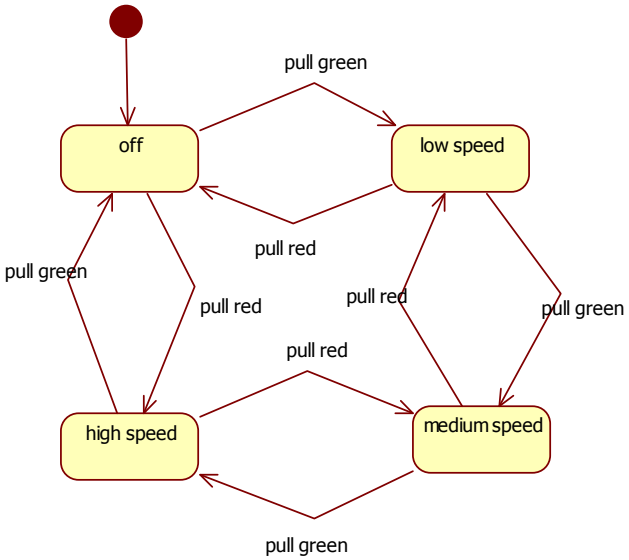
State pattern Lab

We have to write the software that goes into a fancy ceiling fan. This ceiling fan has 3 speeds, with 2 pull chain cords. A red pull chain cord and a green pull chain cord
Here is the state diagram that shows the behavior we have to implement in the ceiling fan.



The ceiling fan always starts in the Off state.

Given is the following code:

```
public class Application {

	public static void main(String[] args) {
		CeilingFan fan = new CeilingFan();
		fan.pullgreen();
		fan.pullgreen();
		fan.pullgreen();
		fan.pullgreen();
		fan.pullred();
		fan.pullred();
	}
}
public class CeilingFan {
	int current_state=0;
	public void pullgreen() {
		if (current_state == 0) {
		   current_state = 1;
		   System.out.println( "low speed" );
		} else if (current_state == 1) {
		   current_state = 2;
		   System.out.println( "medium speed" );
		} else if (current_state == 2) {
		   current_state = 3;
		   System.out.println( "high speed" );
		} else {
		   current_state = 0;
		   System.out.println( "turning off" );
		}
	}

	public void pullred() {
		if (current_state == 0) {
		   current_state = 3;
		   System.out.println( "high speed" );
		} else if (current_state == 1) {
		   current_state = 0;
		   System.out.println( "turning off" );
		} else if (current_state == 2) {
		   current_state = 1;
		   System.out.println( "low speed" );
		} else {
		   current_state = 2;
		   System.out.println( "medium speed" );
		}
	}
}
```

If you run the Application, you get the following output:

```
low speed
medium speed
high speed
turning off
high speed
medium speed
```

The disadvantages of this code are:

1. The long if-else structure is not very nice to program, and you make mistakes easily if you have to change it.
2. When we add a new state, we have to change both methods in the CeilingFan class.

We learned that the State pattern can improve our application when we have different behavior in different states.

Rewrite the Java code, but now with the State pattern applied to the code.

The requirements for your solution are:

1. The new CeilingFan class should be independent of the states (also for the start state). If I add a new state, then I don't need to modify the CeilingFan class.
2. The different states themselves are responsible for creating and setting the next state. This means that the business rule deciding the next state, is implemented in the states themselves. This also means that the Application class only knows about the start state.
3. The output of your modified application should be exactly the same as the given output of the given application.
4. There should be no if-then-else structure in your code for the ceiling fan.

Todo

a. Draw the class diagram of your design.
b. Draw a sequence diagram that shows how your new design works. On the sequence diagram you should show how the state pattern works.
c. Modify the given code such that you meet the given requirements above.