

Implementing more states

Now that you're starting to get a feel for how the Gumball Machine and the states fit together, let's implement the HasQuarterState and the SoldState classes...

```
public class HasQuarterState implements State {
    GumballMachine gumballMachine;

    public HasQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("You can't insert another quarter");
    }

    public void ejectQuarter() {
        System.out.println("Quarter returned");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

    public void turnCrank() {
        System.out.println("You turned...");
        gumballMachine.setState(gumballMachine.getSoldState());
    }

    public void dispense() {
        System.out.println("No gumball dispensed");
    }
}
```

When the state is instantiated we pass it a reference to the GumballMachine. This is used to transition the machine to a different state.

An inappropriate action for this state.

Return the customer's quarter and transition back to the NoQuarterState.

When the crank is turned we transition the machine to the SoldState state by calling its setState() method and passing it the SoldState object. The SoldState object is retrieved by the getSoldState() getter method (there is one of these getter methods for each state).

Another inappropriate action for this state.

Now, let's check out the SoldState class...

```
public class SoldState implements State {
    //constructor and instance variables here

    public void insertQuarter() {
        System.out.println("Please wait, we're already giving you a gumball");
    }

    public void ejectQuarter() {
        System.out.println("Sorry, you already turned the crank");
    }

    public void turnCrank() {
        System.out.println("Turning twice doesn't get you another gumball!");
    }

    public void dispense() {
        gumballMachine.releaseBall();
        if (gumballMachine.getCount() > 0) {
            gumballMachine.setState(gumballMachine.getNoQuarterState());
        } else {
            System.out.println("Oops, out of gumballs!");
            gumballMachine.setState(gumballMachine.getSoldOutState());
        }
    }
}
```

Here are all the inappropriate actions for this state

And here's where the real work begins...

We're in the SoldState, which means the customer paid. So, we first need to ask the machine to release a gumball.

Then we ask the machine what the gumball count is, and either transition to the NoQuarterState or the SoldOutState.



Look back at the GumballMachine implementation. If the crank is turned and not successful (say the customer didn't insert a quarter first), we call dispense anyway, even though it's unnecessary. How might you fix this?