CSSE 374: Exam 3

# Policy

This is an open-book, open-note, open-Moodle exam. The work you do must be novel and your own original work. You can use Moodle, any URLs that are directly reachable from Moodle, and your eBook if you have one. You are not allowed to use any other online resources. This exam contributes **10%** to your final course grade.

Design Analysis (D)

This category explores your insights into the fundamentals of good design. It is worth 25% of the exam grade. To earn a grade on the exam, you must get at least half of the questions correct.

1. A key similarity between Adapters and Decorators is that they both compose an external object and delegate their method calls to the composed object. What are key differences between Adapters and Decorators?

2. A key similarity between Command and Strategy is that they both favor composition over inheritance. Furthermore, they both encapsulate some algorithms. What are their key differences?

3. Template method is a way for a super type to let subtypes hook in custom algorithms for the pieces that can change in the future. It would also be possible to achieve the same behavior using Strategy. Do any of these two approaches violate the Hollywood Principle? Explain the tradeoffs you are making when you use Template over Strategy and vice versa.

4. Iterators and Streams both allow traversal of composite objects. Why would you prefer using a Stream over an Iterator?

5. When you add an ActionListener to a JButton, are you adding an Observer or a Command? Explain.

6. Which of the following design principle(s) are violated in the Façade pattern? Give your reasoning for each of the following:

a. Separate what changes from what remains the same

b. Favor composition over inheritance

c. Dependency Inversion Principle

d. Single responsibility Principle

e. Open-Closed Principle

# Design Synthesis

In the discussion of the Command Pattern, Freeman describes a Macro Command that operates many devices. On this exam, we will add programmable macros to the remote control. Buttons 6-10 will be reserved for macros.

Programmable remote controls are never fun to program, but from the end user's perspective, the basic functionality is:

* When the user presses a macro's On button, the macro will turn on all devices it knows about.
* When the user presses a macro's Off button, the macro will turn off all devices it knows about.
* When the user holds down a macro's On button and presses a different command's On button, the macro will add the command held in the slot to itself. Users may add macros to other macros in this way.
* When the user holds down a macro's On button and presses a different command's Off button, the macro will remove the command held in the slot from itself. Users may remove macros from other macros in this way **no matter how deeply nested the descendent macros are**.

Freeman's **RemoteControl** class has four event handlers that get called when the user invokes each of the above features:

**public** **void** onButtonWasPushed(**int** slot)

**public** **void** offButtonWasPushed(**int** slot)

**public** **void** addToMacro(**int** slot\_to\_add, **int** macro\_slot)

**public** **void** removeFromMacro(**int** slot\_to\_remove, **int** macro\_slot)

We’ve seen the first two event handlers before; leave them the same. The last two event handlers are new, but are not yet implemented. Your job is to invent a new macro Command type to facilitate your implementation of the last two event handlers.

Follow the **Hollywood Principle**: Commands must **not** know about the RemoteControl.

The remote manufacturer already wrote Commands for numerous devices, so you must preserve *binary compatibility* of the Command interface:

* You are not allowed to change any existing concrete Commands.
* You are not allowed to change the existing method signatures of the Command interface.
* You must supply default implementations for any new methods you add to the Command interface. Use best-practice for defining these methods.

## C: UML

* Design the UML for this system for both the B and A parts. If you augment Freeman's Command, add the "default" keyword to the new methods you add to the interface to indicate your intent.
* In a note on the UML, write a code snippet with a sequence of RemoteControl event handler calls to set, invoke, and remove macros. In particular, demonstrate using the RemoteControl’s event handlers to nest macros inside of other macros and then remove them.
* Follow the Hollywood Principle: Commands must not know about the RemoteControl.

## B: Macro

* Code the composite version of the Macro command.
  + Add a new method to Command for adding new child Commands.
  + Only macros have child Commands; non-macro Commands must use best practice to indicate that they do not support children.
* In RemoteControl:
  + Implement addToMacro
  + Implement removeFromMacro using **internal iteration**.
  + Leave onButtonWasPushed and offButtonWasPushed unchanged.
* Demonstrate that your part C code snippet works as-is.

## A: Removable CompositeIterator

* Make Commands **Iterable**.
  + Provide default implementations for all of Iterable’s methods in Command.
* Write a composite iterator for your Macro that implements all of Iterator, including the **Iterator.remove**() method to remove the last returned element from the macro.
* In Remote, implement removeFromMacro using external iteration.