CS112 – Spring 2024

Project01 – Poker

Instructor: Paul Haskell

INTRODUCTION

For Project01 you will develop a program that plays the card game Poker. At the end of the project, we will have a Casino Day in which everyone's programs play a tournament together, with a single Dealer coordinating the play. Grading is not based on the outcome of the tournament (maybe we play more than one), but rather on code quality and sophistication of your approach to planning the game.

Your game is not an interface for a person to play Poker. Your program will be the player, making the playing and betting decisions, playing against another computer program that acts as the Dealer. Your player and the dealer may play hundreds of hands per second!

Before the actual Casino Day, you will have an opportunity to test your program in class with a simplified Dealer.

The Game

There are many different variations of Poker. For this project, your program will play a simple version that you probably have not played before called "Three card stud". Like all poker games, three card stud is a betting game. Each player starts with a stack of money. Every round the player gets cards, places bets, and wins or loses. The objective of the game is to not run out of money for as long as possible. Multiple players (the entire class?) all play the game together, with the Dealer coordinating game play.

Ante

The word "ante" means "before" in Latin. In English, an "ante" is a bet that players must make in a betting game before play starts.

In poker, there is a single "pot" of money. Every round begins with every player putting a $1 bet—the "ante"—into the pot. All other bets made during the game also are placed into the pot. At the end of each round, the winner or winners get the money in the pot. Then the next round begins.

Card Play

At the start of each round, after all the players ante, each player is dealt one card visible only to that player ("face down" or "in the hole") and one card visible to everyone ("face up" or "up"). Beginning with the player with the highest "up" card, each player has an opportunity to place an additional bet. After a player makes a nonzero bet, all following players must:

- match ("see") the bet with an equal one of their own
- raise the bet by making a larger bet
- "fold" i.e. withdraw from the current round of play, conceding a loss in the round

For this project, the maximum bet or raise shall be $10.

So for example, if we have four players playing, named Amit, Brian, Carlos, and Denise, the following bets would be legal:

- Amit bets $10
- Brian raises to $15
- Carlos folds
- Denise sees the $15 bet
- Amit adds $5 to match Brian's bet and also adds $10 more, raising the total bet to $25
- Brian adds $10, to "see" the $25 bet
- Denise folds

At this point, everyone playing either has folded or has matched the current bet.

After the first two cards have been dealt and the first round of betting, the Dealer gives one more card, face up (visible to everyone), to each player who has not folded. Then there is another round of betting, conducted with the same rules as the first round.

After the second round of betting, a winner is determined.

- Three-of-a-kind beat any other hand. If more than one player has three-of-a-kind, then the player with the higher card value wins.
- The next highest hand is a pair (two cards with the same value). If more than one player has a pair, the pair with the highest card value wins (ignoring the third card). If two players have pairs with matching values, then the values of only those players' remaining cards are compared, and the higher card wins. If all three cards match, then the two players are both winners and split the winnings.
- If no player has a pair, then the player with the highest card wins. If more than one player has the same highest card, then those players only compare their second-highest (and third-highest if necessary) cards to determine the round's winner.
- One more rule to keep things fun: the player who wins the poker hand only wins half the pot! The other half of the pot goes to the player with the highest "spade in the hole", that is, the highest face-down card whose suit is Spades. If no player has a Spade "in the hole", then the winner of the poker round wins the entire pot. If the same player wins the poker hand and has the highest spade in the hole, that player wins the entire pot.

Reshuffling

The Dealer reshuffles the card deck before every round of play.

Betting

Bets must be made in increments of $1. A player is not permitted to bet more money than remains in her stack, so a player may be forced to fold. When a player runs out of money, i.e. has no money left to ante, then she is out of the game.

Betting Zero

A player may bet zero chips on any opening bet. This is a wise bet if the player's hand is not very strong. Sometimes, every player in a round will bet $0, and only the ante bets will be claimed by the round's winner.

Your Program

The instructor has written a Java program that will perform as Dealer. Each of your **Poker.java** programs will communicate with the Dealer over the classroom WiFi network, using Internet Protocol ("IP"). Luckily, Java makes it easy to set up this communication.

Communication with Sockets

Your **Poker.java** program should take two command-line arguments:

- IpAddress: IP network of Dealer
- IpPort: IP port number of server

Your program should include something similar to the following code—you need not understand all of this code, please simply copy it into your program:

```
import java.net.Socket;
import java.io.IOException;
import java.io.DataInputStream;
import java.io.DataOutputStream;

Socket socket = new Socket(args[0], Integer.parseInt(args[1]));
DataInputStream dis =
        new DataInputStream(socket.getInputStream());
DataOutputStream dos =
        new DataOutputStream(socket.getOutputStream());


private void write(String s) throws IOException {
        dos.writeUTF(s);
        dos.flush();
}
private String read() throws IOException {
        return dis.readUTF();
}
```

Your program will read commands from the Dealer by calling `read()` and looking at the returned `String`. Your program will write responses to the Dealer by calling `write()`, passing in your program's response.

Please think about where in your program to put the code that creates `socket`, `dis`, and `dos`, and the `read()` and `write()` methods.

Game Protocol

A protocol is a set of rules by which two (or more) entities interact with each other. Your program will communicate with the Dealer by implementing the following protocol. The Dealer will initiate communications every time, by sending a command (along with data) to your program. Your program should repeatedly try to read from the Dealer. When a command is received, your program interprets it and responds. Several commands require a reply. Here are the Dealer commands and required replies:

- **login** – whenever your program receives this command, it must reply with
  `<<your GitHubId>>:<<your avatar name>>`
  For example
  `StephenCurry:Steph`
  Your program may receive the "login" command more than once. You always must reply with the same values. Your "avatar" name will be shown on a graphics window, to update everyone on the Casino Day game status. Your GitHubId will be known only to the instructor.
- **bet1:<<number of chips in your stack>>:<<size of the current pot>>:<<current bet to match or beat>>:<<your "hole" card>>:<<your "up" card>>:up:<<first player's "up" card>>:<<second player's "up" card>>::etc**
  For example,
  bet1:208:24:12:KS:10D:up:AS:8H:10D:QD:2C
  Notice that your "up" card is in the list of dealt cards.

  You must reply with one of the following:
  `bet:<<amount of your bet for this hand>>`
  `fold`

  If you do not fold, your bet must be at least as large as the current bet, and no more than $10 larger. A sample response to the above open command could be:
  `bet:20`

  If players who bet after you raise your bet, you will receive another "bet1" command. The format will be the same as shown above. Your "hole" card and "up" card values will be the same, and the other players' cards will be the same. Your stack will be reduced by your previous bet, and the current bet to match or beat may be different.
  For example, after the above command, you may receive another command such as
  `bet1:188:68:5:KS:10D:up:AS:8H:10D:QD:2C`
  The "188" reflects the 20 that you bet previously (208 - 20 = 188), and the "5" is the new bet amount you must match or beat, or else you must fold.

  After the first round of betting finishes, if you have not folded, you will receive a "bet2" command.

  - **bet2:<<number of chips in your stack>>:<<size of the current pot>>:<<current bet to match or beat>>:<your "hole" card>>:<<your first "up" card>>:<<your second "up" card>:up:<<first player's first "up" card>>:<<first player's second "up" card>>:<<second player's first "up" card>>:<<second player's second "up" card>>:etc**

For example:
bet2:183:66:0:KS:10D:10S:up:AS:AQ:8H:6D:10D:4S:QD:JC:2C:4H

- As with the first round, you must reply with one of the following:
  `bet:<<amount of your bet for this hand>>`
  `fold`

  If you bet, you must bet at least the amount of the current bet and at most $10 more. As with the first betting round, in the second round, betting continues until every player has bet the same total amount or folded.

  If you are the only player not to fold after the "bet1" round, you simply will receive a status command, without a "bet2" command.

- **status:<<win** or **lose>>:<<first card of winning hand>>:<<second card of winning hand>>:<<third card of winning hand>>** - you should not reply to this message. You can print it out, parse it for use playing the game, etc. You get this message after each round ends.
- **done:<<message>>** – you should not reply to this message. You are done with the game, either because you ran out of money or because you broke the rules. The `message` will explain why, so it is a good idea to print out the message. After your program receives the **done** command, it should close the `socket` and exit the program.

Details

- As you see, commands and replies often use the colon character ":" as a separator.
- Some of the commands contain a card or list of cards. A card is represented with two or three characters. The last character is the "suit": S for spades, H for hearts, C for clubs, D for diamonds. The first character is the card value: 2-9, J for jack, Q for queen, K for king, A for Ace. Only the 10 requires three characters: "10" plus the suit. Examples are:
  2C
  AD
  10S
  QH
- It is a requirement that your program respond to every command within 1 second, or else your program will be deemed nonresponsive, and you will be kicked out of the game.

Strategy for Play

Here are a few hints:

The only decisions your program must make are whether to fold or to bet, and how much to bet. Clearly if you have three-of-a-kind, you should bet as much as possible. As a simple strategy, you also can bet big if you have a pair, and bet $0 or fold if you do not have a pair or better.

More complicated strategies will consider the "up" cards from the other players to try to figure out the probability that you will win the current hand. Even more complex strategies will consider the expected pay-off of betting vs folding.

The object of the game is to keep playing as long as possible. If you have a bet that has a 70% chance of winning, you bet your whole stack of chips, and you lose, then you are out of the game. A smaller bet might be better.

The "high spade in the hole" is just as valuable as a good poker hand. If you have a high spade in the hole, you should bet high. If you see high spades among the "up" cards, that helps confirm that other players do not have those cards as their "hole" cards.

If you (almost) never bet, you will lose money steadily from having to ante as other players win the pot. This strategy might keep you playing for a while, but it is unlikely to win any tournament.

Grading

Your program's performance in the Casino Day tournament does not affect your grade—Casino Day is just for fun.
Your deliveries for this project must come in two parts

- Part 1 supports Internet connectivity and legal response to commands
- Part 2 implements your best algorithms for betting and for playing

For grading, after Part 2, you must schedule a 10-minute time slot with the instructor to present your code to the instructor or TA's. (We will post a sign-up sheet with time slots for the interviews.) During this meeting, you will explain your strategy for playing the game and will give a walkthrough of your code. Your code will go through an automatic tester beforehand (separate from the class Casino Day) to see how well it followed the rules of the Project.

Put your program into a subdirectory called "**Project01**" (not Lab15) inside your **MyWork** directory, and remember to push your Project01 to GitHub before the deadlines.

- Part 1 must be turned in before 11:59pm Weds March 27
- Part 2 must be turned in before 11:59pm Fri April 5

Rubric

Milestone	Points	Comments
Part 1: Program is in correct location, with correct name, pushed to GitHub before Part 1 deadline, compiles successfully, and connects to Dealer server	15	
Part 1: correct play: program gives correct reply messages	35	Response must be given within 1 second. Always bets a legal amount. Always replies to "play" command with legal reply.

		Closes socket after "done" command.
Part 2: correct play: program gives correct reply messages, plays correctly in given test situations	70	Response must be given within 1 second. Always bets a legal amount. Always replies to "play" command with legal reply. Closes socket after "done" command.
Part 2: Intelligent rules for play	40	Auto grader tests multiple game scenarios
Part 2: Intelligent rules for play that take advantage of other players' "up" cards	20	Auto grader tests multiple game scenarios
Part 2: Software quality	30	Judged subjectively by graders
10-minute interview shows understanding of one's own software. Bring an informal one-page document describing how you tested your program	40	Judged subjectively by graders

Conclusion

I hope this project ends up being fun. You will implement a communication protocol, your program will communicate with other computers, you will get to explore game strategy, and to decide what strategy to implement.

This project is an excellent opportunity for you to practice testing! You might want to build a simple Dealer to deal random cards to your Player, to see which playing strategies do well and which do poorly. You will certainly want to test to ensure that your program implements the game protocol correctly.