Assignment 1

The goal of this assignment is to get statistical data about Blackjack using a Charlie plugin.

 if you did not complete Lab 1. Go back and complete that lab first.

**Background**

Charlie maintains the global game-state but only informs players of *changes* in game-state. For instance, in Blackjack a player can “win” as follows:

|  |  |
| --- | --- |
| Condition | Outcome |
| 1. Player hand value > dealer hand value where both are ≤ 21 | Win |
| 2. Dealer breaks (i.e., dealer hand value > 21). | Win |
| 3. Player has Blackjack | Blackjack |
| 4. Player has Charlie | Charlie |

**Table A. Winning player outcomes**

Notice that although there are four ways for the player to win, there are two distinct *Win* outcomes, one *Blackjack* outcome, and one *Charlie* outcome. (A “charlie” is the case where the player has a five-card charlie--five cards with the hand value ≤ 21.)

A player can “lose” as follows:

|  |  |
| --- | --- |
| Condition | Outcome |
| 1. Player hand value < dealer hand value where both are ≤ 21 | Lose |
| 2. Player breaks. | Bust |

**Table B. Losing player outcomes**

Notice here that although there are two ways for the player to lose, there is one *Lose* outcome and one *Bust* outcome.

A *Push* outcome is the case where player hand value = dealer hand value.

Thus, the total number of player hands = sum(Win, Blackjack, Charlie, Lose, Bust, Push) outcomes.

Charlie does *not* maintain win-lose statistics for a player, that is, the number of Win outcomes, Blackjack outcomes, etc. Charlie only tracks bankroll, that is, the dollar credits. However, Charlie provides the “trap” design pattern implemented through the ITrap Java interface. It’s in the charlie.plugin package. When plugged in properly, ITrap receives game state changes via inbound and outbound messages which it can be used for various purposes, including to accumulate outcome statistics of number of wins, losses, Blackjacks, Charlies, etc. outcomes.

The ITrap is a client-side interface with two behaviors: onReceive and onSend. Charlie invokes onReceive whenever it gets an inbound message from the server and onSend whenever it sends an outbound message to the server. Both methods have one formal parameter: Message, an abstract base class. Message has a subclass, charlie.message.view.to.Outcome[[1]](#footnote-0) which itself has multiple subclasses: Win, Lose, Push, etc. that are the actual parameters of onReceive and onSend. In turns out that the outcomes in Tables A and B above are Java classes in Charlie, named as such.

The Outcome classes inherit a member, Hid (or hand id) from Message. Hid contains a member, Seat, which is an enumerated type with four possibilities: RIGHT, LEFT, YOU, and DEALER. They correspond to the Charlie player to whom the message refers. For instance, YOU is the human player.

Splits are not a subclass of Outcome but an immediate subclass of Message: SplitResponse.

For efficiency reasons, Charlie broadcasts game state changes to all players. Thus, it’s the player’s responsibility, whether as a “real” player or a bot, to update its local game-state and if necessary, respond accordingly. Use SplitResponse to count the number of splits.

**Example**

For instance, the snippet below shows how to measure *only* the number of YOU wins, i.e., *not* Blackjacks or Charlies.

|  |
| --- |
| public class Trap implements ITrap {  public final static Logger LOG = Logger.getLogger(Trap.class);  protected int youWin = 0;  @Override  public void onReceive(Message message) {  if(message instanceof Win) {  Hid hid = ((Outcome) message).getHid();  if(hid.getSeat() == Seat.YOU)  youWin++;  }  LOG.info(“wins: “+youWin);  }  @Override  public void onSend(Message message) {  } } |

**Snippet 1. How to trap YOU Win outcomes. (Note: the code leaves out package and import statements.)**

The dealer indirectly transmits information about changes in bankroll. How so? Through Outcome. For instance, if the player wins via condition #1 in Table A, invoke getAmt on Hid to get the bet magnitude of change in bankroll: Win (i.e., ×1.0), Blackjack (i.e., ×1.5), Charlie (i.e., ×2.0) and Lose and Bust (i.e., × 1.0). However, the direction of change, plus or minus, infer from Outcome. For instance, if Win, increment bankroll by the amount in Hid; if Lose, decrement bankroll by the amount.

|  |
| --- |
| **How to write diagnostic logs**  Using the logger is an inexact science. Logging too much info, not enough, irrelevant, incorrect, or redundant info, etc. are all issues. Two best practices are: 1) use a standard logger utility like *log4j* which Charlie uses; and 2) write diagnostics appropriate to the *level*. Log4j supports several levels. For instance, the “info” level is meant to report state and progress information whereas the “error” level is for serious faults. Charlie uses the log4j.properties to configure the logger so that diagnostic information is written both to log file and the console. |

**Preparation**

Before starting to code do the following: review *The Elements of Java Style* (Ambler, et al, 2000) -- required reading for this course.

Include Javadoc, a guide you can find [here](https://www.oracle.com/technetwork/java/javase/tech/index-137868.html).

**Tasks**

Part I - Configure the Charlie trap plugin.

1. Download the latest version of Charlie from GitHub.com
2. Open Charlie in NetBeans.
3. Open charlie.props and add this line...  
     
   charlie.trap charlie.client.Trap

Part II - Create a new NetBeans project, MyCharliePlugins.

1. Create a new NetBeans project as follows:  
     
   Click on File > New Project… > Java > Java Class Library  
   1. Give it the name “MyCharliePlugins”
   2. Make the Project Location: some suitable folder on the Thawed (or D) drive, if you are working in the lab or some other appropriate place on your personal computer.
   3. Right-click on MyCharliePlugins and do Clean and Build.
2. Link MyCharliePlugins to Charlie.
   1. Right-click on MyCharliePlugins > Properties > Libraries > Add Project… and browse to, (the Charlie project) then click Add Project/Jar Files.
   2. Right-click on Charlie > Properties > Libraries > Add Jar/Folder and browse to MyCharliePlugins.jar in the dist folder then click open.

Part III - Create the trap plugin in MyCharliePlugins project.

1. Create a new package by right-clicking on Source Packages > New… > Java Package… and giving it the name charlie.client.  
   1. In this package create a new Java class, Trap by right-clicking on the charlie.client package > New > Java Class...
   2. Make Trap implement charlie.plugin.ITrap.  
        
      Note: If you click on the little red dot (see left, below), NetBeans proposes to add the import for charlie.plugin.ITrap.  
        
      
   3. Implement the onReceive and onSend methods initially with no code.  
        
      Note: If you click on the little red dot again, NetBeans proposes to implement all abstract methods to make Trap a concrete class.  
        
      Observe that if NetBeans implements onReceive and onSend, it generates code to throw UnsupportedOperationException in both cases; you’ll change this code below.

Part IV - Complete the trap plugin.

Implement the onReceive method as suggested in the “Background” section. Delete the code to throw the exception and replace it with code to instead write diagnostic info about the state of the game in terms of wins, losses, etc..

1. Copy log4j-y.jar (where y is the current version of log4j) from the Charlie project root directory to MyCharliePlugins root directory.
2. Right-click on the MyCharliePlugins project > Properties > Libraries > Add Jar/ Folder and open the log4j jar.
3. In Trap.java, import org.apache.log4j.Logger.
4. Finally, in the class scope of Trap.java, include this statement:  
     
   protected static Logger LOG = Logger.getLogger(Trap.class);  
     
   Use LOG to write diagnostic info (literally, use the info method of LOG) as suggested by the “Background” section to output game-state, e.g., number of player wins, loses, etc.
5. Start with the default bankroll from charlie.util.Constant.PLAYER\_BANK\_ROLL and use the trap messages and the bet amount (it’s the amt member in Hid) to track the player bankroll.  
     
   For instance, if YOU lose or break, subtract the bet amount from the current bankroll.

Part V - Gather the game statistics.

1. Clean and build MyCharliePlugins then Charlie.
2. Follow the instructions for Exercises 1 & 2 below.

**Deliverables**

1. Upload Trap.java to the assignment shell.
2. Upload the renamed client and server diagnostic logs (see below).
3. Complete Exercises 1 & 2 below, rename this .docx to <your-name>-assign1.docx, and upload it to the assignment shell.

**Evaluation**

I evaluate the assignment on the basis of delivery of *all* the components as specified, correctness of the results, correctness of the code, appropriateness of the diagnostic logs, and good programming style.

**Exercise 1**

1. Restart Charlie and play until the end of the game *after* the first burn card. Look in the diagnostic log and get the game statistics the ITrap recorded.
2. Enter the data into Table 1 below, that is, the number of wins, loses, breaks, etc. and the bankroll for the player.
   1. Print a hardcopy of the table including your name(s).
   2. Rename the client-side diagnostic log file as ex1-log-client.out and upload it to the assignment shell.
3. Upload the client-side diagnostic log file.

DELETE EVERYTHING ON THE PAGE ABOVE THIS LINE.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Name(s):

Table 1. Tabulation of game statistics with the default shoe.

|  |  |
| --- | --- |
|  | Player |
| Wins |  |
| Loses |  |
| Pushes |  |
| Breaks (>21) |  |
| Blackjacks |  |
| Charlies |  |
| Total |  |

Splits:

Bankroll:

**Exercise 2**

1. Stop Charlie.
2. Open the Charlie properties file, charlie.props, and change the line…  
     
   charlie.shoe charlie.card.Shoe01  
     
   and make it  
     
   charlie.shoe charlie.card.Shoe00  
     
   This change configures Charlie to use a six deck shoe where you won’t encounter the burn card for play for 80 - 90 games.
3. Clean and build *both* projects.
4. Restart Charlie and play until the end of the game *after* the first burn card. Look in the diagnostic log and get the game statistics that ITrap recorded.
5. Enter the data into Table 2 below, that is, the number of wins, loses, breaks, etc. and bankroll for the player.
   1. Print a hardcopy of the table including your name(s).
   2. Rename the client-side diagnostic log file as ex2-log-client.out and upload it to the assignment shell.
   3. Staple this sheet with the prior one and bring both to class for handin.

DELETE EVERYTHING ON THE PAGE ABOVE THIS LINE.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Names(s):

Table 2. Tabulation of game statistics with Shoe00.

|  |  |
| --- | --- |
|  | Player |
| Wins |  |
| Loses |  |
| Pushes |  |
| Breaks (>21) |  |
| Blackjacks |  |
| Charlies |  |
| Total |  |

Splits:

Bankroll:

1. There is another Outcome, an enumerated type of AHand. Don’t use that Outcome. [↑](#footnote-ref-0)