State Pattern

Each state should have a fight() method that takes in 2 Warriors. The first is the attacker and the second is the defender. The method should return the winning Warrior.

States:
Power – If the attacker's calculated power is greater than the defender's calculated power, the attacker wins. Defender wins on ties.
Attack – If the attacker's calculated attack is greater than the defender's calculated attack, the attacker wins. Defender wins on ties.
Defense – If the attacker's calculated defense is greater than the defender's calculated defense, the attacker wins. Defender wins on ties.
Traditional – If the attacker's calculated attack is greater than the defender's calculated defense, the attacker wins. Defender wins on ties.
Inverse – If the attacker's calculated defense is greater than the defender's calculated attack, the attacker wins. Attacker wins on ties

Combat is the **user** of the state pattern. Combat should have a duel() method that takes in 2 Warriors. The first is the attacker and the second is the defender. Go through each of the 5 states. The Warrior that wins the most of the 5 fights is the winner. The method should return the winning Warrior.

How you implement the state pattern is up to you, but you will be graded on your final decision making. There will be no resubmissions on this assignment that are related to your design choice. **Please see special submission requirement.**

Submission

Same as A1 + **text explanation for why you decided to implement the state pattern the way you did.** The text explanation must be in the BB Learn submission box, **not** a Word doc or file of any kind.

No resubmissions will be allowed for this assignment, as I am grading on your design decisions.

Educator notes:
My hope with the second half of the course is to highlight the benefits of automated testing. Imagine having to manually test whether your fights are correct and whether your duels are correct. With all the complexity of the warrior types, templates, and decorators, the math alone for a full duel should, hopefully, make you cringe.
Test Driven Development (TDD) is the software development paradigm in which no code is written – ever – unless to meet the need of a test. All the software in this course was written using TDD – by me. There are countless benefits to building software in this manner.
If you would like to learn more about this, I encourage you to sign up for SE 570 in the fall or any future term. I designed the entire course to teach and encourage the best software development practices, including clean code.