

# **18. The system is controlled by complex combinations of inputs**

Some internal components of your system need to accept requests with instructions to do some actions.

Over time, the structure of the requests is becoming more and more complicated. For example, there may be different objects in your system that end up executing a very similar action, while having a completely different set of inputs.

This may start causing issues in your code by making it less readable. For example, if different object types end up executing the same action, you may put this action into a base class that all of these objects inherit from. But what if those objects are totally different, so having them inherit from the same base type wouldn't be appropriate? Also, if you have a large variety of such objects, you may end up with a lot of base classes.

So, your goal is to standardize your requests.

## **Suitable design patterns**

### **Command**

Command is a design pattern where the request is already embedded in the class that executes a particular action. This class is known as a Command and it only contains a single method, which is usually called Execute.

So, none of the objects will be responsible for generating the request. Once you create a Command, it can be executed from any object. This leaves the Command object with the sole responsibility of executing the action. Otherwise, all the calling objects need to do is call its `Execute` method.

Usually, the `Execute` method comes with no parameters (perhaps, except for a cancellation token). But you can still populate the Command with some unique values by passing those into its constructor when the Command is created.

### **Why would you use Command**

- Separating the request from the objects and making them, which enforces the single responsibility principle.
- Standardized instructions across your application.
- Ability to implement undo/redo actions.
- Ability to assemble simple instructions into complex ones.
- Easy to schedule instructions.