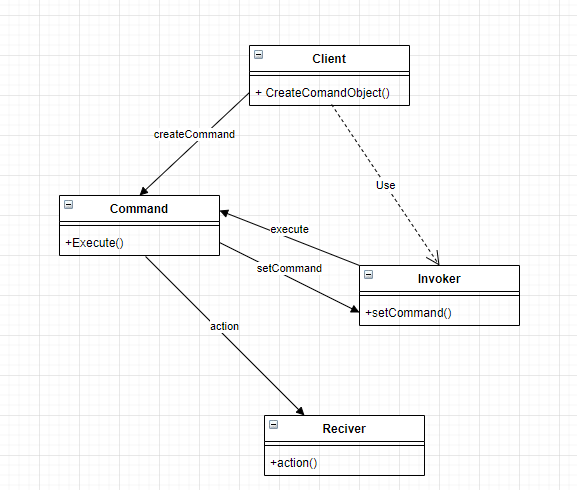
# Workout – 2020-09-21 – Command Pattern

Create a workout directory. Edit this document with answers to questions 1-10 and place it in the workout directory. Then place all your code in that directory. When you are finished with the workout, zip up the workout directory and submit the zip file for this assignment.

1. Match each Diner Example component to the appropriate Command Pattern concept (Command, execute(), Client, Invoker, Receiver, setCommand()):
   * Waitress Invoker
   * Short Order Cook Reciver
   * orderUp() execute()
   * Order Command
   * Customer Client
   * takeOrder() setCommand()
2. Draw the UML for the generic Command Pattern. Export the result as a graphic and paste it into your document.



1. What does the client need to know about specific command objects?

Everything about what it is not much about how its made

1. What does the Invoker need to know about command objects?

Only its type its just stores it until its needed

1. Does the answer to the last two questions exemplify “loose” coupling? Explain.

Yes, they need to know the bare minimum to handle it and nothing more

1. What is the NoCommand object? Why is it often useful?

It is a void command its useful as a default command so you only see the slots that have been given commands

1. In the context of the Command Pattern, define the following terms:
   * Invoker – where commands are stored until needed
   * Receiver – where actions are invoked
   * Command – the object being passed has an execute function to be called later
   * Execute – invokes the action handled by reciver
   * Client – creates the command object
2. Match each Home Automation Example component to the appropriate Command Pattern concept (Command, execute(), Client, Invoker, Receiver, setCommand()):
   * Remote Control - invoker
   * Light- reciver
   * LightOnCommand - command
   * RemoteLoader- client
3. In relation to the Design Principle, "Encapsulate what varies.", what is being encapsulated in the Command Pattern?

commands

1. How does the Design Principle, "Program to interfaces, not implementations.", apply in the Command Pattern?

Execute() comes from the command interface and extends to all commands

1. The source code for the Remote Control project in the textbook is available in a zip file in this section of our ASULearn course. If any file you need for the exercises below has a package statement, remove it. Make the following changes to the code.

* Change the Hottub controls so that the user can increment and decrement the temperature.
  + Add a getter to Hottub for temperature. Add code to the Hottub so that the temperature can never go outside the range 80 to110 (make those values constants in HotTub and modify all methods that change the temperature to enforce this rule).
  + When the Hottub is turned on, always make the initial temperature 105.
  + Write two command classes: IncrementHottubTemperature and DecrementHottubTemperature that correspondingly add 1 or subtract 1 to/from the temperature. In addition to changing the hottub, make each command displays a message stating what the command did along with the resulting hottub temperature. These commands should do nothing if the hottub is off.
* Modify the RemoteLoader class adding a 5thwith a pair of commands on/off for the Hottub. Make the 6th pair of commands incrementing and decrementing the Hottub temperature.
* Add the following remoteControl commands to the end of main:
  + Turn on the hottub
  + Increment the hottub temperature 6 times
  + Decrement the hottub temperature
  + Turn off the hottub