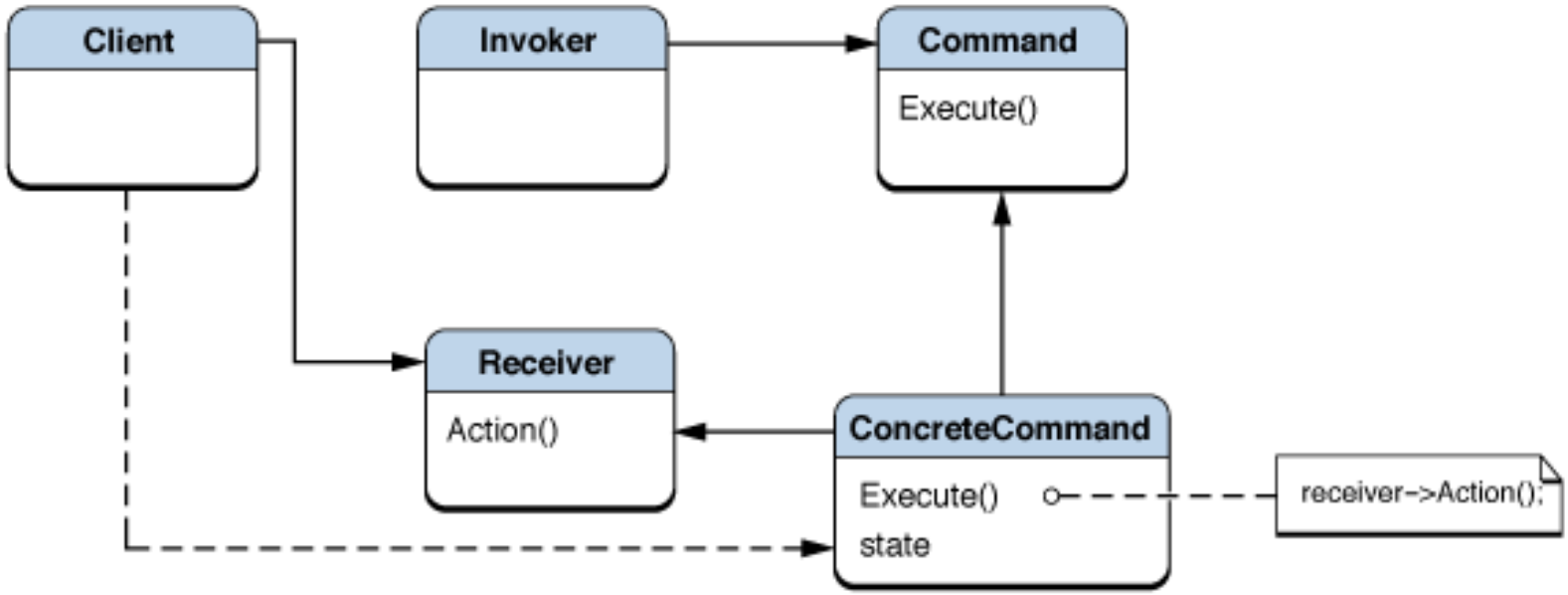

Image from: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html

Template Method

- The Template Method design pattern defines the skeleton of an algorithm, leaving some parts to be implemented by subclasses
- This allows subclasses to refine certain parts of the algorithm without changing the structure
- In iOS this lets parts of a program "hook" into an algorithm, but the framework still determines how they are needed

Template Method

- One example of this in iOS is the document architecture defined by AppKit, a framework
- Three classes are key to this architecture: NSDocument, NSWindowController, and NSDocumentController
- AppKit sends messages to each of these objects at runtime and then requests it to perform specific operations
- The developer needs to override many methods in these messages to add behavior specific to their application

Template Method

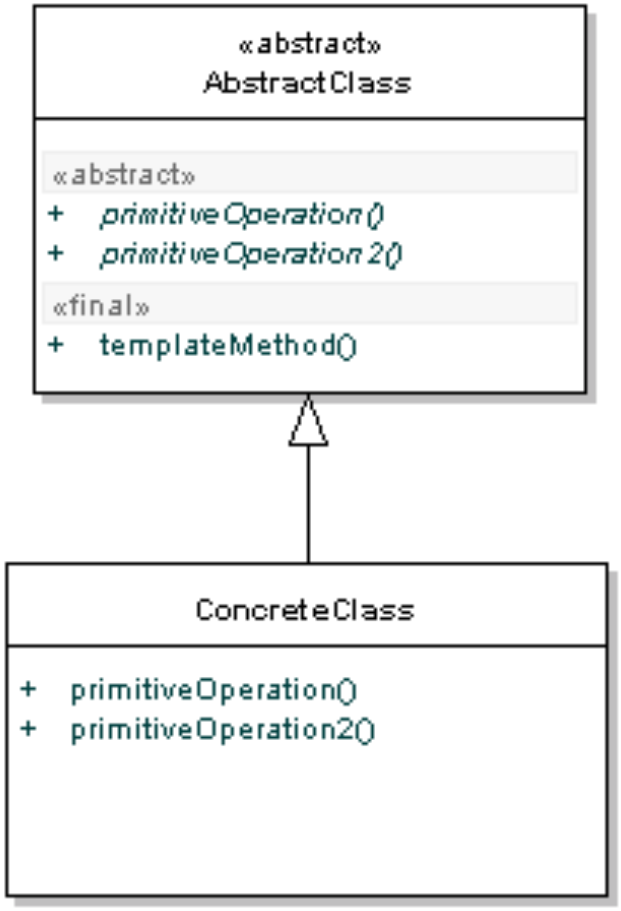

Image from: http://java.dzone.com/articles/design-patterns-template-method

Further Resources

- https://developer.apple.com/library/mac/#documentation/cocoa/conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html
- http://www.amazon.com/Pro-Objective-C-Design-Patterns-iOS/dp/1430233303