COMP 504: Graduate Object-Oriented Programming and Design

Lecture 11: Command Design Pattern

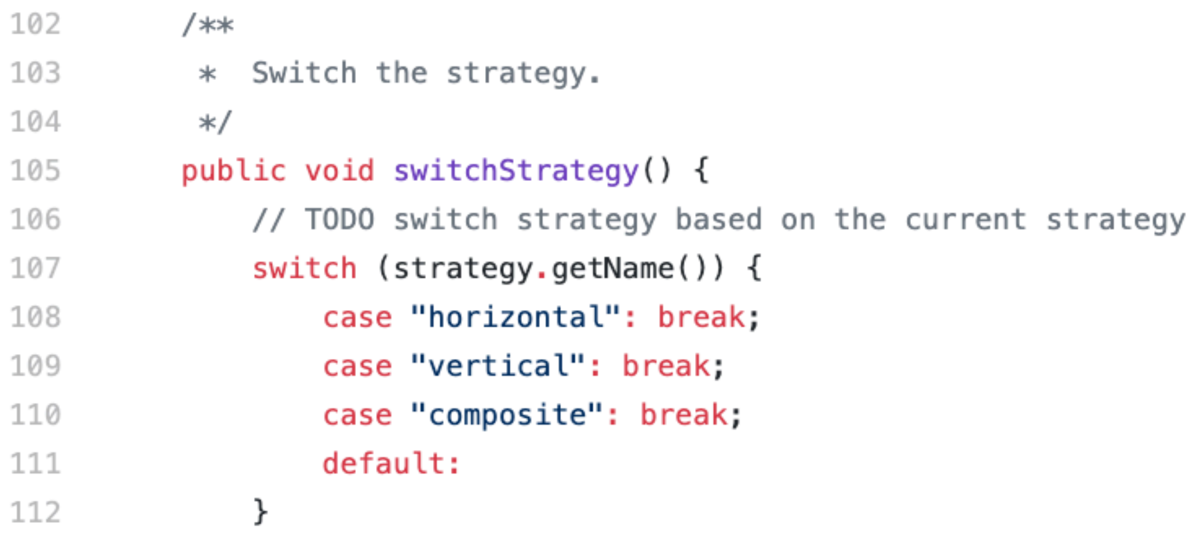
Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp504

Announcements & Reminders

- HW #2 due today at 11:59pm
- Quiz #2 due Wednesday, Sept 23rd at 11:59pm
- HW #3 will be available today, due Wed, Oct 7th at 11:59pm
- Use Piazza (public or private posts, as appropriate) for all communications re. COMP 504
- See course web site for syllabus, work assignments, due dates, office hours schedule.

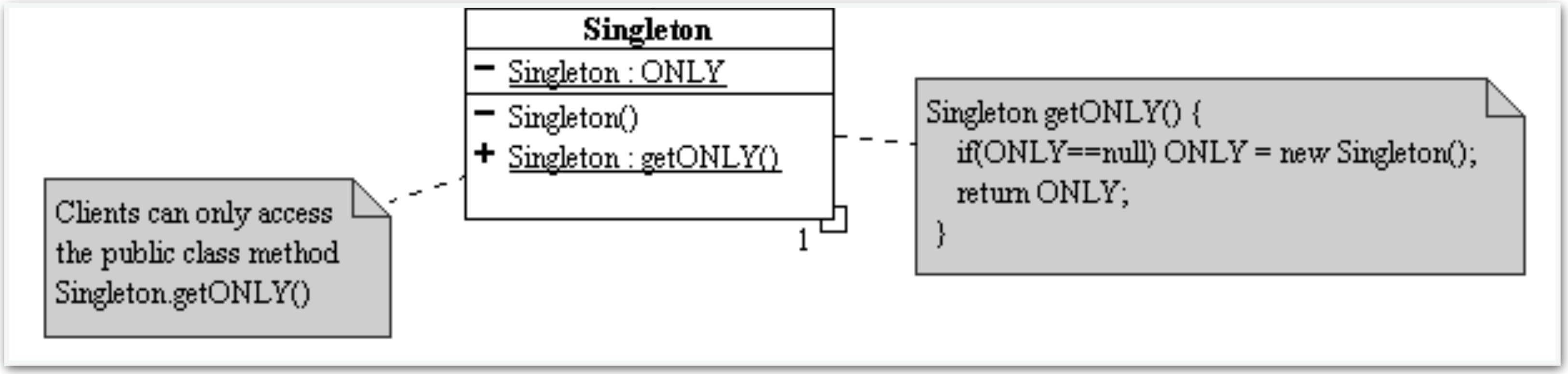
Ex 4: Switch Strategy (Singleton Design Pattern)



```java
    /**
     *  Switch the strategy.
     */
    public void switchStrategy() {
        // TODO switch strategy based on the current strategy
        switch (strategy.getName()) {
            case "horizontal": break;
            case "vertical": break;
            case "composite": break;
            default:
        }
```

Singleton Design Pattern

- The *singleton* design pattern is used when only 1 instance of the class is needed
- The constructor is made private to prevent users from creating more than 1 instance
  - use public static method (e.g. getONLY) to access instance
- Static field ONLY is not automatically initialized


Singleton
− Singleton : ONLY
− Singleton()
+ Singleton : getONLY()
1
Clients can only access the public class method Singleton.getONLY()

```
Singleton getONLY() {
   if(ONLY==null) ONLY = new Singleton();
   return ONLY;
}
```

Singleton Design Pattern Transition

- Create a private static field in the strategy classes
  - Type should be IUpdateStrategy
  - Not automatically initialized, beneficial when initialization cost is expensive
- The field will represent the single instance of each class
- The field must be static because a static method will be used to access the field representing the single class instance

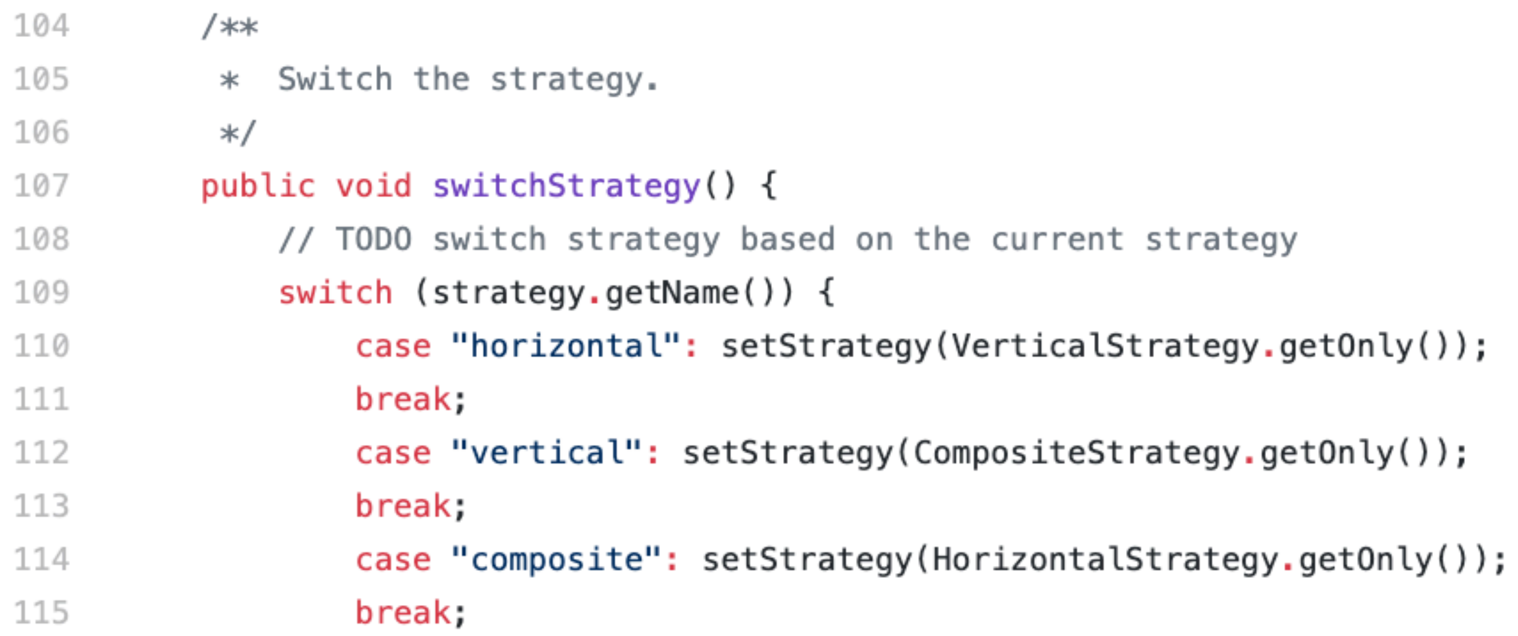
Singleton Design Pattern Transition

- Create a public static method called *getOnly* that will return the static field
  - Initialize field if currently null
  - Return type should be IUpdateStrategy

- Only way to access private static field is through this method

- The method must be static because the user cannot create an instance of the class

MovingLine Strategies



```java
    /**
     *  Switch the strategy.
     */
    public void switchStrategy() {
        // TODO switch strategy based on the current strategy
        switch (strategy.getName()) {
            case "horizontal": setStrategy(VerticalStrategy.getOnly());
            break;
            case "vertical": setStrategy(CompositeStrategy.getOnly());
            break;
            case "composite": setStrategy(HorizontalStrategy.getOnly());
            break;
```

Singleton Design Pattern Transition

- Update *switchStrategy* to call *getOnly* or *makeStrategy* method
- When should you possibly keep CompositeStrategy constructor public?

Command Design Pattern

- Two objects communicate by sending a command from one to the other
  - Invoker (first object)
  - Recipient (second object)

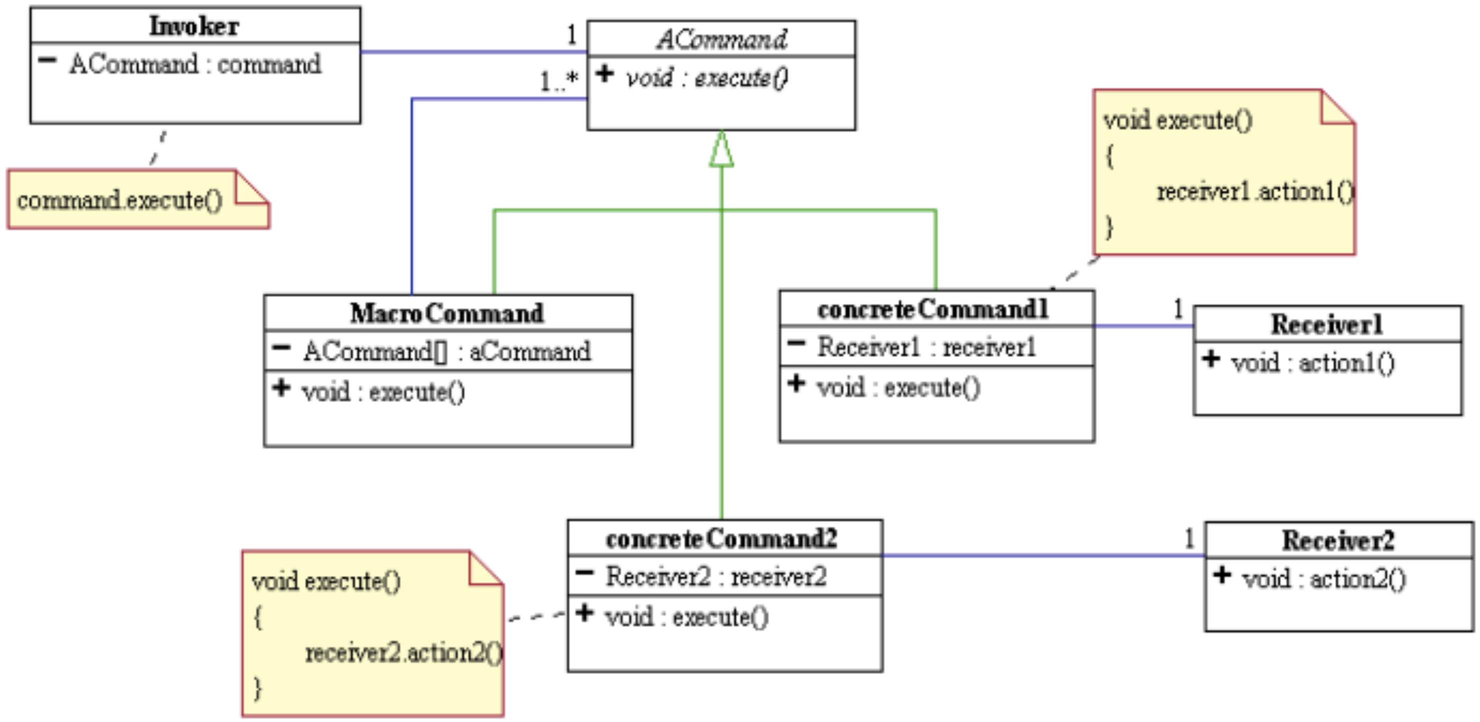
- Invoker holds reference to command instead of recipient, doesn't care who is the recipient

- Invoker sends a command by executing a command method

- Command object dispatches command to specific recipient

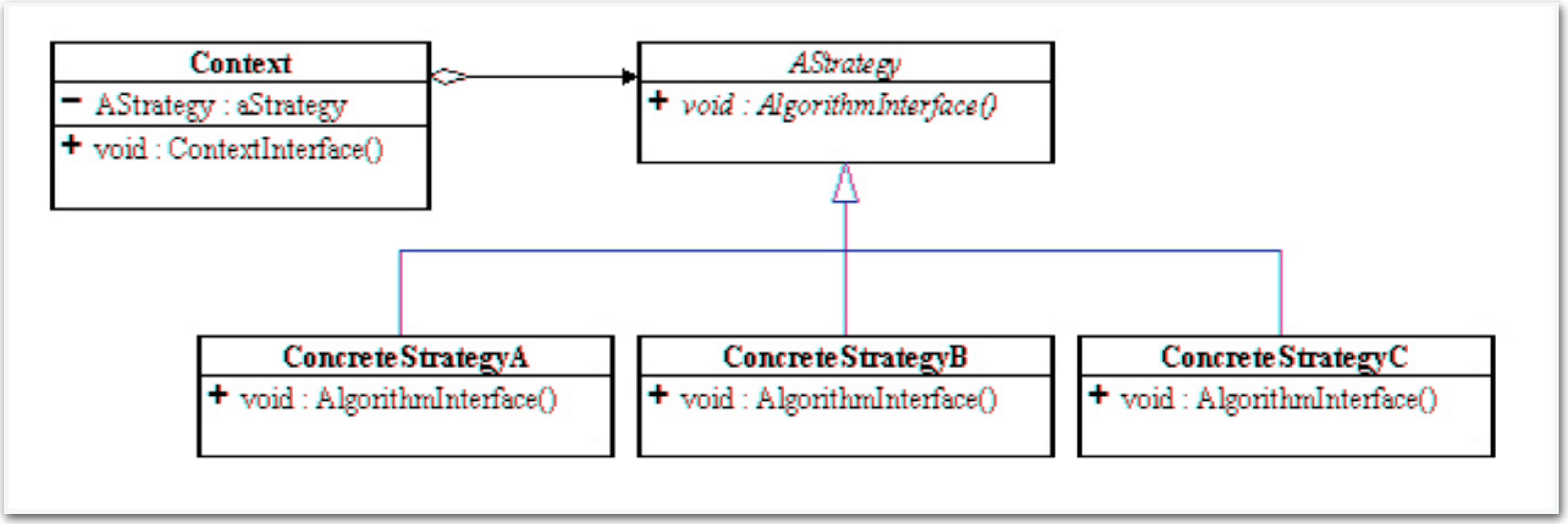
Command Design Pattern

- Invoker holds abstract command
  - issues command by calling command execute
- Concrete command calls specific action on receiver



Strategy Design Pattern

- A *strategy* implements the behavior of the *Context* class
  - the context is a concrete class
  - strategy should be connected to a context
- The context is invariant, the strategy (behavior) is variant
- The strategy design pattern uses the union design pattern


Context
− AStrategy : aStrategy
+ void : ContextInterface()
AStrategy
+ void : AlgorithmInterface()
ConcreteStrategyA
+ void : AlgorithmInterface()
ConcreteStrategyB
+ void : AlgorithmInterface()
ConcreteStrategyC
+ void : AlgorithmInterface()

Command vs Strategy Design Pattern

What are the similarities/differences between the command and strategy design patterns?

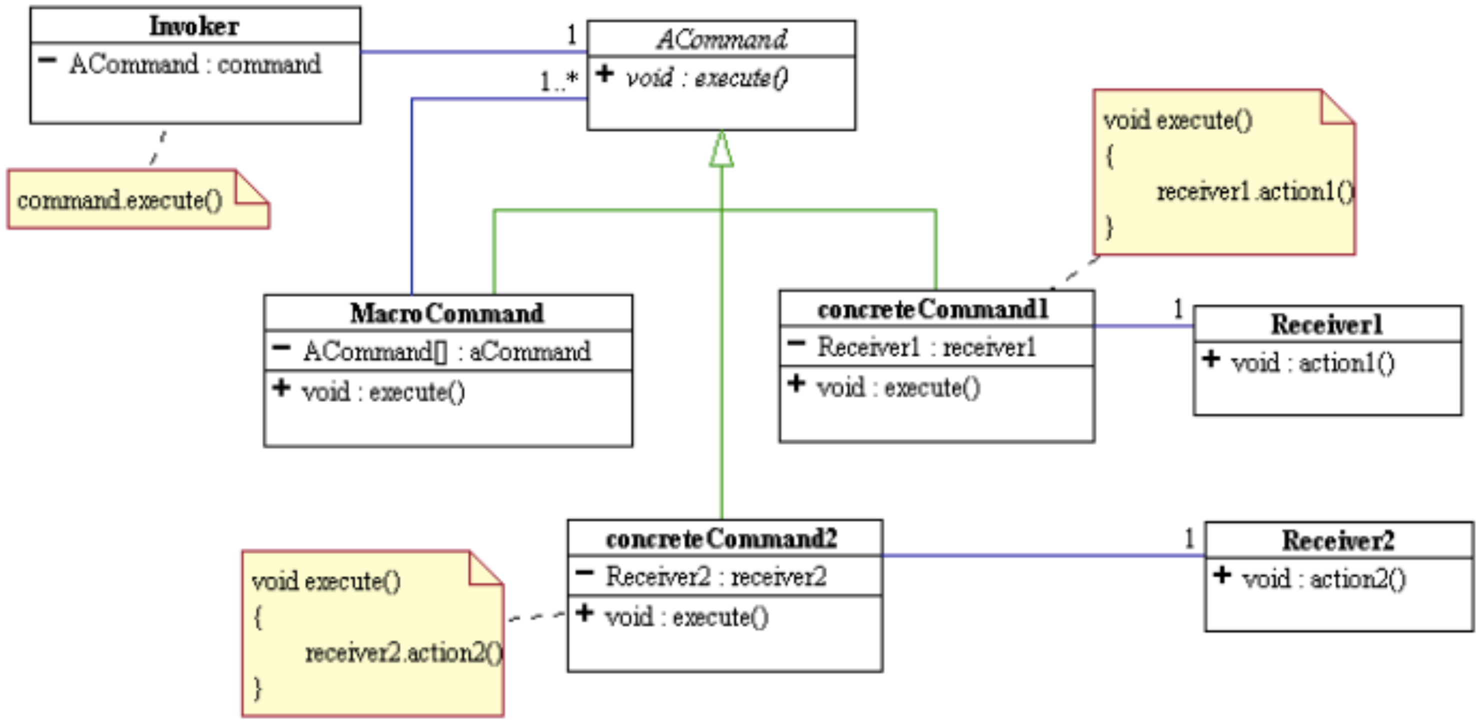
Command vs Strategy Design Pattern

- Both have a receiver (context)
  - update the receiver, call action that affects behavior (BallWorld).
- Strategy
  - invoker directly calls strategy (action)
- Command
  - invoker directly calls command which invokes receiver action

Why use a command design pattern over the strategy design pattern?

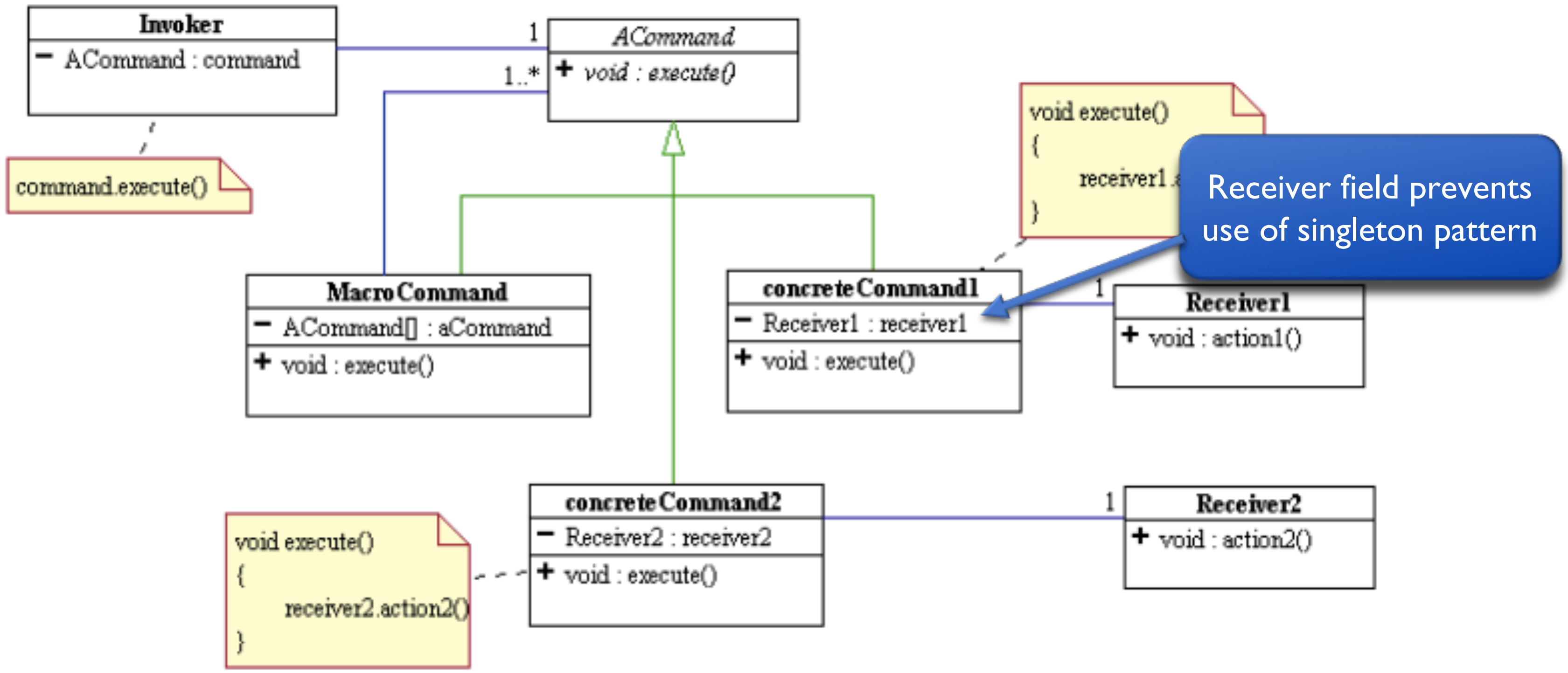
Command Design Pattern

Could the singleton design pattern be used with the command design pattern given the UML diagram below?



Command Design Pattern

Could the singleton design pattern be used with the command design pattern given the UML diagram below?



Worksheet #7: Command Design Pattern

How would you change the command design pattern to be able to use the singleton design pattern?