

## The Command Pattern defined

You've done your time in the Objectville Diner; you've partly implemented the remote control API, and in the process you've got a fairly good picture of how the classes and objects interact in the Command Pattern. Now we're going to define the Command Pattern and nail down all the details.

Let's start with its official definition:

**The Command Pattern** encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations.

Let's step through this. We know that a command object *encapsulates a request* by binding together a set of actions on a specific receiver. To achieve this, it packages the actions and the receiver into an object that exposes just one method, `execute()`. When called, `execute()` causes the actions to be invoked on the receiver. From the outside, no other objects really know what actions get performed on what receiver; they just know that if they call the `execute()` method, their request will be serviced.

We've also seen a couple examples of *parameterizing an object* with a command. Back at the diner, the Waitress was parameterized with multiple orders throughout the day. In the simple remote control, we first loaded the button slot with a "light on" command and then later replaced it with a "garage door open" command. Like the Waitress, your remote slot didn't care what command object it had, as long as it implemented the Command interface.

What we haven't encountered yet is using commands to implement *queues and logs and support undo operations*. Don't worry, those are pretty straightforward extensions of the basic Command Pattern, and we'll get to them soon. We can also easily support what's known as the Meta Command Pattern once we have the basics in place. The Meta Command Pattern allows you to create macros of commands so that you can execute multiple commands at once.

An encapsulated request.

