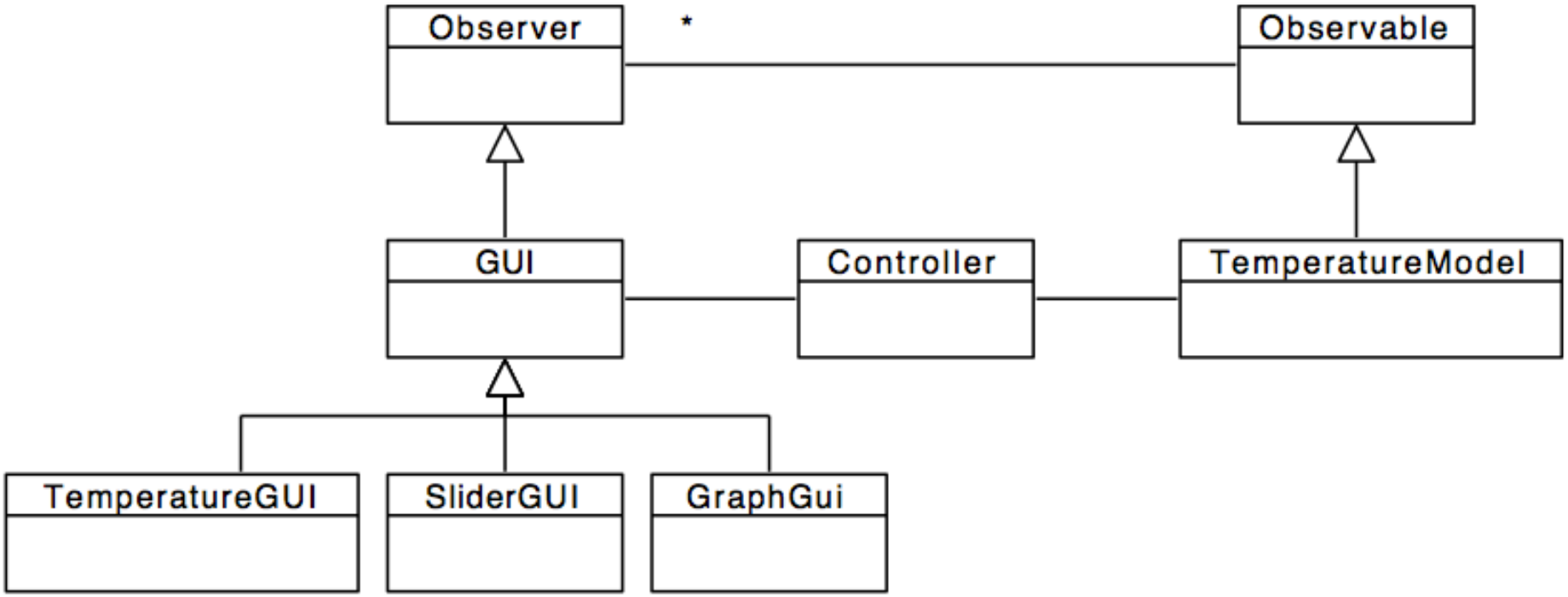
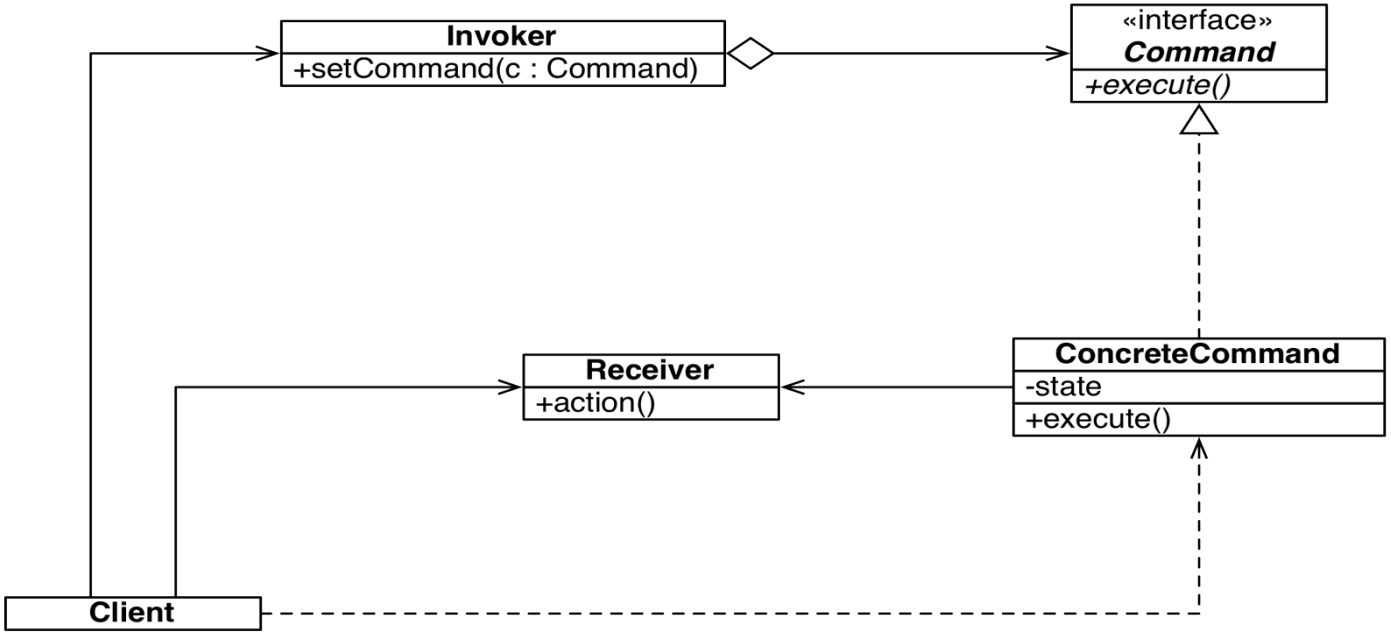
Sample Solution for Exercise #2

- Necessary Changes
  1. Create Command interface
     - Add execute, undo, and redo methods
  2. Create concrete Command classes
     - Create a subclass for RaiseTemperature, LowerTemperature, and SetTemperature
  3. Implement the Invoker
     - Implement command execution
     - Implement undo & redo stacks
     - Delegation of undo and decrease to the Invoker
  4. Add MenuItems to the View
     - Add an ActionListener for each of the menu items
     - Within the ActionListeners call the Invoker
- Note that you did not have to change
  - The view (apart from *adding* menu items, and wiring them)
  - The model (i.e. Receiver)
- In following sample code we show only the implementation of the LowerTemperature command.

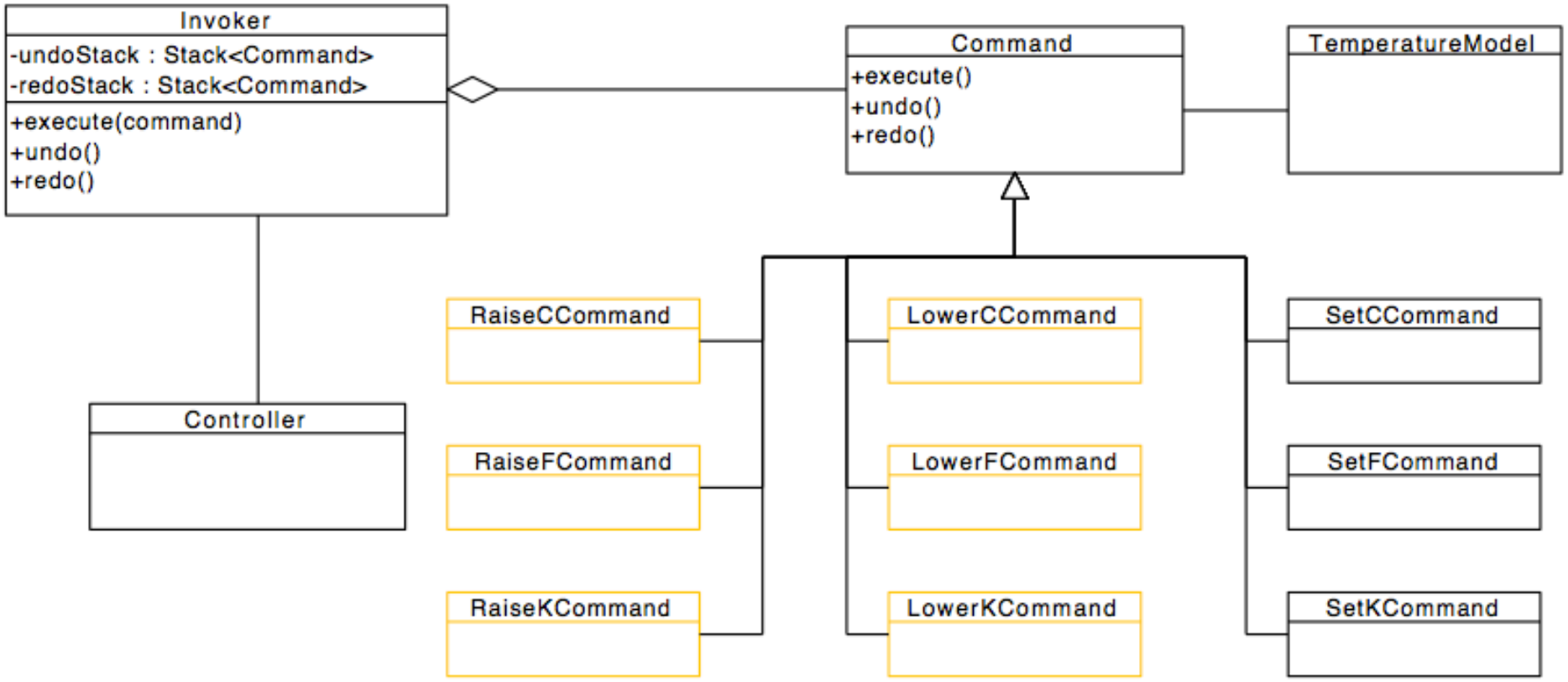
UML Diagram of the Temperature Application



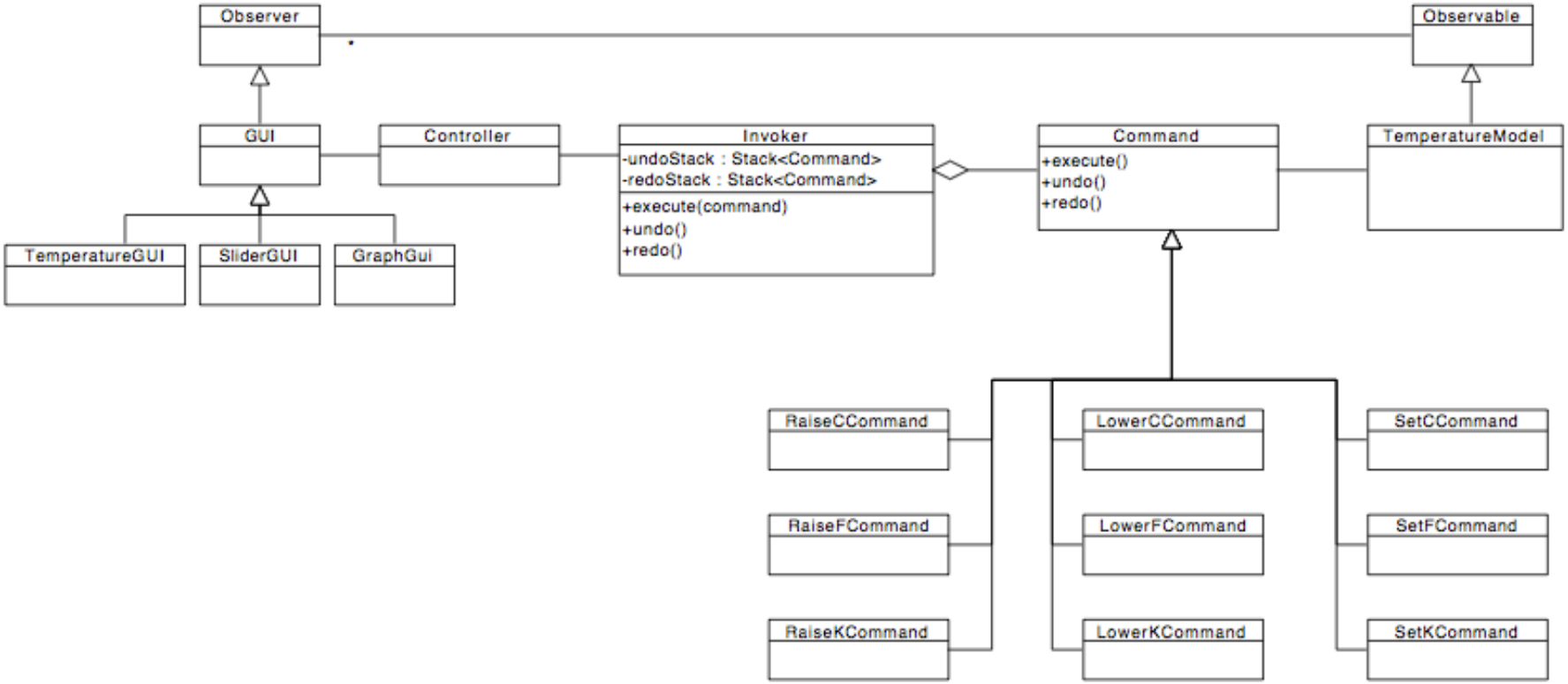
Review: Command Pattern



Create Command Class and Concrete Command Classes



Integrate with the Views



1. Create Command interface

```
/**
 * Represents an abstract TemperatureConverter command.
 * All commands shall implement this interface.
 */
public interface Command {

/**
 * Execute the command.
 */
public void execute();

/**
 * Undo the command.
 */
public void undo();

/**
 * Redo the command.
 */
public void redo();
}
```

2. Create concrete Command classes

```
public class LowerCCommand extends Command {

    public LowerCCommand(TemperatureModel model) {
     this.model = model;
    }

    public void execute() {
     model.setC(model.getC()-1);
    }

    public void redo() {
     model.setC(model.getC()-1);
    }

    public void undo() {
     model.setC(model.getC()+1);
    }
}
```

3. Implement the Invoker

```
public class Invoker extends java.util.Observable{

// Stack containing undone commands
private Stack<Command> undoStack = new Stack<Command>();

// Stack containing done commands
private Stack<Command> redoStack = new Stack<Command>();

// Execute the given command
public void execute(Command cmd) {
    redoStack.clear();
    undoStack.push(cmd);
    cmd.execute();
    setChanged();
    notifyObservers();
}

// undo and redo source  code are available in Moodle

// Needed for greying out undo menu item
public boolean isUndoable(){
    return !undoStack.empty();
}

// TODO: Implement isRedoable()
```

3. (ctd) Delegation of undo and decrease to the Invoker

```
public class Controller{

// Holds a reference to the Invoker
private Invoker manageInvoker = new Invoker();

// Delegate decrease coming from the View to the Invoker
public void decreaseC() {
    manageInvoker .execute(new LowerCCommand(model));
}

// Delegate undo to the Invoker
public void undo() {
    manageInvoker .undo();
}
```

4. Add MenuItems to the View

```
private MenuItem lowerItem = new MenuItem("Lower Temperature");
private MenuItem redoItem = new MenuItem("Redo");
private MenuItem undoItem = new MenuItem("Undo");

// Within the view constructor:
lowerItem.addActionListener(new LowerTempListener());

// Add action listener and delegate to the controller
class LowerTempListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
     controller.decreaseC();
    }
}

// This greys out undo / redo when not available
public void update(Observable arg0, Object arg1) {
    Invoker manager = (Invoker)arg0;
    undoItem.setEnabled(manager.isUndoable());
    redoItem.setEnabled(manager.isRedoable());
}
```