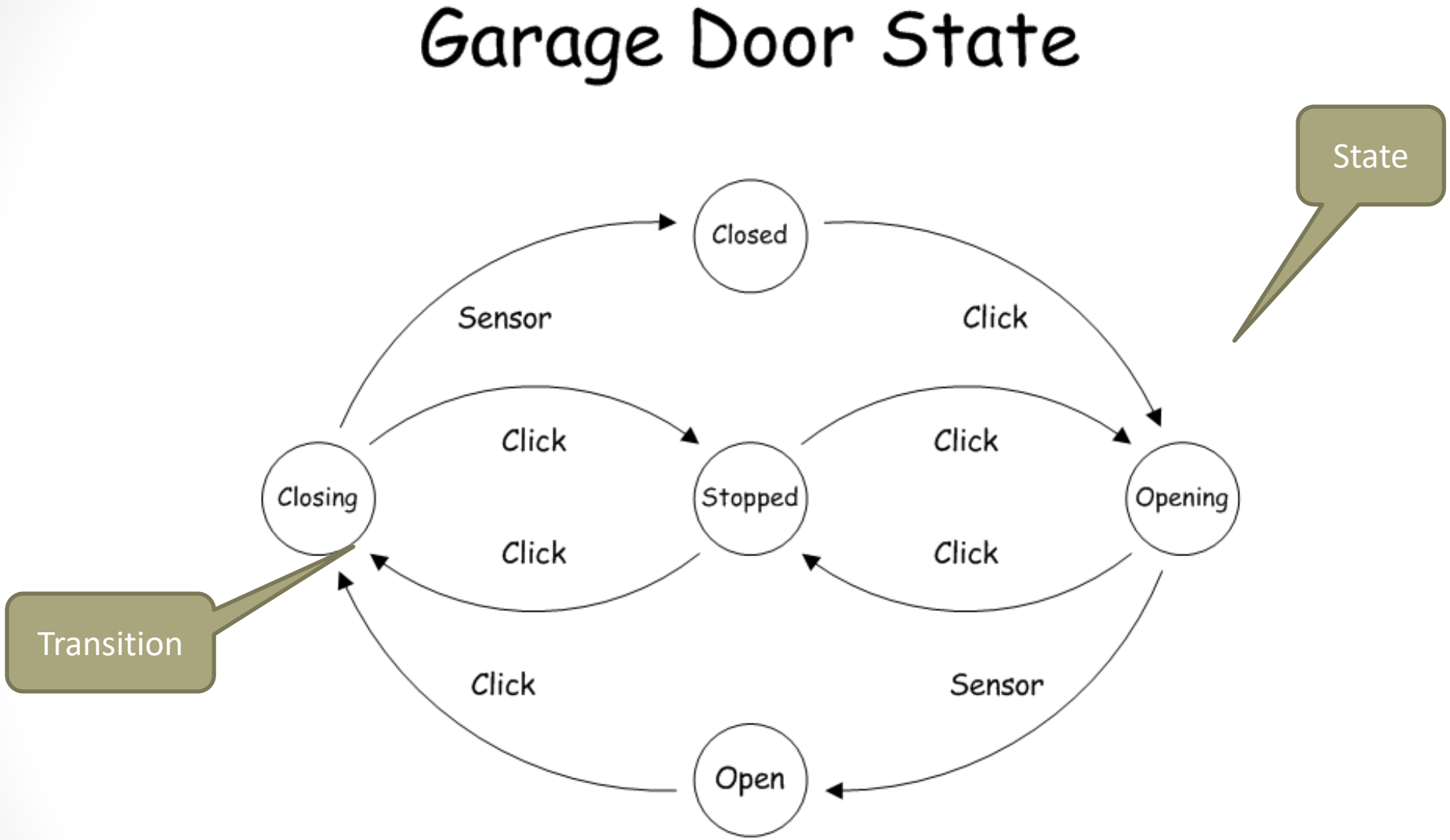
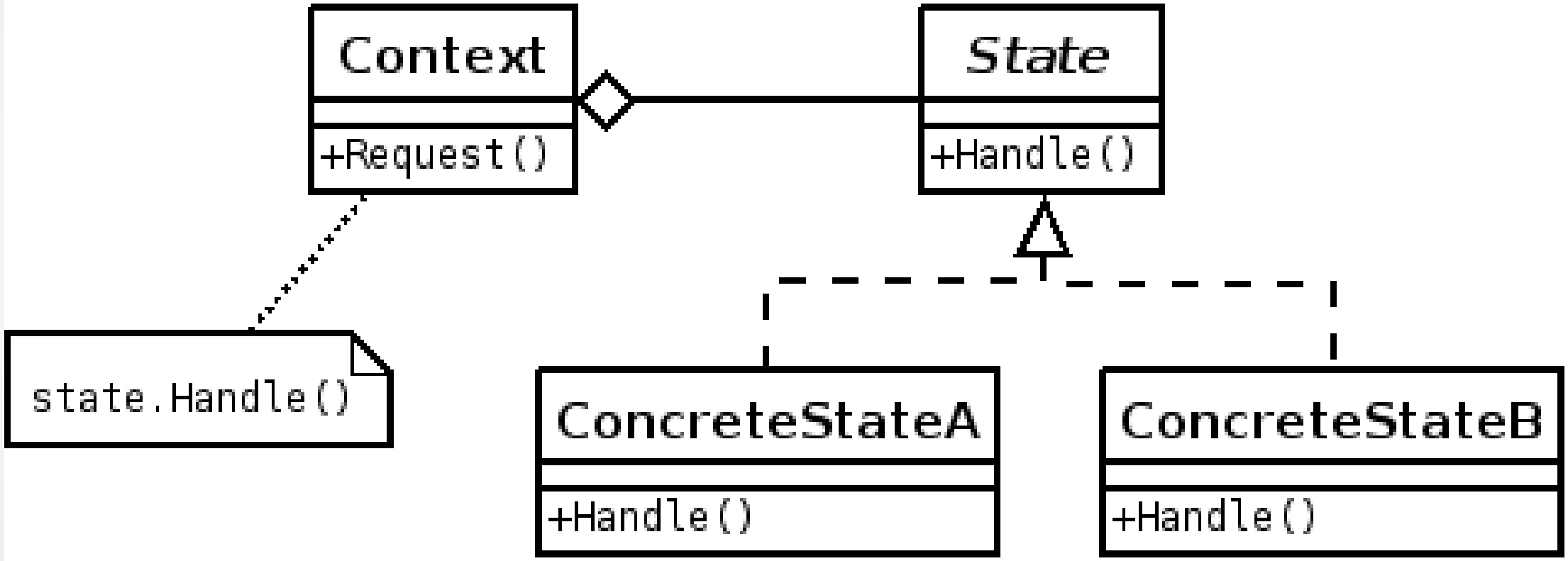
State Design Pattern

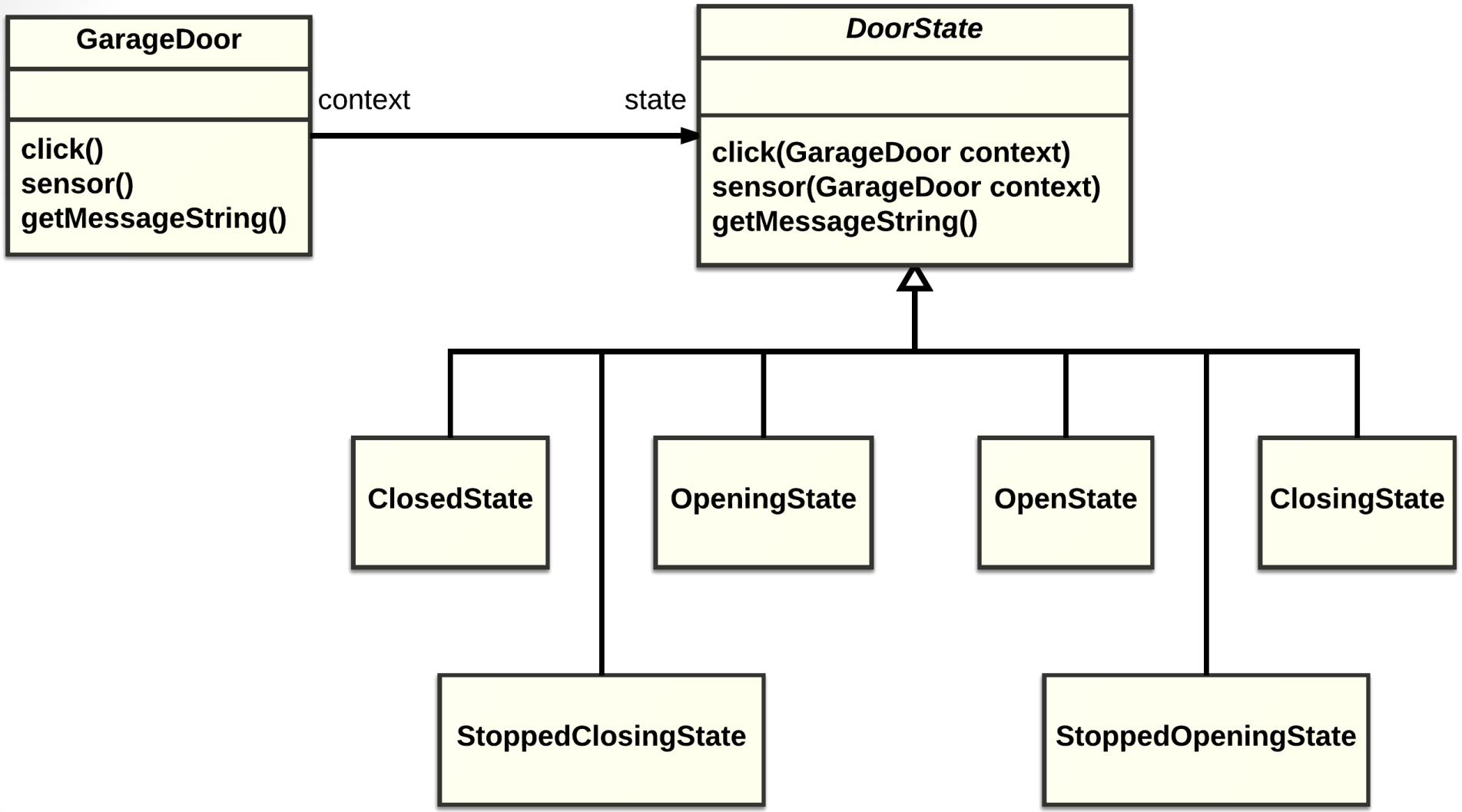
By Mike Rieser



State Chart Diagram



State Design Pattern



Garage Door Class Diagram

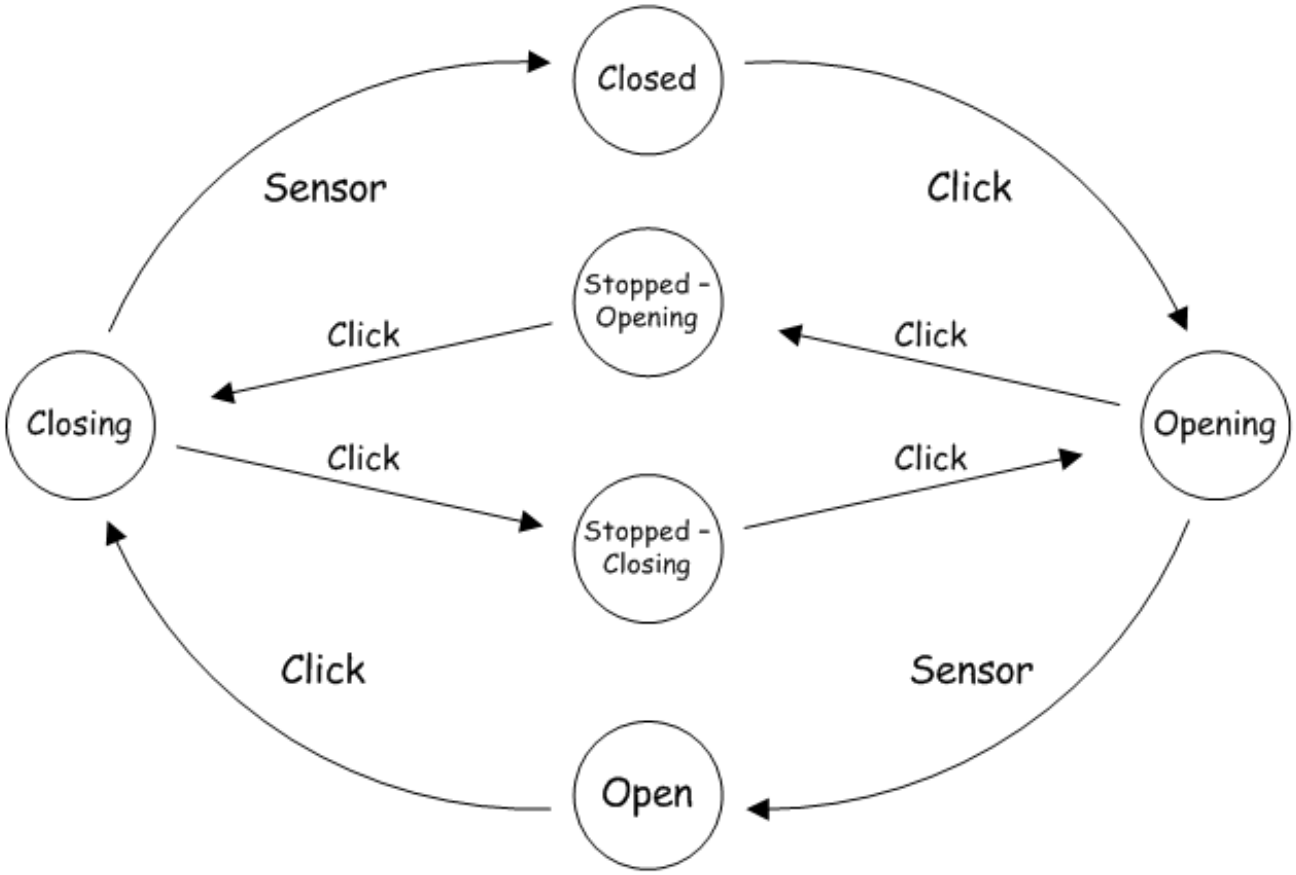
Garage Door – Java

```
public class GarageDoor {

   private DoorState state = new ClosedState();

   void setState(DoorState state) {
      this.state = state;
   }

   public void click() {
      state.click(this);
   }

   public String getMessageString() {
      return state.getMessageString();
   }

   public void sensor() {
      state.sensor(this);
   }
}
```

```
abstract class DoorState {
   abstract void click(GarageDoor door);
   abstract void sensor(GarageDoor door);
   abstract String getMessageString();
}
```

```
class ClosedState extends DoorState {
   @Override
   void click(GarageDoor context) {
      context.setState(new OpeningState());
   }

   @Override
   String getMessageString() {
      return "Closed";
   }
}
```

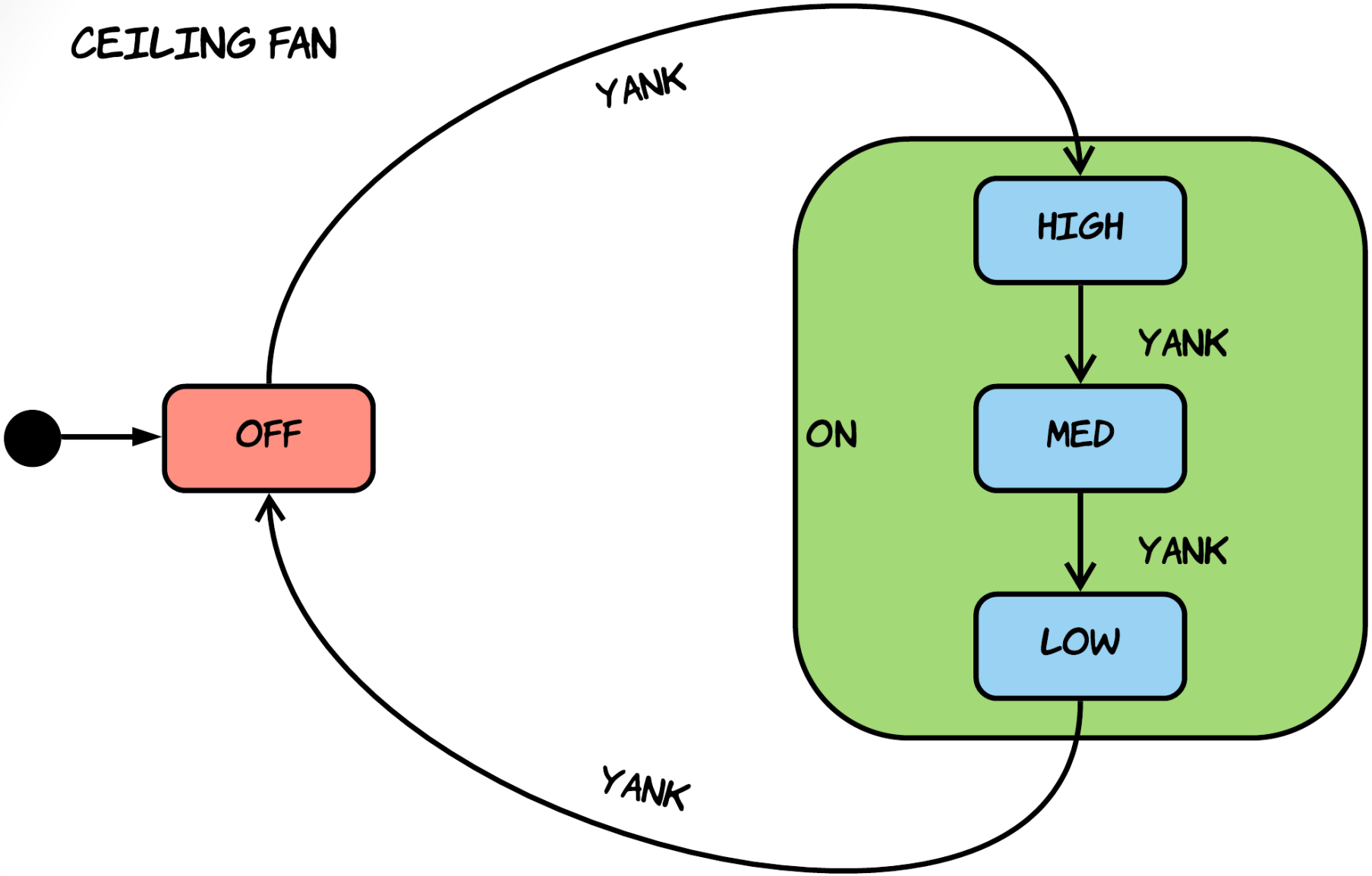
Garage Door State



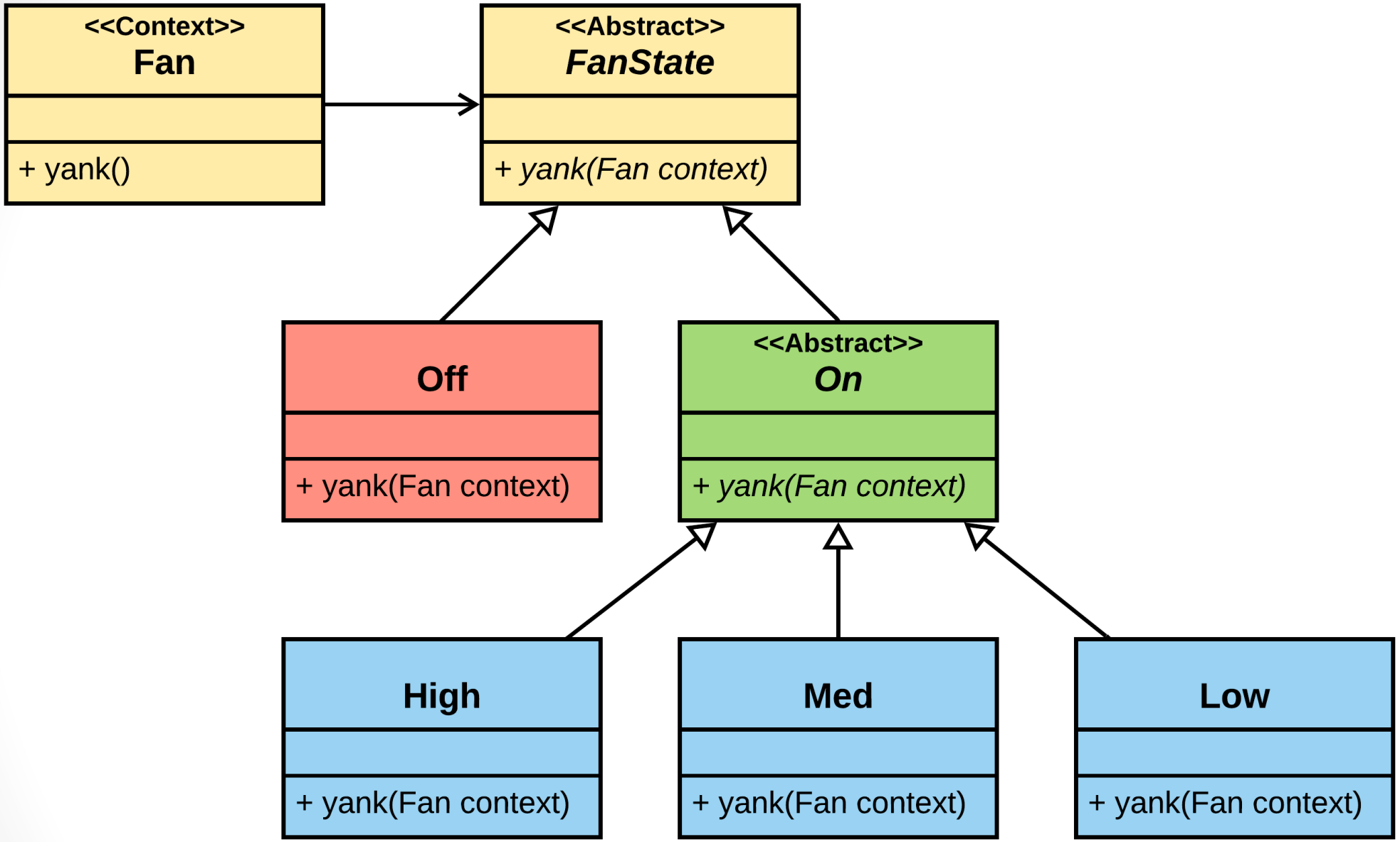
Added new Stopped states

Refactoring to State Pattern

1. Create an abstract State super class.
2. Copy the public method signatures from the context to the abstract State.
3. Mark them as abstract in the State class, additionally pass the context as an argument.
4. In the context, create an instance variable to hold the state.
5. Initialize the instance variable to the start state.
6. Add a setState(State) to the context.
7. Find the location to **Sprout Class**, add an "// old code" comment.
8. Add the sprout and move the code to the State subclass.
9. Add each transition in each State.
10. Make sure the state has the correct code for the other public methods.
11. Try switching the public methods from the context to the states.



Ceiling Fan State Chart



Ceiling Fan Class Design