Command Pattern

CS 446

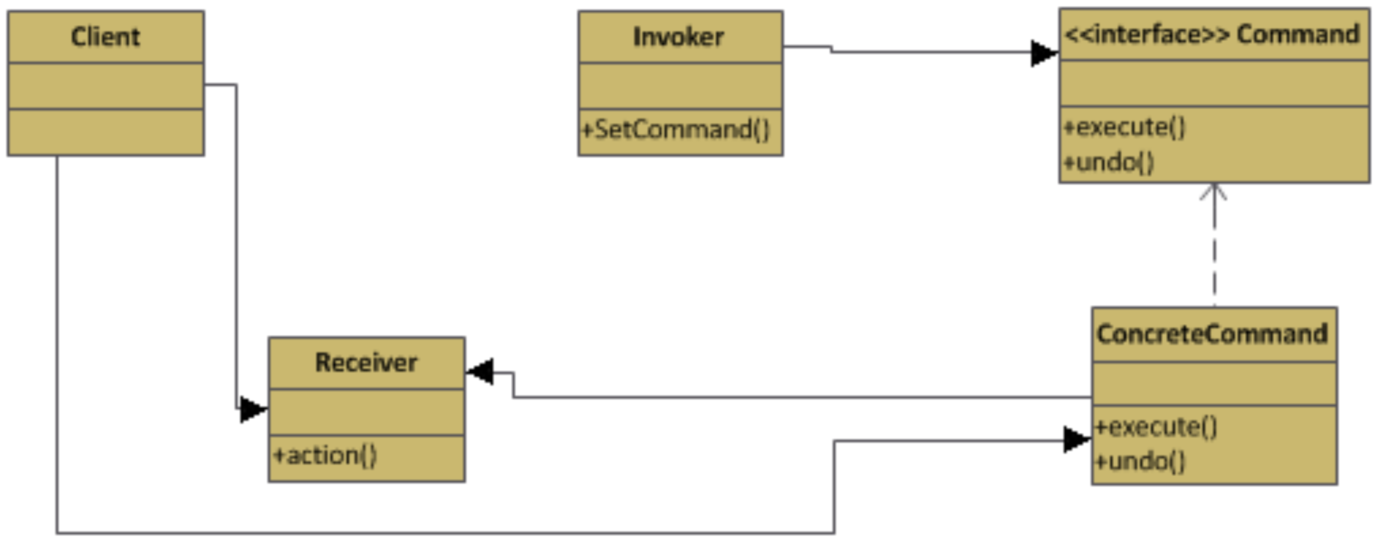
The Command Pattern

- Encapsulates a request as an object
  - Packages requests into a single execute() method
  - Other objects do not know what actions are performed
- Enables parameterizing an object with a command
  - Object can be passed any command which implemen ts execute()
- Support logs and undo operations
- Allows decoupling of the requester of an action from the object performing the action

Command Pattern Components

- Client
  - Creates Command Object
- Command
  - Consists of a set of actions and a receiver
  - Provides one method: execute()
- Invoker
  - Provides setCommand() method called by client
  - Stores command until it is needed
- Receiver
  - Actions invoked by command

Command Pattern Diagram



The Command Interface

```
Public interface Command{
   Public void execute();
}

Public class SwitchOnCommand implements Command{
   Switch switch;

   public LightOnCommand(Switch switch){
         this.switch = switch;
   }

   public void execute(){
         switch.on();
   }

}
```

Using the Command Object

```
Public class RemoteControl {
  Command slot;

  Public RemoteControl() {}

  public void setCommand(Command command){
       slot = command;
  }

  public void buttonPressed(){
       slot.execute();
  }

}
```

Example 1

- Assume GarageDoor class has methods up() and down()

```
Public class RemoteControlTest{
  public static void main(String[] args) {

        RemoteControl remote = new RemoteControl();
        GarageDoor garageDoor = new GarageDoor();
        GarageDoorOpenCommand = new
        GarageDoorOpenCommand(garageDoor);

        remote.setCommand(garageOpen);
        remote.buttonPressed();
  }

}
```

Undo Operations

```
Public class SwitchOnCommand implements Command{
  Switch switch;

  public LightOnCommand(Switch switch){
        this.switch = switch;
  }

  public void execute(){
        switch.on();
  }

  public void undo(){
        switch.off();
  }

}
```

Undo Operations

```
Public class RemoteControl {
   Command onCommand;
   Command offCommand;

   Public RemoteControl() {
          Command noCommand = new NoCommand();

          onCommand = NoCommand;
          offCommand = NoCommand;
          undoCommand = NoCommand;
   }
   public void setCommand(Command on, Command off){
          onCommand = on;
          offCommand = off;
   }
   public void onButtonPressed(){
          onCommand.execute();
          undoCommand = offCommand;
   }
   public void undoButtonPressed(){
          undoCommand.undo();
   }

}
```

Using State to implement undo

```
Public class ceilingFanHighCommand implements Command {
   CeilingFan ceilingFan;
   int prevSpeed;

   public CeilingFanHighCommand(CeilingFan ceilingFan){
          this.ceilingFan = ceilingFan;
   }
   public void execute(){
          prevSpeed = ceilingFan.getSpeed();
          ceilingFan.high();
   }
   public void undo(){
          if(prevSpeed == CeilingFan.HIGH){
                     ceilingFan.high();
          }elseif(prevSpeed == CeilingFan.MEDIUM){
                     ceilingFan.medium();
          } elseif(prevSpeed == CeilingFan.LOW){
                     ceilingFan.low();
          } elseif(prevSpeed == CeilingFan.OFF){
                     ceilingFan.off();
          }
   }
}
```

Macro Commands

```
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(){




   }

}
```

Exercise

- Design a remote control class with 5 on/off button pairs
- Add an "undo" button to support one undo operation
- Assume you already have the following:

```
Public interface Command
{
   Public void execute();
   public void undo();
}


Public Class NoCommand implements Command
{
          public void execute() { }
}
```

Other uses for the Command Pattern

- Queuing
  - Add jobs to a queue
  - Threads remove a command from the queue, call execute() and wait for the call to finish
  - Effective for limiting number of concurrent threads
- Logging requests
  - Add store() and load() methods to command interface
  - Store all commands as they are executed
  - Upon a crash, load all commands since last checkpoint