Meanwhile, back at the Diner..., or, A brief introduction to the Command Pattern

As Joe said, it is a little hard to understand the Command Pattern by just hearing its description. But don't fear, we have some friends ready to help: remember our friendly diner from Chapter 1? It's been a while since we visited Alice, Flo, and the short-order cook, but we've got good reason for returning (beyond the food and great conversation): the diner is going to help us understand the Command Pattern.

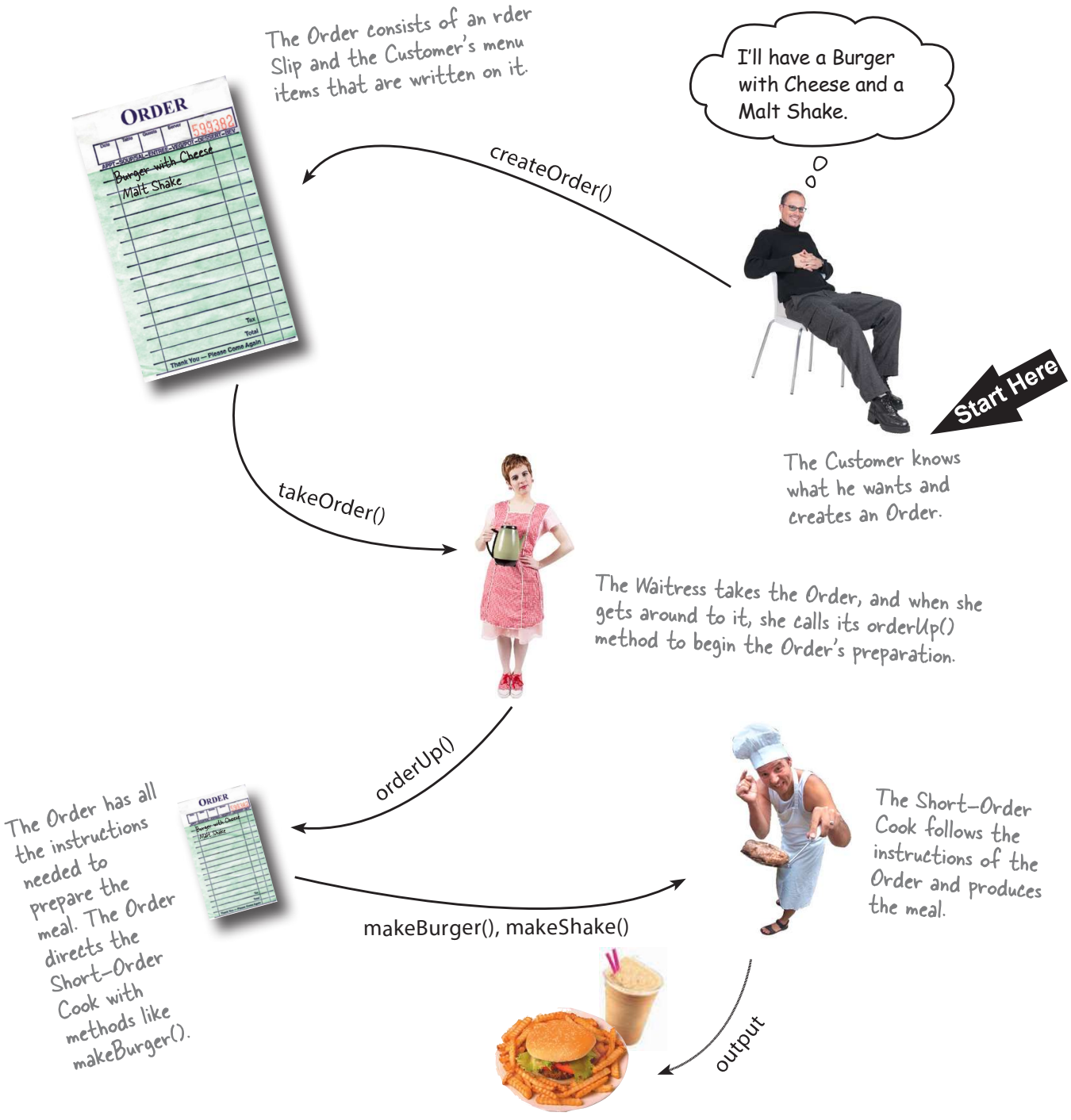
So, let's take a short detour back to the diner and study the interactions between the customers, the waitress, the orders, and the short-order cook. Through these interactions, you're going to understand the objects involved in the Command Pattern and also get a feel for how the decoupling works. After that, we're going to knock out that remote control API.

Checking in at the Objectville Diner...

Okay, we all know how the Diner operates:

Let's study the interaction in a little more detail...

...and given this Diner is in Objectville, let's think about the object and method calls involved, too!



The Objectville Diner roles and responsibilities

An Order Slip encapsulates a request to prepare a meal.

Think of the Order Slip as an object that acts as a request to prepare a meal. Like any object, it can be passed around—from the Waitress to the order counter, or to the next Waitress taking over her shift. It has an interface that consists of only one method, orderUp(), that encapsulates the actions needed to prepare the meal. It also has a reference to the object that needs to prepare it (in our case, the Short-Order Cook). It's encapsulated in that the Waitress doesn't have to know what's in the Order or even who prepares the meal; she only needs to pass the slip through the order window and call "Order up!"

Okay, in real life a waitress would probably care what is on the order slip and who cooks it, but this is Objectville...work with us here!

The Waitress's job is to take Order Slips and invoke the orderUp() method on them.

The Waitress has it easy: take an Order from the Customer, continue helping customers until she makes it back to the order counter, and then invoke the orderUp() method to have the meal prepared. As we've already discussed, in Objectville, the Waitress really isn't worried about what's on the Order or who is going to prepare it; she just knows Order Slips have an orderUp() method she can call to get the job done.

Now, throughout the day, the Waitress's takeOrder() method gets parameterized with different Order Slips from different customers, but that doesn't faze her; she knows all Order Slips support the orderUp() method and she can call orderUp() any time she needs a meal prepared.

The Short-Order Cook has the knowledge required to prepare the meal.

The Short-Order Cook is the object that really knows how to prepare meals. Once the Waitress has invoked the orderUp() method; the Short-Order Cook takes over and implements all the methods that are needed to create meals. Notice the Waitress and the Cook are totally decoupled: the Waitress has Order Slips that encapsulate the details of the meal; she just calls a method on each Order to get it prepared. Likewise, the Cook gets his instructions from the Order Slip; he never needs to directly communicate with the Waitress.