1 Assignment

Implement a system that plays the "Ace-King-Queen" card game described below. Specifically, your system should include the following:

- an interface for different player strategies as described below.
- multiple different player implementations
- a game engine that competes the players against each other and reports their relative winnings, as described below

2 The Ace-King-Queen Game

The Ace-King-Queen game is pretty much the simplest betting game that there is. The game consists of two players and a deck with three cards: an Ace, a King, and a Queen. Each player has a collection of chips (that represent their money in the game) and they play a series of hands trying to win chips from the other player. Each hand of the game is played as follows:

1. Both players put a single chip into the *pot*; this action is called the *ante*. (The pot is initially empty.)
2. The cards are shuffled and each player is dealt a card at random. The players can look at their cards but don't show the other player at this time.
3. The *betting player* is the first to act. Based on their card, they can either: *bet* by placing a single additional chip into the pot, or *check* by refusing to place further money into the pot.
4. The *calling player* then responds. If the betting player has bet, the calling player can *call* by also putting a single additional chip in the pot or *fold*, which means refusing to put additional chips in the pot and forfeiting the pot. If the betting player checked, then the calling player takes no action.
5. Both cards are revealed, and the winner is decided. If the calling player folded, the betting player automatically wins. Otherwise, the highest card wins: $Ace > King > Queen$. The winner collects all of the chips in the pot.

The two players alternate between playing as the betting player and the calling player.

3 Player Strategies

Clearly, there are a lot of potential strategies for playing this game. For this assignment, you will build a system that permits "plugging" new player strategies into your game engine by creating an `interface` that classes can implement to act as players of the game.

Your interface should be called PlayerStrategy and provide three main kinds of functionality. (You get to decide how many methods it takes cleanly implement this interface.)

1. allow the game engine to query the player about what action it wants to perform given its current role and what card it was dealt.
2. provide feedback to the strategy after the hand about what cards each player had and what

actions they performed.

3. a method that can be called on the strategy to reset its internal state before competing it against a new opponent.

You should design the interface to be as clean as possible and provide the information necessary for any potential strategy.

Implement a couple of different strategies, so you compete them against each other.

4 The Game Engine

Your game engine should be capable of taking a pair of player strategies and running the game for a specified number of turns.

At the end of the game, the game engine should provide information about which player did better. Since this is a zero-sum game (i.e., money is neither created or lost, just transferred from one player to another), you can initialize both players with 0 chips and the score of one player will be the negation of the score of the other player. In this way, it is sufficient to report the score of just the first player.

Create a main function that instantiates one of each of your strategies. Compete each strategy against every strategy (including itself) and report their winnings.