State Pattern

Lecture Question

- Simulate a Car without using control flow (ie. Use the state pattern)
- In a package named oop.car, create a Class named Car with no constructor parameters
- The Car must contain the following methods as its API:
  - shiftToDrive(): Unit
  - shiftToPark(): Unit
  - shiftToReverse(): Unit
  - accelerate(): Unit
  - brake(): Unit
  - velocity(): Int
- In the tests package, write a test suite named TestCar that will test all the functionality on the spec sheet
  - Note: Only call the API methods while testing. Other classes/methods/ variables you create will not exist in the grader submissions

Car Spec Sheet

- Car is initially in Park
- Initial velocity is 0
- When the Car is in Park:
  - Accelerating and braking have no effect
  - The car can shift into drive or reverse
- When the car is in Drive:
  - Calling accelerate will **increase** the velocity of the car by 10
  - Calling brake will completely stop the car (velocity of 0)
- When the car is in Reverse:
  - Calling accelerate will **decrease** the velocity of the car by 5 (negative velocity)
  - Calling brake will completely stop the car (velocity of 0)
- If the car shift gears while moving (velocity != 0):
- If the car shift gears from reverse to drive or drive to reverse while moving (velocity != 0):
  - The car come to a stop and breaks. No methods have any functionality after this
- If the car shifts to park while moving:
  - The car stops and is damaged
  - If a the car is damaged a second time it will break and have no functionality (eg. Shifting to park while moving once if ok, but doing a second time will break the car)

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

Live Coding

Lecture Question

- Simulate a Car without using control flow (ie. Use the state pattern)
- In a package named oop.car, create a Class named Car with no constructor parameters
- The Car must contain the following methods as its API:
  - shiftToDrive(): Unit
  - shiftToPark(): Unit
  - shiftToReverse(): Unit
  - accelerate(): Unit
  - brake(): Unit
  - velocity(): Int
- In the tests package, write a test suite named TestCar that will test all the functionality on the spec sheet
  - Note: Only call the API methods while testing. Other classes/methods/ variables you create will not exist in the grader submissions

Car Spec Sheet

- Car is initially in Park
- Initial velocity is 0
- When the Car is in Park:
  - Accelerating and braking have no effect
  - The car can shift into drive or reverse
- When the car is in Drive:
  - Calling accelerate will **increase** the velocity of the car by 10
  - Calling brake will completely stop the car (velocity of 0)
- When the car is in Reverse:
  - Calling accelerate will **decrease** the velocity of the car by 5 (negative velocity)
  - Calling brake will completely stop the car (velocity of 0)
- If the car shift gears while moving (velocity != 0):
- If the car shift gears from reverse to drive or drive to reverse while moving (velocity != 0):
  - The car come to a stop and breaks. No methods have any functionality after this
- If the car shifts to park while moving:
  - The car stops and is damaged
  - If a the car is damaged a second time it will break and have no functionality (eg. Shifting to park while moving once if ok, but doing a second time will break the car)