Module 20: "State"





Agenda

- ▶ Introductory Example: Setting a Timer
- Challenges
- ▶ Implementing the State Pattern
- Pattern: State
- Overview of State Pattern



Introductory Example: Setting a Timer

```
switch (_state)
   case StateKind.Normal:
        return (ConsoleColor.Gray, DateTime.Now.ToShortTimeString());
    case StateKind.SetHours:
        return (ConsoleColor.Red, $"{ timerHours:00}");
    case StateKind.SetMinutes:
        return (ConsoleColor.Red, $"{_timerHours:00}:{_timerMinutes:00}");
    case StateKind.Completed:
        return (ConsoleColor.Green, timerSet?.ToShortTimeString());
   default:
        throw new NotImplementedException($"State {_state} not expected");
```



Timer Setup Display

```
Timer 21:24 << >> OK
Timer 21 << >> OK
Timer 23:00 << >> OK
Timer 23:57 << >> OK
Timer 21:24 << >> OK
```



Challenges

- Extensibility problem when adding more states
- Highly repetitive code
- Multiple responsibilities mixed
- Almost impossible to unit test



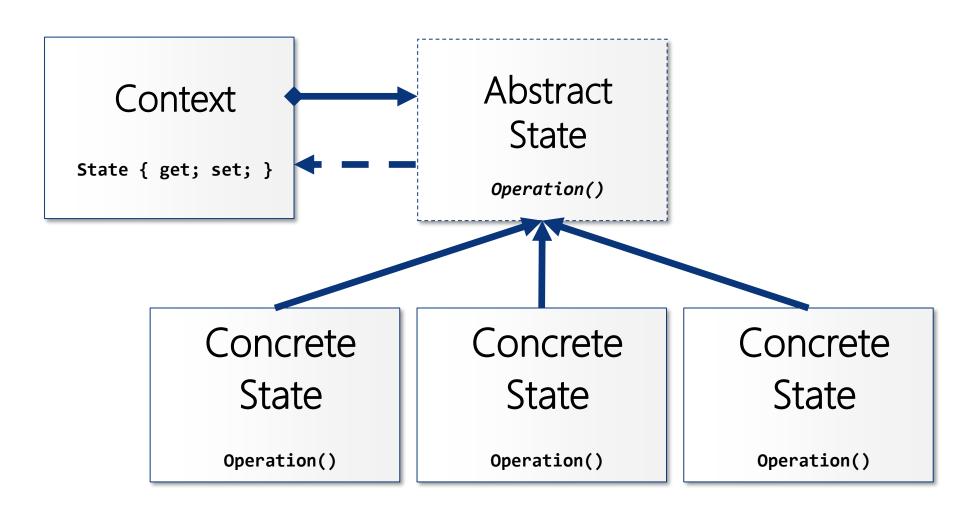
Pattern: State

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

- Outline
 - Encapsulate logic of distinct states into separate classes
 - Owner class will act a as proxy to state objects
 - Make program maintainable (and testable!)
- Origin: Gang of Four



Overview of State Pattern





Overview of State Pattern

Context

- Main class accepting requests
- Has no state-specific behavior
- Refers to the Abstract State interface (or abstract base class)

Abstract State

- Interface or abstract class defining state behavior interface
- Might contain common state functionality or helpers, including State property or method

Concrete States

 Each concrete state class contains state-specific behavior relating to the particular individual state



