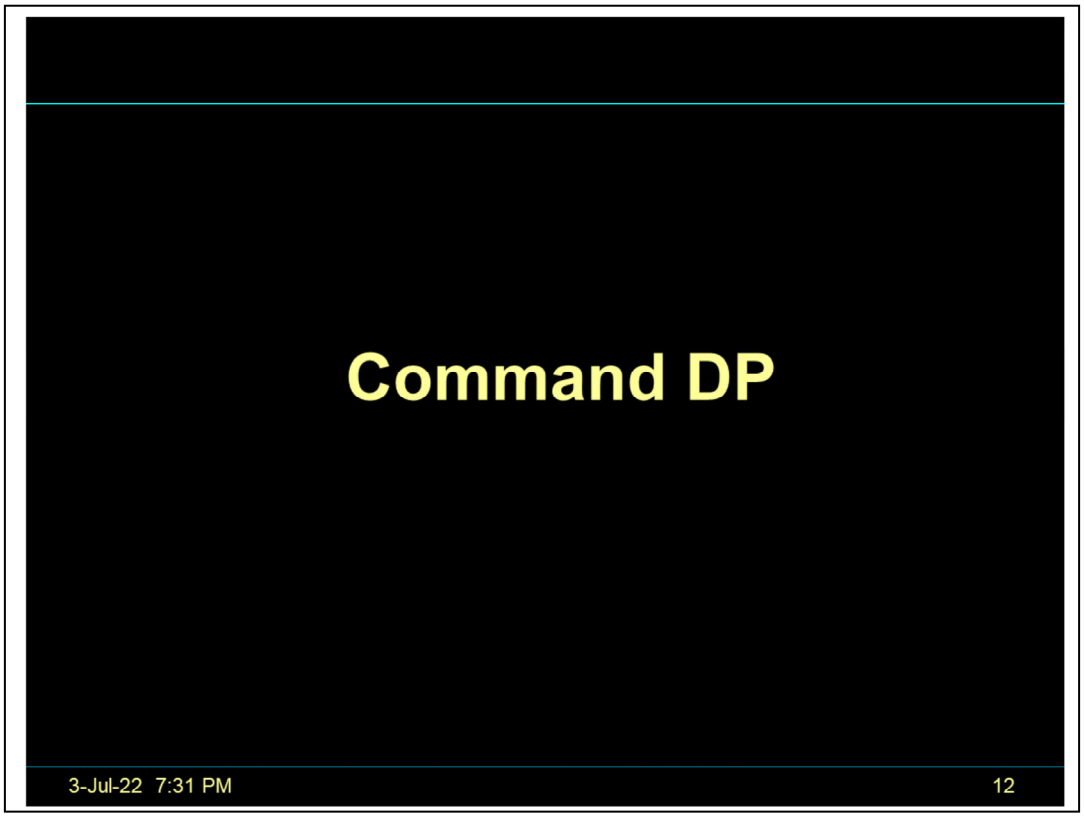
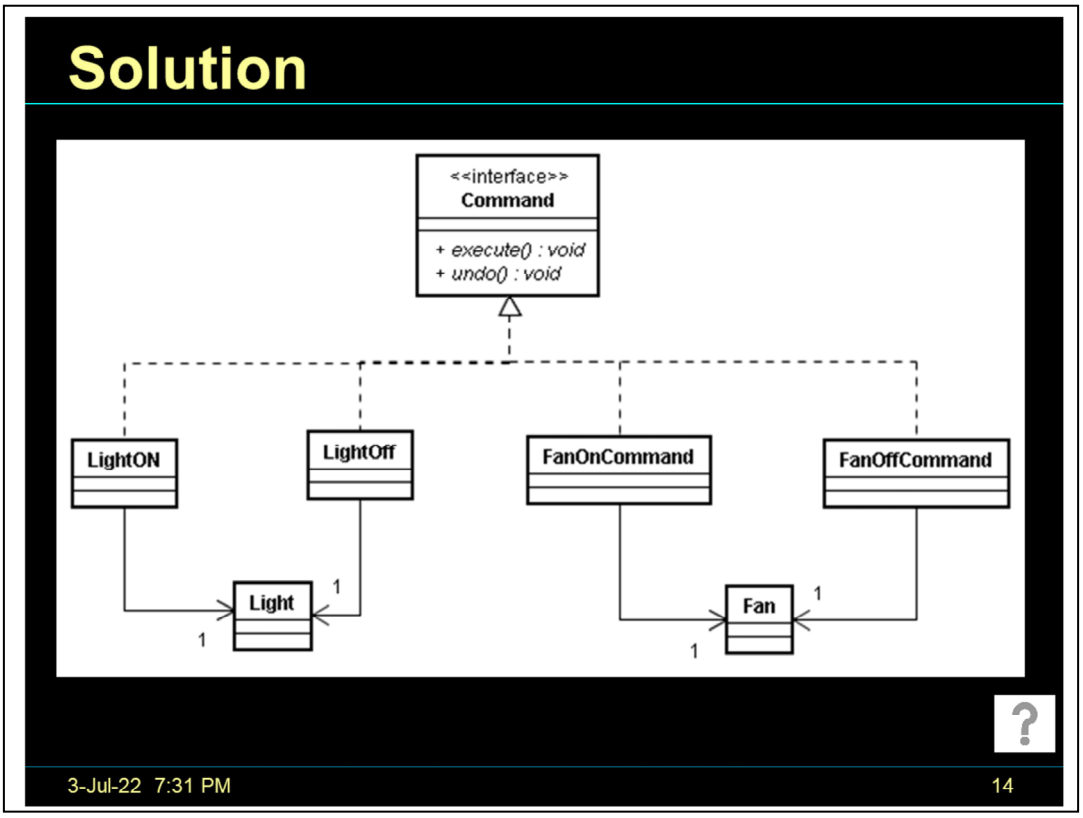

Command DP
3-Jul-22 7:31 PM
12

Problem

- A remote control is used to control home appliances.
  - We have buttons on the remote for various appliances. Each button does a different task.
  - The remote has an undo button.
  - For each device, the vendor has supplied an API.
  - Design an API for the remote such that it can handle the existing devices, and also any new devices added later on.



This pattern encapsulates a request as an object. It effectively makes a method (of an object) a new object.

They can be manipulated and extended like other objects.

A set of commands can be assembled to form a new command.

It is easy to add new commands because we don't have to change existing commands.

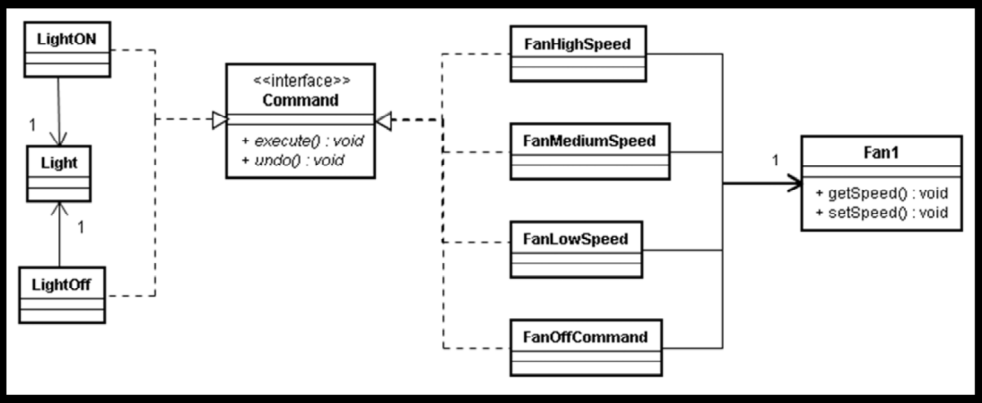
It decouples the requester from the performer.

Code present.

Problem

- The ceiling fan has buttons on the remote
  - high speed
  - low speed
  - medium speed
  - off
- How do we implement these features.
  - Undo must still be supported

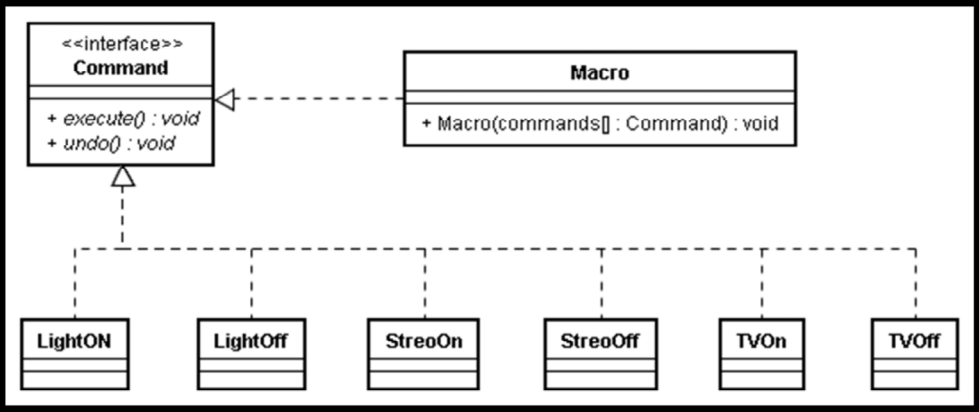
Solution


LightON
1
Light
1
LightOff
<<interface>>
Command
+ execute() : void
+ undo() : void
FanHighSpeed
FanMediumSpeed
FanLowSpeed
FanOffCommand
1
Fan1
+ getSpeed() : void
+ setSpeed() : void

Problem

- The remote must have buttons for party
  - PartyOn mean lights on, stereo on, TV on
  - PartyOff means lights off, stereo off and TV off

Solution


<<interface>>
Command
+ execute() : void
+ undo() : void
Macro
+ Macro(commands[] : Command) : void
LightON
LightOff
StreoOn
StreoOff
TVOn
TVOff

Solution

- We create two arrays of commands

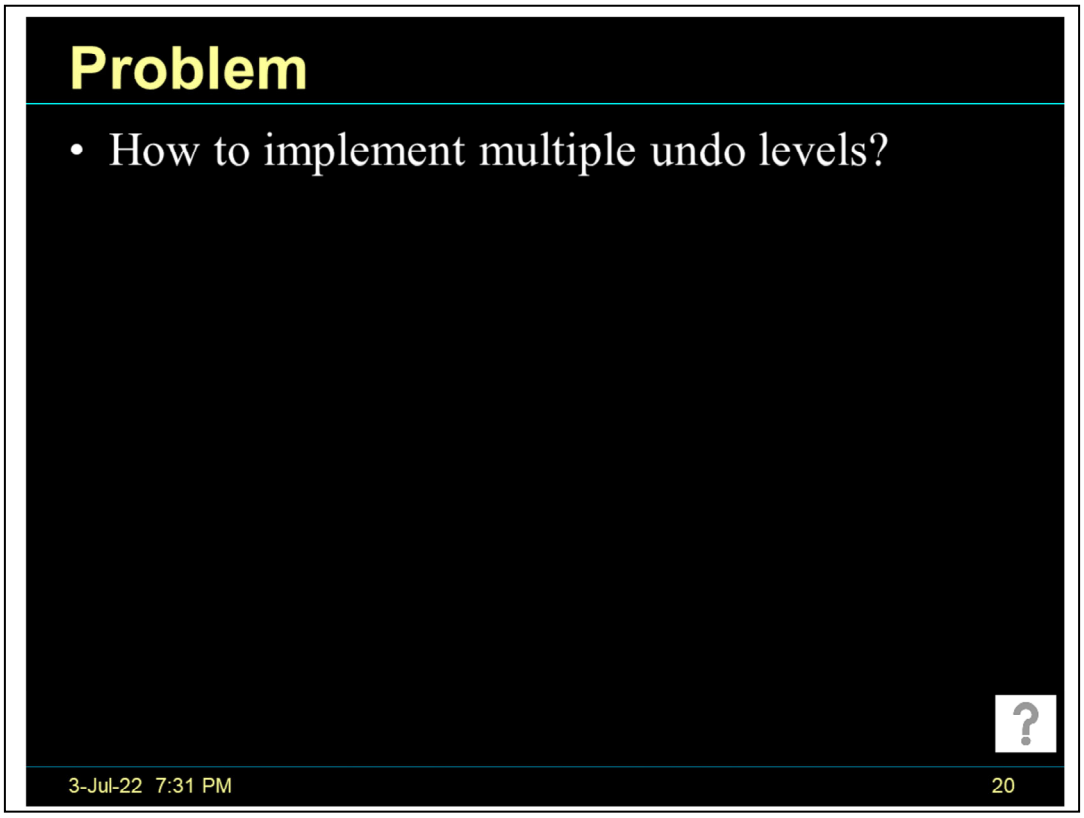
```
Command[] partyOn =
  {lightOn, stereoOn, tvOn};
Command[] partyOff =
  {lightOff, stereoOff, tvOff};
```

- One remote button is assigned

```
Macro(partyOn);
```

- Another remote button is assigned

```
Macro(partyOff);
```



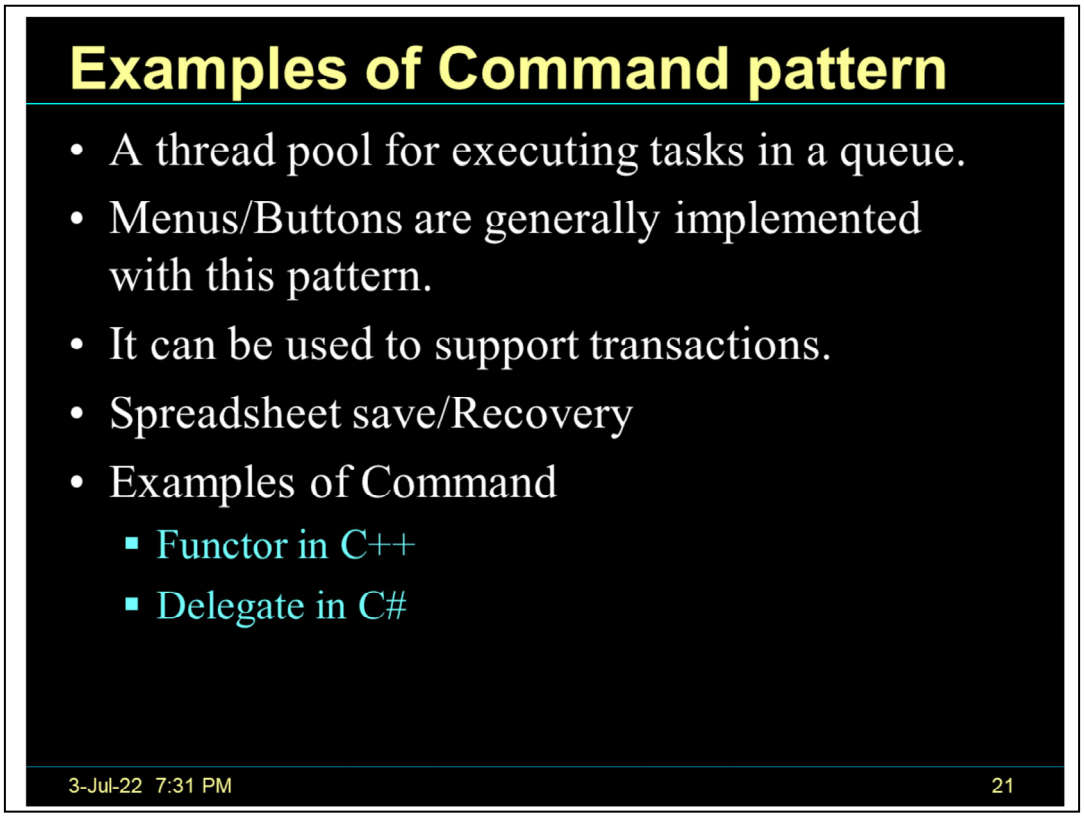
How to implement multiple undo levels?

Solution:

Keep a stack of previous commands.

Note: Unlimited-level undo and redo can be achieved by traversing the list of commands backwards and forwards.

Similarly redo / log can be supported.



Menu/Buttons:

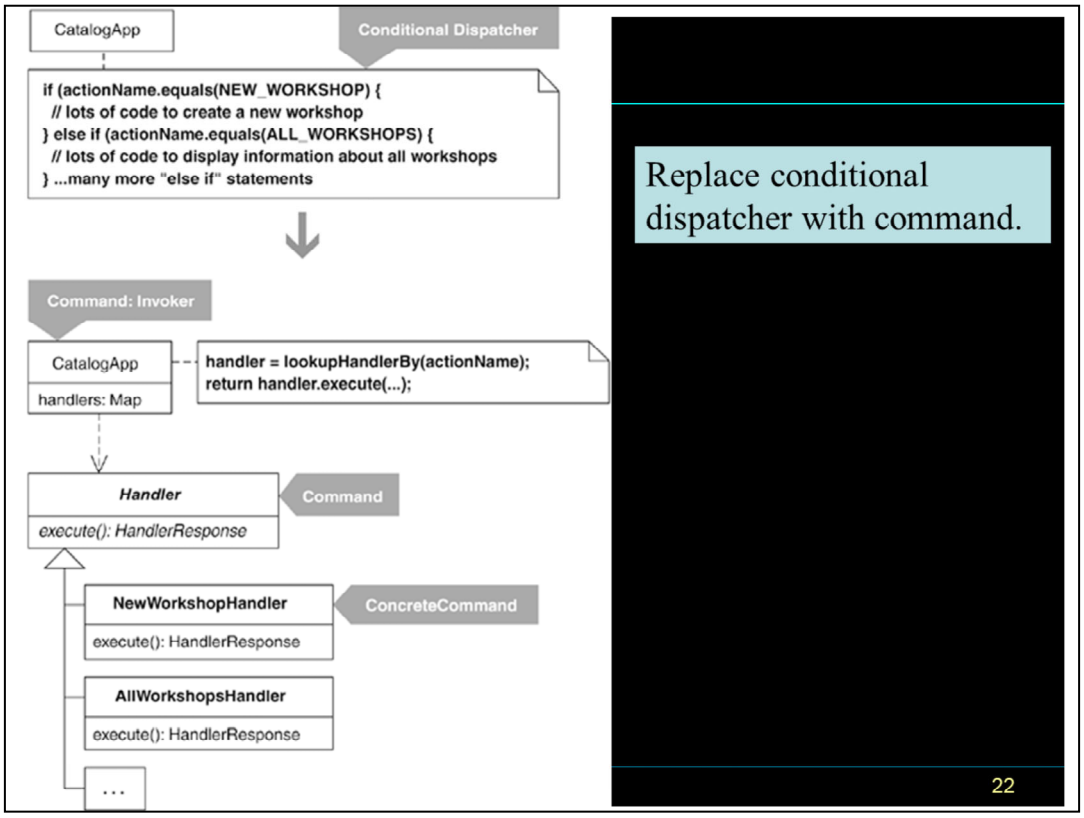
When the menu item is chosen or the button is pressed, the corresponding command is executed.

This helps in separating the user interface code from the logic portion of the code.

A spreadsheet writes all the commands executed using serialization on the hard disk. If there is a system failure,

it loads the commands from the hard disk using deserialization and executes the command again.

Thus we can cover to the last command without saving the entire spreadsheet file again and again.



Create a Command for each action. Store the Commands in a collection and replace the conditional logic with code to fetch and execute Commands.

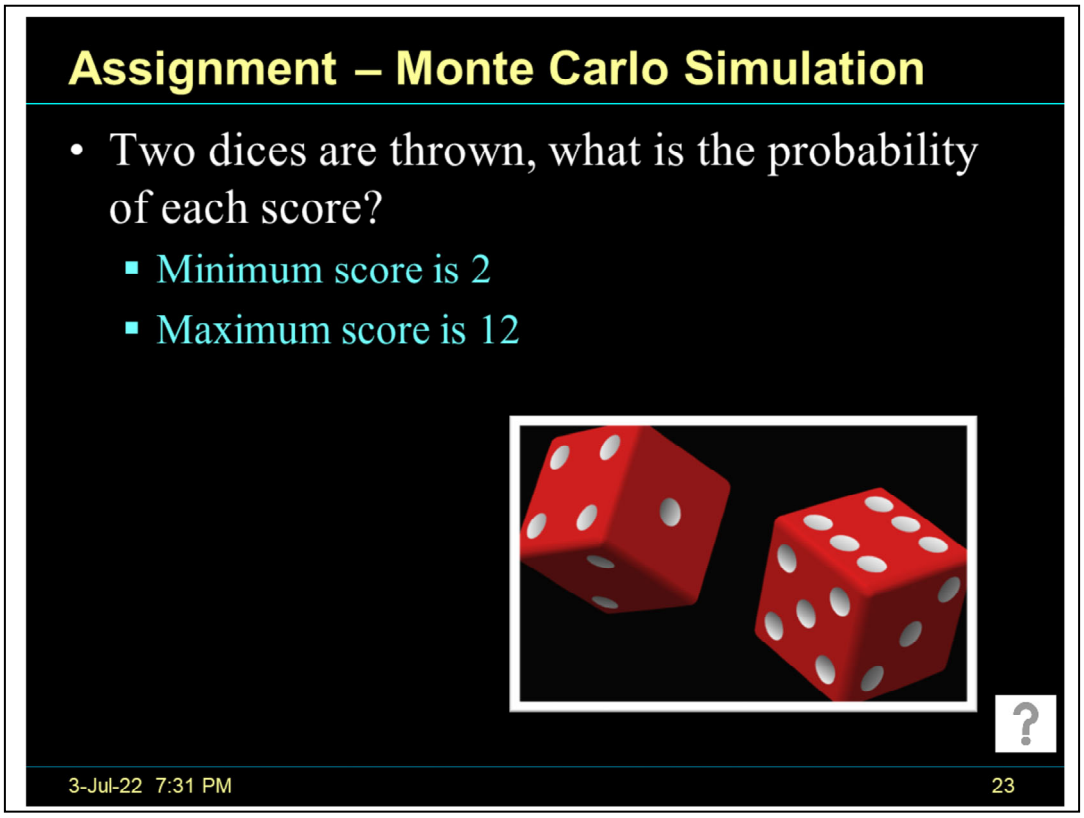
Benefits and Liabilities

+Provides a simple mechanism for executing diverse behavior in a uniform way.

+Enables runtime changes regarding which requests are handled and how.

+Requires trivial code to implement.

–Complicates a design when a conditional dispatcher is sufficient.



Monte Carlo simulations work by running the same simulation many times over with different random

seeds on every run. The results of each run are recorded and aggregated in order

to build up a comprehensive simulation. They have many uses in engineering, finance,

and scientific computing.

Code example with participants