State Pattern

Lecture Question

- Simulate a TV without using control flow (ie. Use the state pattern)
- In a package named tv, create a Class named TV with no constructor parameters
- The TV must contain the following methods as its API:
    - volumeUp(): Unit
    - volumeDown(): Unit
    - mute(): Unit
    - power(): Unit
    - currentVolume(): Int
- In the tv package, write a test suite named TestTV that will test all the functionality on the spec sheet
    - Note: Only call the API methods while testing. Other methods/variable you create will not exist in the grader submissions

20 points

Due: Saturday/tomorrow night @11:59pm

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
    - Volume up/down and mute buttons do nothing
    - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button mutes the TV
- When the TV is muted:
    - Current volume is 0
    - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was turned off while muted, when it is turned back on it should not be muted

Lab Lecture

- Lecture will be very short today
- Use this time to start/complete your lecture question

State Pattern - Closing Thoughts

State pattern trade-offs

- **Pros**
  - Organizes code when a single class can have very different behavior in different circumstances
  - Each implemented method is only concerned with the reaction to 1 event (API call) in 1 state
  - Easy to change or add new behavior after the state pattern is setup
- **Cons**
  - Can add complexity if there are only a few states or if behavior does not change significantly across states
  - Spreading the behavior for 1 class across many classes can look complex and require clicking through many files to understand all the behavior

State Pattern - Closing Thoughts

- Do not use the state pattern everywhere
  - Decide if a class is complex enough to benefit from this pattern before applying it
- The state pattern in this class
  - I have to force you to use it by removing control flow (Not realistic)
  - Used to reinforce your understanding of **inheritance** and **polymorphism**
  - Used as an example of a design pattern that can help organize your code
- When you're not forced to use this pattern
  - Weight the pros and cons to decide when it is the best approach

Lecture Question

- Simulate a TV without using control flow (ie. Use the state pattern)
- In a package named tv, create a Class named TV with no constructor parameters
- The TV must contain the following methods as its API:
    - volumeUp(): Unit
    - volumeDown(): Unit
    - mute(): Unit
    - power(): Unit
    - currentVolume(): Int
- In the tv package, write a test suite named TestTV that will test all the functionality on the spec sheet
    - Note: Only call the API methods while testing. Other methods/variable you create will not exist in the grader submissions

20 points

Due: Saturday/tomorrow night @11:59pm

TV Spec Sheet

- TV is initially off when created
- Initial volume is 5
- When the TV is off:
    - Volume up/down and mute buttons do nothing
    - Current volume is 0
- The power button turns the TV on/off
- Volume up button increases volume by 1 up to a maximum volume of 10
- Volume down button decreases volume by 1 down to minimum volume of 0
- Pressing the mute button mutes the TV
- When the TV is muted:
    - Current volume is 0
    - Pressing the mute, volume up, or volume down buttons will unmute the TV and restore the volume to the pre-mute volume (Do not in/decrease the volume)
- When turning the TV back on, the volume should return to its value when the TV was last on
- If the TV was turned off while muted, when it is turned back on it should not be muted

Lecture Question

Hints/Suggestions

- You can use Math.min and Math.max to prevent the volume from being negative or greater than 10 without using a conditional
- You can store the volume in a variable in the TV class so it will persist as the state changes (ex. Turning the TV off will not change this variable, the off state just won't access the variable and will return 0 instead)