COM S 362
Object-Oriented Analysis & Design

Command and Observer Patterns

Reading

Alexander Shvets. Dive Into Design Patterns, 2020.

- Command
- Observer

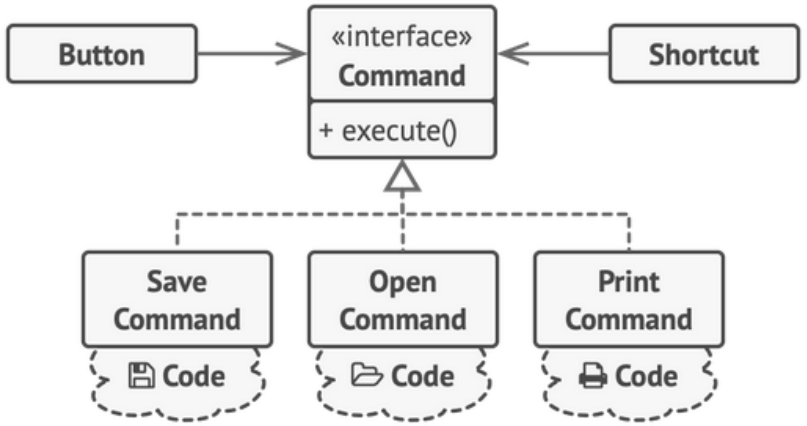
Behavioral Patterns

- Chain of Responsibility
  - Pass requests along a chain of handlers.
- Command
  - Turn a request (method calls) into an object.
- Iterator
  - Traverse elements of a collection.
- Mediator
  - Reduce dependencies between objects.
- Memento
  - Save and restore the state of an object.
- Observer
  - Publish and subscribe to events.
- State
  - State based behavior.
- Strategy
  - Define a family of algorithms, each in their own class
- Visitor
  - separate algorithms from the objects on which they operate

Command Pattern: Overview

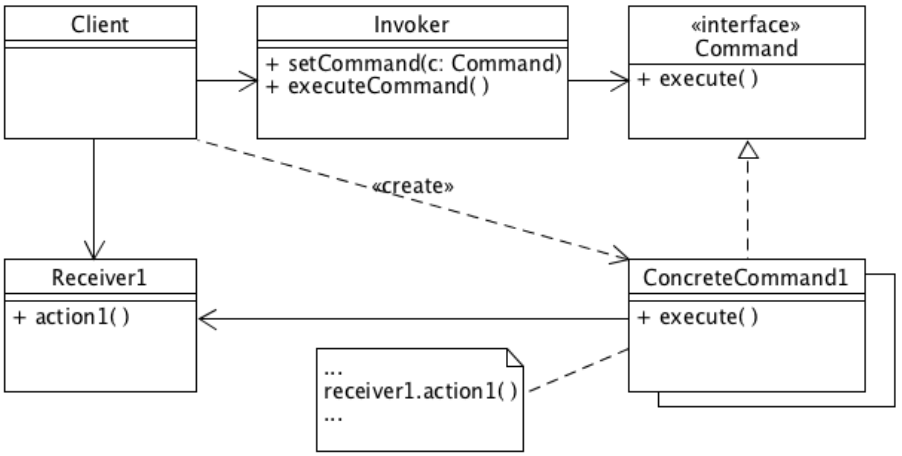
- Intent
  - Turn a request into a stand-alone object that contains all information about the request

Command Pattern: Problem/Solution



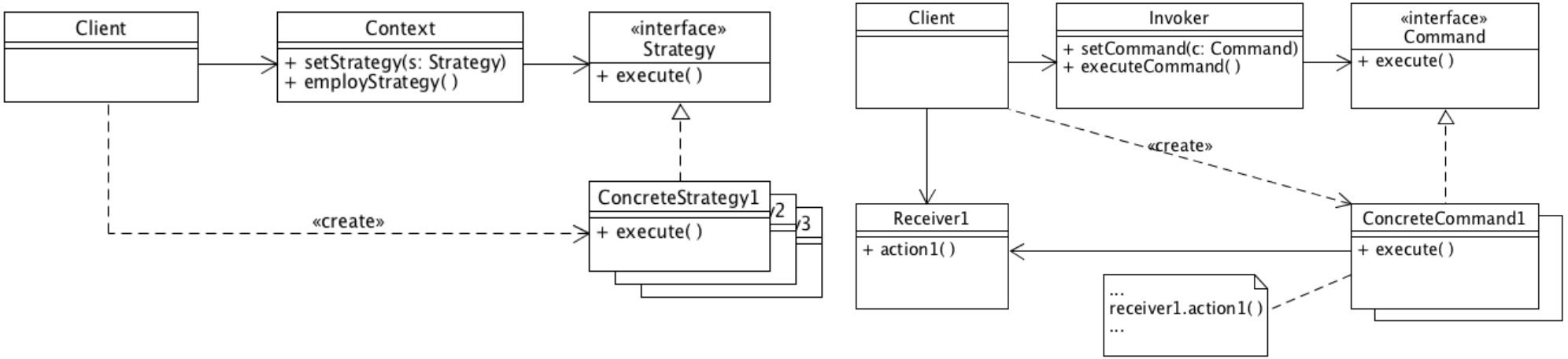
- Problem: You are creating a GUI and notice sequences of actions that are repeated for different inputs, for example, the user can save by clicking a button, going to the save menu or pressing ctrl+s
- Each action may require multiple steps, you want some way to avoid code duplication
- Solution: Encapsulate each action as a method in a Command class, the action can be performed by calling the method

Command Pattern: Structure



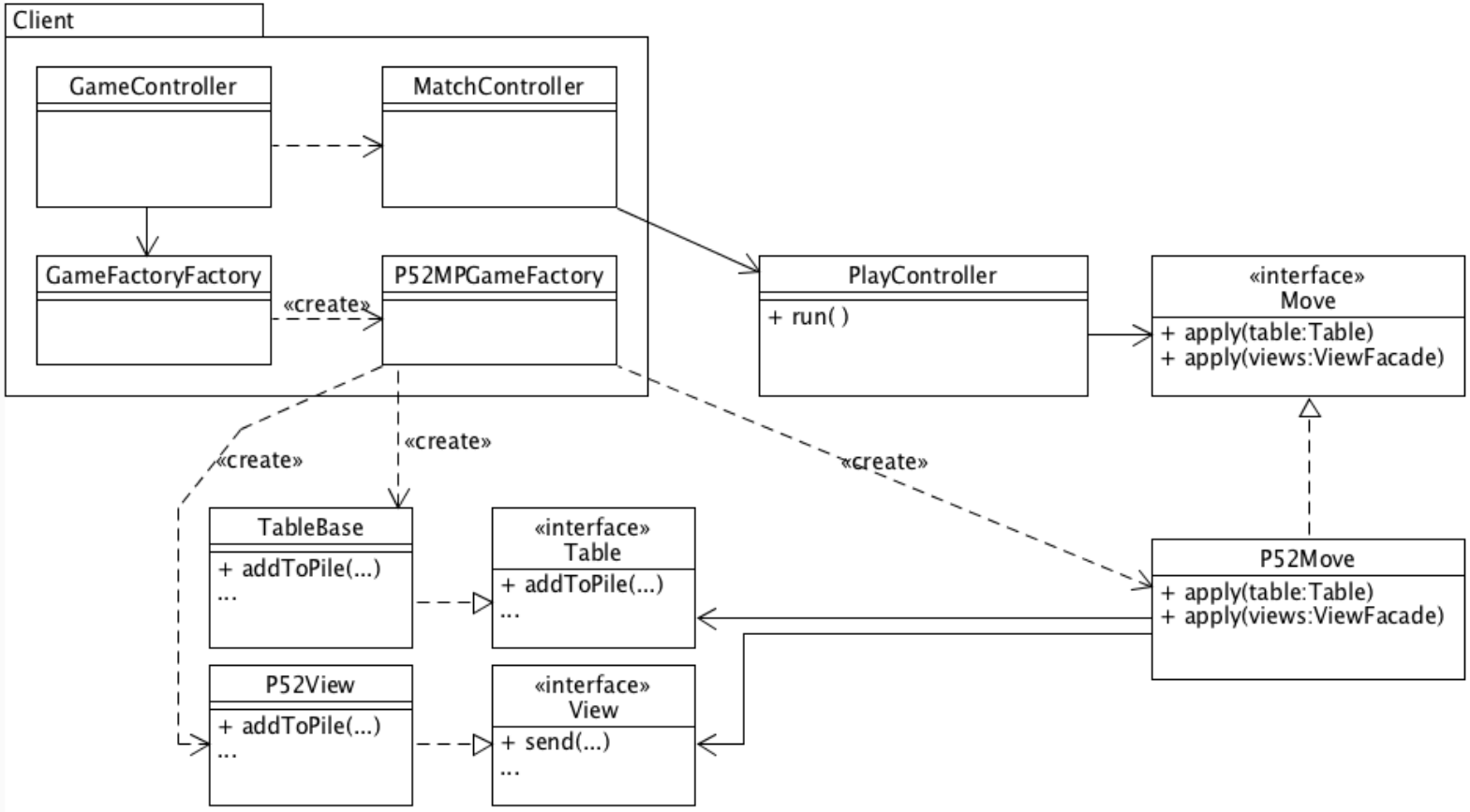
- Invoker: uses the command by calling execute( )
- Command: interface common to all supported commands
- ConcreteCommand: has execute( ) method that performs actions on the Receiver
- Receiver: the target of the command
- Client: creates a ConcreteCommand and sets it in the Invoker

Command vs Strategy Pattern



- Command is similar to Strategy Pattern
- The difference is the intension of Command Pattern is to perform a set of actions on a receiver
- Additional features sometimes provided with Command
  - Save commands for later execution
  - An undo( ) method that reverses the actions of execute( )

Command Pattern in Cards362



Discussion

- Advantages
  - Use to decouple classes that invoke operations from classes that perform these operations
  - Can implement undo/redo and deferred execution of operations
- Disadvantages
  - Adds an indirection making the code more complex

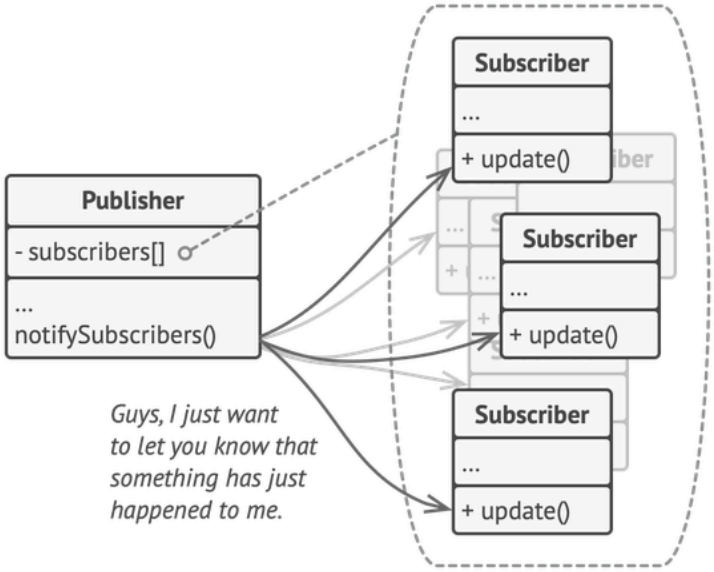
Behavioral Patterns

- Chain of Responsibility
  - Pass requests along a chain of handlers.
- Command
  - Turn a request (method calls) into an object.
- Iterator
  - Traverse elements of a collection.
- Mediator
  - Reduce dependencies between objects.
- Memento
  - Save and restore the state of an object.
- Observer
  - Publish and subscribe to events.
- State
  - State based behavior.
- Strategy
  - Define a family of algorithms, each in their own class
- Visitor
  - separate algorithms from the objects on which they operate

Observer: Overview

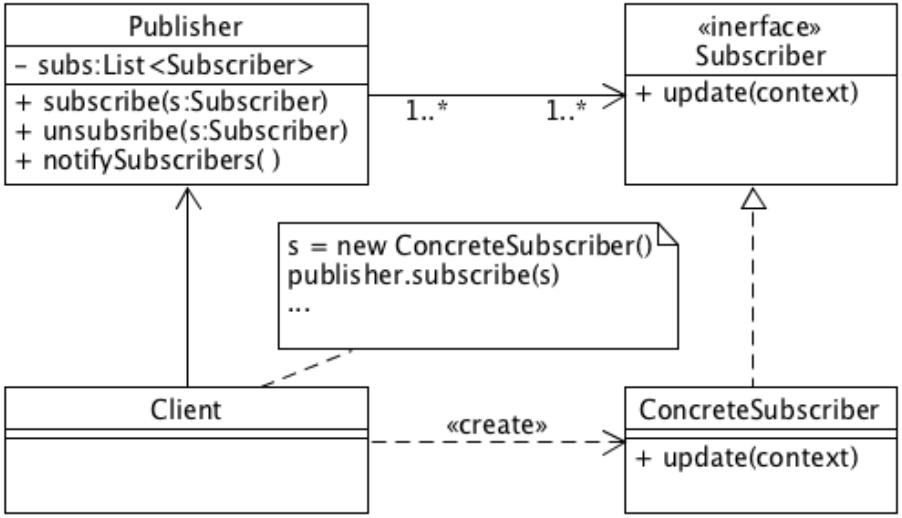
- Intent
  - Provide mechanism for objects to subscribe to information and other objects to publish information

Observer: Problem/Solution



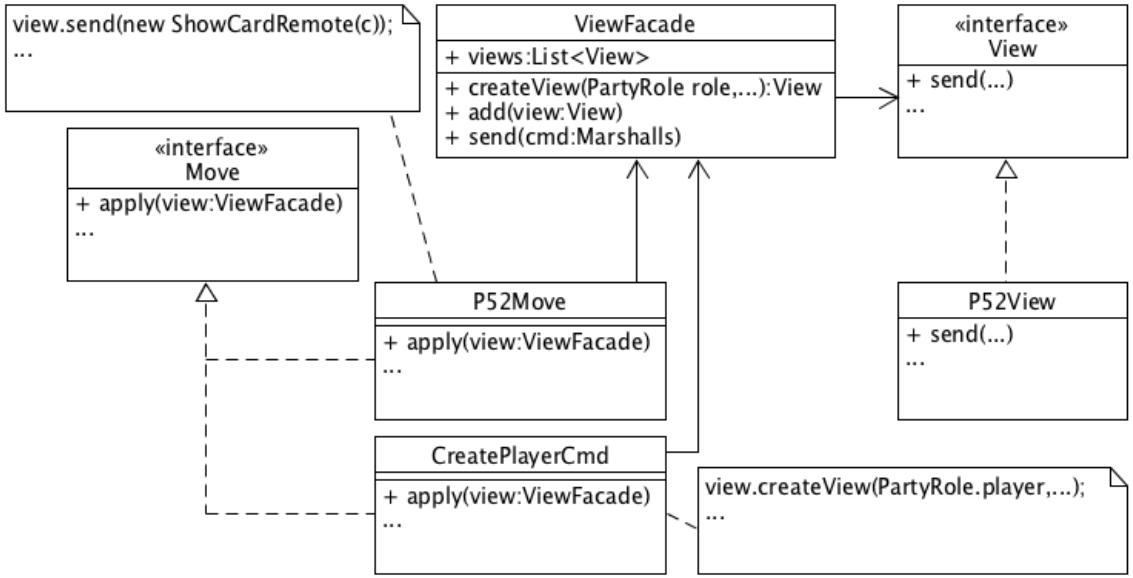
- Problem: You have an update event that might affect the state of multiple different objects, how to inform those objects of the event?
- If the source of the event directly calls the receivers, then it will be coupled to the receivers
- Solution: Create a Publisher class that maintains a list of Subscriber objects

Observer: Structure



- Publisher: maintains a list of subscribers that it can notify
- Subscriber: interface for object that can receive updates
- ConcreteSubscriber: wants to receive updates
- Client: adds subscribers to the publisher

Observer in Cards362



- A View is a player's perspective of the game
- CreatePlayerCmd calls createView( ) to add a view for a player
- Most Move objects (e.g., P52Move) call view.send( ) to send information to the views

Discussion

- Advantages
  - Removes coupling of information providers and receivers, neither need to know anything about the other
- Disadvantages
  - Extra logic require to route or prioritize notifications