The only thing our MacroCommand is missing is its undo functionality. When the undo button is pressed after a macro command, all the commands that were invoked in the macro must undo their previous actions. Here's the code for MacroCommand; go ahead and implement the undo() method:

```
public class MacroCommand implements Command {
    Command[] commands;

    public MacroCommand(Command[] commands) {
        this.commands = commands;
    }

    public void execute() {
        for (int i = 0; i < commands.length; i++) {
            commands[i].execute();
        }
    }

    public void undo() {

    }
}
```

there are no Dumb Questions

Q: Do I always need a receiver? Why can't the command object implement the details of the execute() method?

A: In general, we strive for "dumb" command objects that just invoke an action on a receiver; however, there are many examples of "smart" command objects that implement most, if not all, of the logic needed to carry out a request. Certainly you can do this; just keep in mind you'll no longer have the same level of decoupling between the invoker and receiver, nor will you be able to parameterize your commands with receivers.

Q: How can I implement a history of undo operations? In other words, I want to be able to press the undo button multiple times.

A: Great question. It's pretty easy actually; instead of keeping just a reference to the last Command executed, you keep a stack of previous commands. Then, whenever undo is pressed, your invoker pops the first item off the stack and calls its undo() method.

Q: Could I have just implemented party mode as a Command by creating a PartyCommand and putting the calls to execute the other Commands in PartyCommand's execute() method?

A: You could; however, you'd essentially be "hardcoding" the party mode into PartyCommand. Why go to the trouble? With MacroCommand, you can decide dynamically which Commands you want to go into PartyCommand, so you have more flexibility using MacroCommands. In general, MacroCommand is a more elegant solution and requires less new code.