Objectives

- Exercise with Git, Github, and the State Pattern
  - Making a pull request
  - Implementing against unit tests

Github

- Previously, I asked that you create a github profile
- We have a repository[1] set up for today's exercise
  - This is a maven project, as we have used previously
  - You need to *fork* this repository, *clone* your fork, and import the clone as a maven project to your workspace
- Complete the state pattern implementation so that all unit tests pass then submit your work as a *pull request*
  - The github documentation explains, in detail, how to submit a pull request[2]

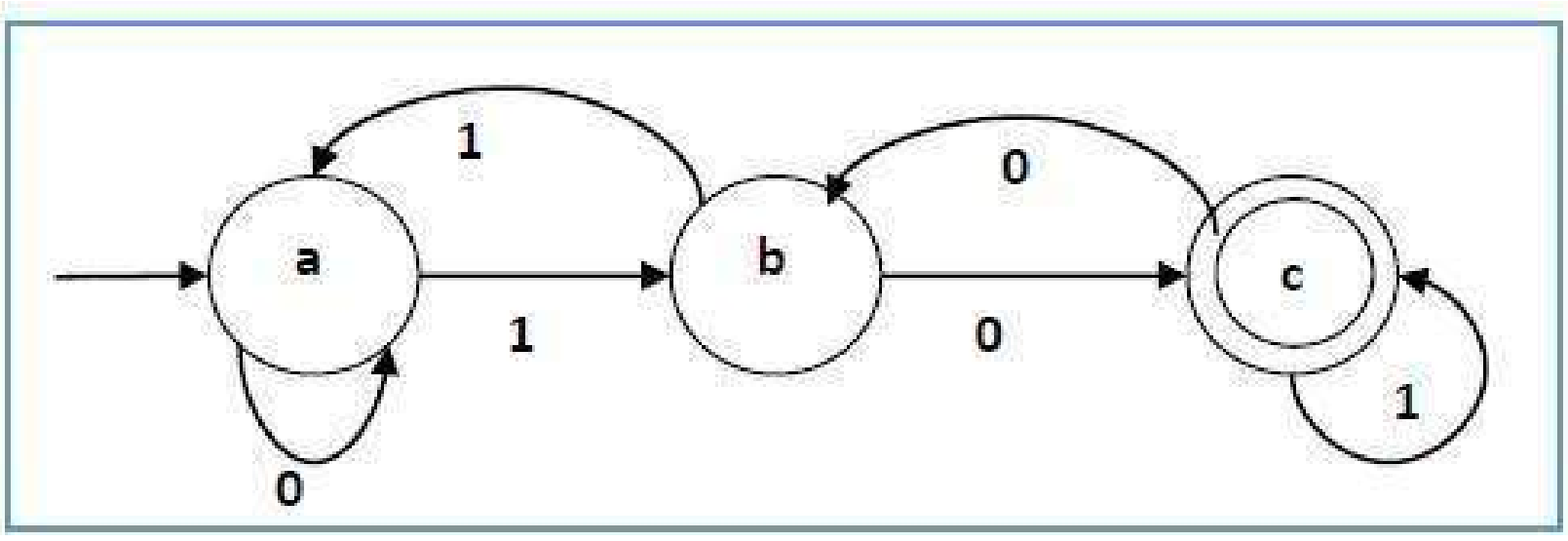
[1] `https://github.com/marks1024/exercise-statepattern-361`

[2] `https://help.github.com/articles/creating-a-pull-request-from-a-fork/`

Github and Email

- Github associates your commits with email addresses
- You can configure your git email address with the following commands
  - Set for the current repository: `git config user.email "your email"`
  - Set email for all repositories: `git config --global user.email "your email"`
  - Show the current value of your git email: `git config user.email`
- Github also provides an anonymous email that you can use, so that your public commits won't show your email address: `<git username>@users.noreply.github.com`

State Machines and DFAs



- Example of a basic state machine used in computer science: basic model of computation, matching patterns
- Two types of states: accept and reject (double-circle)
- Machine consumes a character and follows the appropriate edge to change state
- The patterns that can be recognized with a DFA are equivalent to the patterns we can specify with (strict) regex syntax

Other uses of State Machines

- State Machines can also be used to model simple mechanistic systems such as vending machines and turnstiles
- Vending machine actions are insert quarters, selecting items and states are change entered, number of items remaining etc.

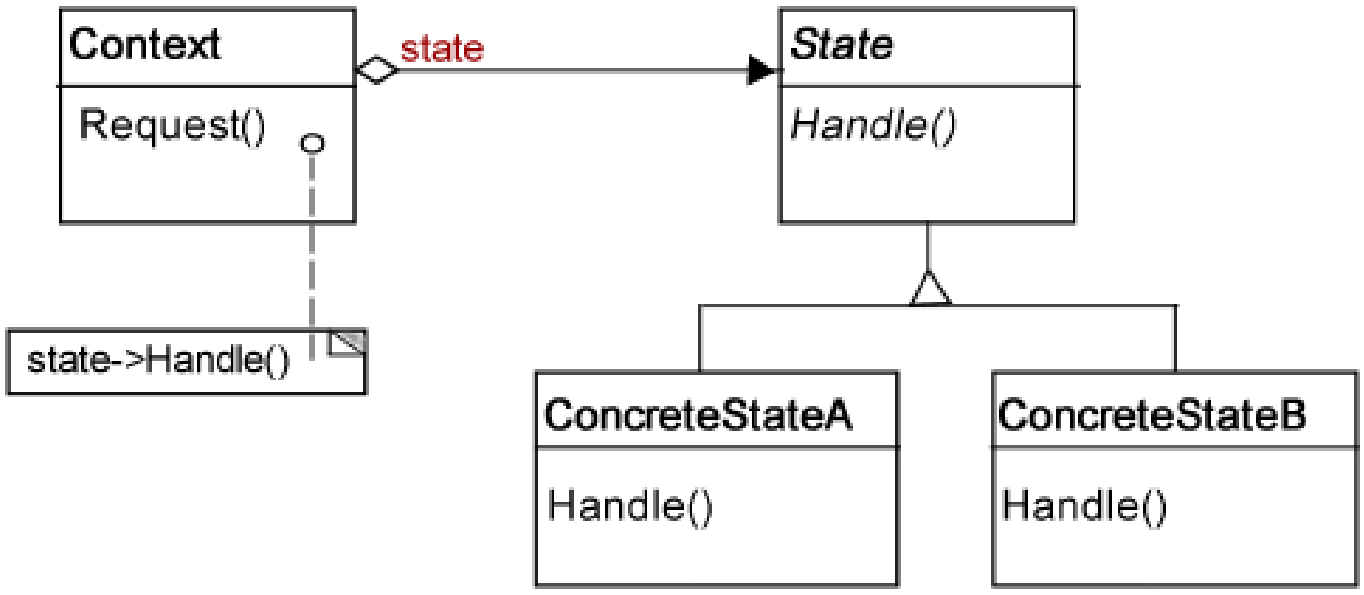
The State Pattern

- Create a state machine in an object oriented fashion
- Classes Involved:
  - Context
  - State Interface
  - Concrete State Classes

State

"Allow an object to alter its behavior when its internal state changes. The object will appear to change its class." *GoF*

State Pattern Class Diagram



- The Context delegates its public methods to a state object that it aggregates
- State is an abstraction
- Concrete implementations of State define the behavior of the context

The Context Class


```
public class StateContext {
    final State state1 = new State1(this);
    final State state2 = new State2(this);
    final State state3 = new State3(this);
    private State currentState;

    public StateContext() {
        this.currentState = state1;
    }

    public void actionA() {
        // complete this method by
        // delegation to the current state
    }
    public void actionB() {
        // complete this method
        // delegate to the current state
    }
    public boolean inAcceptState() {
        // complete this method and return correct value
        // delegate to the current state
        return false;
    }

    public State getCurrentState() {return currentState;}

    public void setCurrentState(State currentState) {
        this.currentState = currentState;
    }
}
```

- The context for our example

The State Abstraction


```
public abstract class State {
    protected StateContext sc;
    protected boolean accept = false;

    public void actionA() {
    }

    public void actionB() {
    }

    public boolean isAccept() {
        return this.accept;
    }
}
```

- Concrete States inherit from this abstract class

Using the State Machine

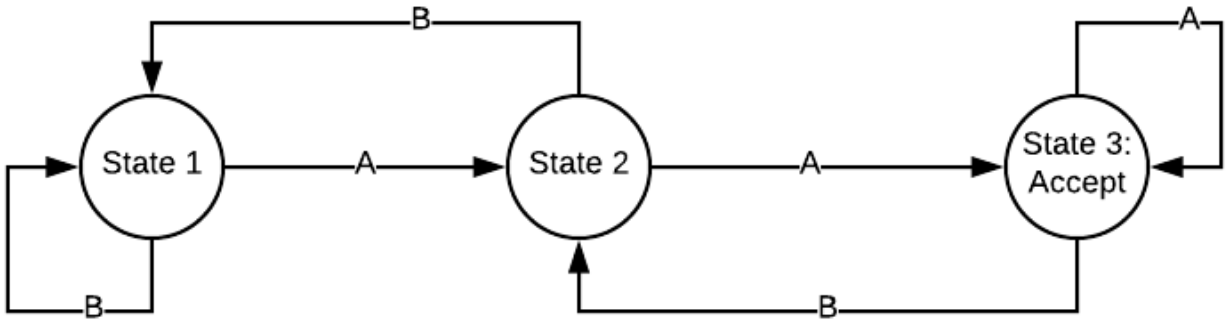

```
public class StateTest {
    public static void main(String[] args) {
        StateContext sc = new StateContext();

        sc.actionA();
        sc.actionA();
        sc.actionB();

        // Is machine in an accept state
        // after receiving AAB
        sc.inAcceptState();
    }

}
```

- Example of the behavior we would like from our class
- Sequence of actions executed on the context will make it appear to change its behavior from the outside
- Internal state evolves in pre-defined way

State Pattern Summary

- State Pattern allows an object to have an internal state that changes its behavior
- Each state is represented by a class (will increase the number of classes in your design)
- The class diagrams for State and Strategy are the same
  - *Strategy*: alternative to subclassing
  - *State*: prevent a lot of conditional statements from appearing in your main class

Task: State Pattern Implementation



- The machine should begin in `State1` and only `State3` should be an accept state
- Must provide implementations for the three concrete state classes
- All of the unit tests (6 of them) in the project should pass
- Submit your solution as a pull request
- **due by the end of the day**