**Project 8 – Blackjack**

**(20 points)**

**Name: Ozaner Hansha**

**Due Date: Friday, January 29, 2016**

**Description:**

In this project, you will learn how to write an application according to a simplified version of the Model-View-Controller (MVC) pattern. One of the great benefits of MVC is the strict separation of concerns in separately coding the model from the view. The GUI will use some of the Swing interactors (JLabels, JButtons, etc.) and will incorporate interfaces.

Here is how MVC applies in situations such as our Blackjack game. We want to separate out the knowledge about the logic of the Blackjack game into one set of classes and knowledge about the GUI in a separate set of classes. Doing this allows the model to be reused with many different kinds of user interfaces. This releases the inventiveness of other people in application design, user interface design, graphics, animation, sound, etc.

In our ACM Graphics setting, we will combine the View and Controller aspects of the MVC pattern into our Blackjack class, which is the main application class (extends GraphicsProgram). It will create a BlackjackModel instance and pass itself in the constructor, so that the model can make callbacks to alert the view as to important changes in the state of the game (the player got dealt a certain card, the dealer won, etc.)

How do you document in code (vs. English) what methods a potential coder will need to provide in a compatible view class? An interface!! All we need to do is to provide a BlackjackView interface. The BlackjackModel class can refer generically to the BlackjackView interface class. Any potential implementation of BlackjackView is said to “implement” the interface. Notice the keyword “**implements**” in the class declaration below, rather than the keyword “**extends**”.

**public** **class** Blackjack **extends** GraphicsProgram **implements** BlackjackView {

**private** BlackjackModel bm;

/\*\*

\* Create the Model (passing 'this' for callbacks) and set up the GUI.

\*/

**public** **void** init() {

bm = **new** BlackjackModel(**this**);

setSize(***INITIAL\_WIDTH***, ***INITIAL\_HEIGHT***);

setBackground(Color.***LIGHT\_GRAY***);

//...

From the perspective of the BlackjackModel, it is passed a BlackjackView. It is guaranteed that the instance will be able to receive the messages (method calls) that it wants to use to signal important events that arise in the course of the game:

**public** **class** BlackjackModel {

**private** BlackjackView bv;

. . .

/\*\*

\* Create a BlackjackModel (given a BlackjackView for notifications).

\* **@param** blackjackView

\*/

**public** BlackjackModel(BlackjackView blackjackView) {

bv = blackjackView;

. . .

}

Here is what the BlackjackView interface looks like. Notice that it has no implementation, only declarations (method signatures) that are implicitly public abstract. Any class such as Blackjack that purports to implement BlackjackView must provide code for all of these declared methods. Think of an interface as a contract between programmers. One coder is saying to the other, “my Blackjack model will properly work if you supply a meaningful application environment that adheres to and provides (view) code for this interface specification.”

**public** **interface** BlackjackView {

/\*\*

\* Place the card dealt to the player on the canvas.

\* The card size is rescaled to be consistent with the current deck image size.

\* **@param** card the card to be added

\*/

**void** cardDealtToPlayerNotification(BlackjackGCard card);

/\*\*

\* Place the card dealt to the dealer on the canvas.

\* The card size is rescaled to be consistent with the current deck image size.

\* **@param** card the card to be added

\*/

**void** cardDealtToDealerNotification(BlackjackGCard card);

/\*\*

\* End of game result notification. Dealer and player both bust.

\*/

**void** bothBustNotification(**int** wins, **int** losses, **int** ties);

/\*\*

\* End of game result notification. Dealer wins, player busts.

\*/

**void** youBustDealerWinsNotification(**int** wins, **int** losses, **int** ties);

/\*\*

\* End of game result notification. Dealer busts, player wins.

\*/

**void** dealerBustYouWinNotification(**int** wins, **int** losses, **int** ties);

/\*\*

\* End of game result notification. Player beats dealer.

\*/

**void** youBeatDealerNotification(**int** wins, **int** losses, **int** ties);

/\*\*

\* End of game result notification. Dealer beats player.

\*/

**void** dealerBeatsYouNotification(**int** wins, **int** losses, **int** ties);

/\*\*

\* End of game result notification. Dealer and player tie.

\*/

**void** bothTieNotification(**int** wins, **int** losses, **int** ties);

/\*\*

\* End of game result notification. Player quits game and loses.

\*/

**void** quitGameNotification(**int** wins, **int** losses, **int** ties);

}

In this project, I am giving you clean versions of the supporting classes that we developed earlier for writing card games. I am also providing the BlackjackView interface, as listed above.

In Eclipse, you can import the file APCS2015Proj08Blackjack.zip.

1. Study the supplied classes carefully so that you thoroughly understand how they work. The code should be familiar from our earlier project. Do ***not*** change the supplied classes. The idea is to show that they can be re-used without modification in a new application context.
2. Complete the Model by finishing the classes BlackJackGCard, BlackjackHand and BlackJackModel. BlackJackHand extends the Hand class with any features that are of unique interest in a Blackjack game. BlackJackGCard extends the GCard class with any features that are of unique interest in a Blackjack game. BlackJackModel is an implementation of the rules of Blackjack. It contains methods which are called from the BlackJack class (our View/Controller) in response to input events such as startGame, hit, stay, and quitGame. It notifies BlackJack when any notable state changes occur in the model that may need to be reflected in the GUI.
3. Complete the BlackJack class. Use GUI interactors to add a JLabel in the North boundary for status messages. In the South, you will add JLabels and JButtons. Use the card images that we used earlier for the graphics. They are included again in the zip file.
4. Submit this Word document to your shared Google Drive folder

* your name filled out above
* the source listings for classes other than the supplied classes of Card, Deck, GCard, Hand, Rank, Suit, and BlackjackView
* key screen captures of your Blackjack GUI interface

**GUI Example**





**Appendix**

**Here are the rules for a simple version of Blackjack (also known as Twenty-One [21])**: Assume that there are only two participants – the dealer and you – and that a standard, shuffled, 52-card deck is used, fresh for each game.

* A card is dealt face-up to you and to the dealer.
* A second card is dealt face-up to you and face down to the dealer (this is called the “hole” card).
* You may draw as many additional cards as you like, but if they total more than 21 you lose. If you want another card say “Hit”; if you don’t want another card, say “Stay”.
* The dealer then draws additional cards according to the house rules.
* Face cards are worth 10. Aces are either 1 or 11. The combination of an ace with a card other than a ten-card is known as a "soft hand," because the player can count the ace as a 1 or 11, and either draw cards or not. For example with a "soft 17" (an ace and a 6), the total is 7 or 17. While a count of 17 is a good hand, the player may wish to draw for a higher total. If the draw creates a bust hand by counting the ace as an 11, the player simply counts the ace as a 1 and continues playing by standing or "hitting" (asking the dealer for additional cards, one at a time).
* If a player’s first two cards are an ace and a “ten card” (a face card of 10), this is a “blackjack” (also known as a “natural”). If either a player or the dealer has a blackjack, then the game is immediately over.

There are relatively few decisions to make when playing Blackjack. You must consider your cards and your dealer's card and remember, if you go over 21, you "bust", and if you "bust" you lose.

The dealer draws cards according to the following house rules:

* If the total is a hard 17 or more, he must stand (stay). If the dealer has an ace, and counting it as 11 would bring his total to 17 or more (but not over 21), he must count the ace as 11 and stand.
* If the total is 16 or under, he must take a card.

Outcomes:

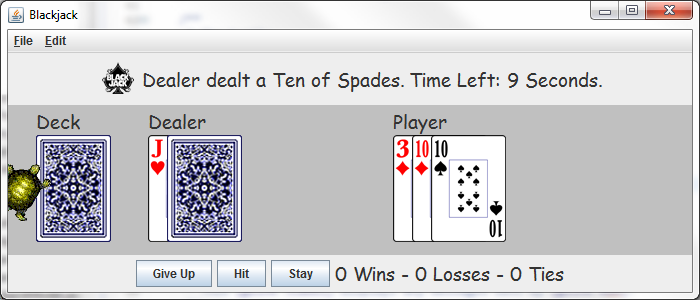
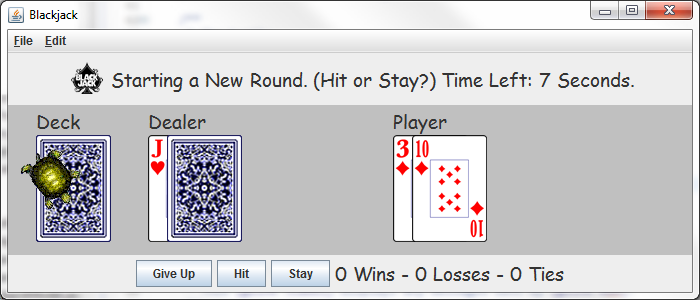
The outcomes differ in multi-person, casino betting versions of the game than in the simple, home rules given below.

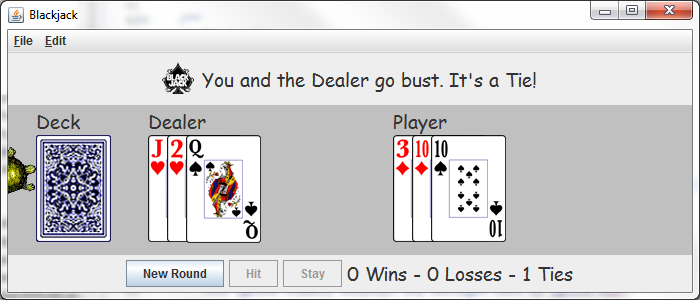
* You win if your total is higher than the dealer and you did not bust.
* You lose if the dealer’s hand is higher than yours and does not bust.
* You win if you have a blackjack (Ace plus a card worth 10) and the dealer does not have a blackjack (even if he has a total of 21).
* You lose if the dealer has a *blackjack* and you do not (even if you have a total of 21).
* You tie (it is a “stand-off”) if your hand and the dealer’s hand tie for the highest total or both hands bust.

Cards in Blackjack are assigned the following point values:

* Picture Cards (Jack, Queen and King) each count as 10 points.
* An Ace counts as 1 point or 11 points, whichever is better for owner of the hand.
* All other cards have their numerical face value.

***Screenshots:***





***Blackjack Class:***

**package** unit6.blackjackProject;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Font;

**import** java.awt.event.ActionEvent;

**import** java.util.Random;

**import** java.util.Timer;

**import** java.util.TimerTask;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JLabel;

**import** acm.graphics.GImage;

**import** acm.graphics.GTurtle;

**import** acm.program.GraphicsProgram;

/\*\*

\* A Blackjack game using the ACM Library.

\* <br><br>

\* AP Computer Science<br>

\* 1/28/16<br>

\* Dr. Jones

\* **@author** Ozaner Hansha

\*/

@SuppressWarnings("serial")

**public** **class** Blackjack **extends** GraphicsProgram **implements** BlackjackView {

/\*\*

\* Initial window size.

\*/

**private** **static** **final** Dimension ***INITIAL\_SIZE*** = **new** Dimension(700,300);

/\*\*

\* Coordinates of where the card stacks start.

\*/

**private** **static** **final** **double** ***DECK\_X*** = ***INITIAL\_SIZE***.getWidth() \* .04,

***PLAYER\_X*** = ***INITIAL\_SIZE***.getWidth() \* .55,

***DEALER\_X*** = ***INITIAL\_SIZE***.getWidth() \* .2,

***CARD\_Y*** = ***INITIAL\_SIZE***.getHeight() \* .1;

/\*\*

\* How much to offset new cards added to dealer/player's hand.

\*/

**private** **static** **final** **double** ***CARD\_OFFSET*** = BlackjackGCard.*cardWidth*() \* .25;

/\*\*

\* The Y Coordinate of the name tags.

\*/

**private** **static** **final** **double** ***NAME\_Y*** = ***INITIAL\_SIZE***.getHeight() \* .01;

/\*\*

\* The font of the {@link #nameTags}.

\*/

**private** **static** **final** Font ***NAME\_FONT*** = **new** Font("Comic Sans MS", Font.***PLAIN***, 19);

/\*\*

\* How long, in seconds, a turn is.

\*/

**private** **static** **final** **int** ***TURN\_TIME*** = 10;

/\*\*

\* The turtle.

\*/

**private** GTurtle turtle = **new** GTurtle();

/\*\*

\* Initial Coordinates of turtle.

\*/

**private** **static** **final** **double** ***TURTLE\_X*** = ***INITIAL\_SIZE***.getWidth() \* .09,

***TURTLE\_Y*** = ***INITIAL\_SIZE***.getWidth() \* .11;

/\*\*

\* A pointer to the {@link BlackjackModel} corresponding with

\* this instance of {@link Backjack} for callbacks.

\*/

**private** BlackjackModel bm;

/\*\*

\* This {@link JLabel} displays any messages sent by {@link #bm}.

\*/

**private** JLabel notifications = **new** JLabel();

/\*\*

\* This {@link JLabel} displays the scores of the games.

\* **@see** #updateScoreboard()

\*/

**private** JLabel scoreboard = **new** JLabel();

/\*\*

\* How much time is remaining this turn.

\*/

**private** JLabel timerLabel = **new** JLabel();

/\*\*

\* Keeps track of how long the player has to make their move.

\*/

**private** Timer timer = **new** Timer();

/\*\*

\* Name tags for the deck, dealer, and the player.

\*/

**private** JLabel[] nameTags = {**new** JLabel("Deck"),

**new** JLabel("Dealer"),

**new** JLabel("Player")};

/\*\*

\* The 3 turn buttons.

\*/

**private** JButton[] buttons = {**new** JButton("New Round"),

**new** JButton("Hit"),

**new** JButton("Stay")};

/\*\*

\* An image of a facedown card, representing the deck.

\*/

**private** GImage deck = BlackjackGCard.*getBackImage*();

/\*\*

\* Entry point when running Blackjack as an application.

\* **@param** args

\*/

**public** **static** **void** main(String[] args)

{

(**new** Blackjack()).start();

}

/\*\*

\* Create the Model (with this for callbacks).

\* Set up the GUI.

\*/

@Override

**public** **void** init()

{

bm = **new** BlackjackModel(**this**);

setSize(***INITIAL\_SIZE***);

setBackground(Color.***LIGHT\_GRAY***);

notifications.setText("Welcome to BlackJack. Press New Game to start a new round.");

notifications.setIcon(**new** ImageIcon(Blackjack.**class**.getResource("/unit6/blackjackProject/bjLogo.png")));

notifications.setFont(***NAME\_FONT***);

timerLabel.setFont(***NAME\_FONT***);

scoreboard.setFont(***NAME\_FONT***);

//GUI

add(notifications, ***NORTH***);

**for**(JButton b: buttons)

{

add(b, ***SOUTH***);

}

buttons[1].setEnabled(**false**);

buttons[2].setEnabled(**false**);

add(deck, ***DECK\_X***, ***CARD\_Y***);

**for**(JLabel j: nameTags)

j.setFont(***NAME\_FONT***);

add(nameTags[0], ***DECK\_X***, ***NAME\_Y***);

add(nameTags[1], ***DEALER\_X***, ***NAME\_Y***);

add(nameTags[2], ***PLAYER\_X***, ***NAME\_Y***);

add(timerLabel, ***NORTH***);

add(scoreboard, ***SOUTH***);

add(turtle, ***TURTLE\_X***, ***TURTLE\_Y***);

startTurtleAI();

updateScoreboard(0,0,0); //Starts scoreboard at 0

addActionListeners();

}

/\*\*

\* Starts the turtle ai on a new thread.

\*/

**public** **void** startTurtleAI()

{

Timer timer = **new** Timer();

timer.scheduleAtFixedRate(**new** TimerTask(){

**public** **void** run()

{

//"A.I" controlled greg army

**switch**(**new** Random().nextInt(5))

{

**case** 0: turtle.left(12); **break**;

**case** 1: turtle.right(12); **break**;

**case** 2: turtle.forward(2); **break**;

**case** 4: turtle.forward(-2); **break**;

}}}, 25, 5);

}

/\*\*

\* Starts the turn timer.

\* (Can't put timer in model because interface is not editable.)

\*/

**public** **void** startTimer()

{

timer.cancel();

timer = **new** Timer();

timer.schedule(**new** TimerTask()

{

**int** time = ***TURN\_TIME***;

@Override

**public** **void** run()

{

**if**(time >= 0)

{

timerLabel.setText("Time Left: " + time + " Seconds.");

time--;

}

**else** **if**(time == -1)

{

bm.stay(); //Will stay automatically if nothing is done.

cancel();

timer = **new** Timer();

}

}

},0,1000);

}

/\*\*

\* Starts a new round.

\*/

**public** **void** newRound()

{

**if**(!bm.isGameInProgress())

{

//Removes all cards from canvas.

**for**(**int** x = getGCanvas().getElementCount()-1; x > 0; x--)

{

**if**(getGCanvas().getElement(x) **instanceof** BlackjackGCard)

{

getGCanvas().remove(getGCanvas().getElement(x));

}

}

bm.newRound();

notifications.setText("Starting a New Round. (Hit or Stay?)");

buttons[0].setText("Give Up");

buttons[1].setEnabled(**true**);

buttons[2].setEnabled(**true**);

startTimer();

}

**else**

{

bm.quitGame();

timerLabel.setText("");

timer.cancel();

}

}

/\*\*

\* Done after a round is over.

\*/

**public** **void** endRound(**int** wins, **int** losses, **int** ties)

{

//Turn all cards face up.

**for**(**int** x = 0; x < getGCanvas().getElementCount(); x++)

{

**if**(getGCanvas().getElement(x) **instanceof** BlackjackGCard)

{

((BlackjackGCard)getGCanvas().getElement(x)).turnFaceUp();

}

}

updateScoreboard(wins, losses, ties);

buttons[0].setText("New Round");

buttons[1].setEnabled(**false**);

buttons[2].setEnabled(**false**);

}

/\*\*

\* Updates the scoreboard with the current wins, losses, and ties.

\*/

**public** **void** updateScoreboard(**int** wins, **int** losses, **int** ties) {

scoreboard.setText(String.*format*("%d Wins - %d Losses - %d Ties", wins, losses, ties));

}

/\*\*

\* Handler for button actions.

\*/

@Override

**public** **void** actionPerformed(ActionEvent e)

{

**if**(e.getActionCommand() == "Hit")

{

bm.hit();

startTimer();

}

**else** **if**(e.getActionCommand() == "Stay")

{

bm.stay();

timer.cancel();

timerLabel.setText("");

}

**else** //Must be reset, only 3 buttons.

newRound();

}

/\*\*

\* This is a helper method for capitalizing strings.

\* **@param** string - String to capitalize.

\*/

**public** **static** String capitalize(String string)

{

**return** string.charAt(0) + string.toLowerCase().substring(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\* Notifications \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* Place the card dealt to the player on the canvas.

\* The card size is rescaled to be consistent with the current deck image size.

\* **@param** card the card to be added

\*/

@Override

**public** **void** cardDealtToPlayerNotification(BlackjackGCard card) {

notifications.setText(String.*format*("Dealer dealt a %s of %s.",

*capitalize*(card.getRank().name()), *capitalize*(card.getSuit().name())));

add(card, ***PLAYER\_X*** + (bm.getPlayerCardNumber() -1) \* ***CARD\_OFFSET***, ***CARD\_Y***);

}

/\*\*

\* Place the card dealt to the dealer on the canvas.

\* The card size is rescaled to be consistent with the current deck image size.

\* **@param** card the card to be added

\*/

@Override

**public** **void** cardDealtToDealerNotification(BlackjackGCard card) {

notifications.setText("Dealer dealt a card to himself");

add(card, ***DEALER\_X*** + (bm.getDealerCardNumber() -1) \* ***CARD\_OFFSET***, ***CARD\_Y***);

}

/\*\*

\* End of game result notification. Dealer and player both bust.

\*/

@Override

**public** **void** bothBustNotification(**int** wins, **int** losses, **int** ties) {

notifications.setText("You and the Dealer go bust. It's a Tie!");

endRound(wins, losses, ties);

}

/\*\*

\* End of game result notification. Dealer wins, player busts.

\*/

@Override

**public** **void** youBustDealerWinsNotification(**int** wins, **int** losses, **int** ties) {

notifications.setText("You went bust. You lose!");

endRound(wins, losses, ties);

}

/\*\*

\* End of game result notification. Dealer busts, player wins.

\*/

@Override

**public** **void** dealerBustYouWinNotification(**int** wins, **int** losses, **int** ties) {

notifications.setText("The Dealer went bust. You win!");

endRound(wins, losses, ties);

}

/\*\*

\* End of game result notification. Player beats dealer.

\*/

@Override

**public** **void** youBeatDealerNotification(**int** wins, **int** losses, **int** ties) {

notifications.setText("You beat the Dealer, You win!");

endRound(wins, losses, ties);

}

/\*\*

\* End of game result notification. Dealer beats player.

\*/

@Override

**public** **void** dealerBeatsYouNotification(**int** wins, **int** losses, **int** ties) {

notifications.setText("Dealer beat you, You Lose!");

endRound(wins, losses, ties);

}

/\*\*

\* End of game result notification. Dealer and player tie.

\*/

@Override

**public** **void** bothTieNotification(**int** wins, **int** losses, **int** ties) {

notifications.setText("You and the Dealer tie!");

endRound(wins, losses, ties);

}

/\*\*

\* End of game result notification. Player quits game and loses.

\*/

@Override

**public** **void** quitGameNotification(**int** wins, **int** losses, **int** ties) {

notifications.setText("You quit, You lose!");

endRound(wins, losses, ties);

}

}

***BlackjackModel Class:***

**package** unit6.blackjackProject;

/\*\*

\* Abstractly models a Blackjack game.

\* **@author** Ozaner Hansha

\*/

**public** **class** BlackjackModel {

/\*\*

\* The BlackjackView corresponding to this model.

\*/

**private** BlackjackView bv;

/\*\*

\* The player object.

\*/

**private** Player player;

/\*\*

\* The dealer object.

\*/

**private** Dealer dealer;

/\*\*

\* The wins, losses, or ties of this game.

\*/

**private** **int** wins, losses, ties;

/\*\*

\* Whether or not the current round has ended.

\*/

**private** **boolean** gameInProgress;

/\*\*

\* Create a BlackjackModel (given a BlackjackView for notifications).

\* **@param** blackjackView

\*/

**public** BlackjackModel(BlackjackView blackjackView)

{

bv = blackjackView;

}

/\*\*

\* **@return** Amount of cards in {@link #player}'s hand.

\*/

**public** **int** getPlayerCardNumber()

{

**return** player.getHand().size();

}

/\*\*

\* **@return** Amount of cards in {@link #dealer}'s hand.

\*/

**public** **int** getDealerCardNumber()

{

**return** dealer.getHand().size();

}

/\*\*

\* **@return** Whether or not a game is in progress.

\*/

**public** **boolean** isGameInProgress()

{

**return** gameInProgress;

}

/\*\*

\* Starts a new Round by dealing 2 cards, one face down,

\* one face up, to both the player and the dealer.

\*/

**public** **void** newRound()

{

gameInProgress = **true**;

dealer = **new** Dealer();

player = **new** Player();

bv.cardDealtToPlayerNotification(dealer.dealFaceUp(player));

bv.cardDealtToDealerNotification(dealer.dealFaceUp(dealer));

bv.cardDealtToPlayerNotification(dealer.dealFaceUp(player));

bv.cardDealtToDealerNotification(dealer.dealFaceDown(dealer));

}

/\*\*

\* Hits the player with one face up card.

\*/

**public** **void** hit()

{

**if**(player.handValue() < 21)

bv.cardDealtToPlayerNotification(dealer.dealFaceUp(player));

}

/\*\*

\* Does dealer's turn and decides who wins.

\*/

**public** **void** stay()

{

**while**(dealer.handValue() < 17) //if hand value is 17 or more, stay.

bv.cardDealtToDealerNotification(dealer.dealFaceDown(dealer));

checkWinConditions();

}

/\*\*

\* Checks if player or dealer has gone bust and

\* sends appropriate callback to the {@link #bv}.

\* **@return** Whether or not this check has resulted in a winner/tie.

\*/

**public** **boolean** checkForBust()

{

**if**(player.handValue() > 21 && dealer.handValue() > 21) //If both went bust.

{

ties++;

bv.bothBustNotification(wins, losses, ties);

**return** **true**;

}

**else** **if**(player.handValue() > 21) //If player went bust.

{

losses++;

bv.youBustDealerWinsNotification(wins, losses, ties);

**return** **true**;

}

**else** **if**(dealer.handValue() > 21) //If dealer went bust.

{

wins++;

bv.dealerBustYouWinNotification(wins, losses, ties);

**return** **true**;

}

**return** **false**;

}

/\*\*

\* Checks to see if any player has a blackjack.

\* Whoever does wins, if both have one its a tie.

\* **@return** Whether or not this check has resulted in a winner/tie.

\*/

**public** **boolean** checkForBlackjack()

{

**if**(player.hasBlackjack() && dealer.hasBlackjack()) //If both got blackjack as well.

{

ties++;

bv.bothTieNotification(wins, losses, ties);

**return** **true**;

}

**else** **if**(player.hasBlackjack()) //If player has a blackjack.

{

wins++;

bv.youBeatDealerNotification(wins, losses, ties);

**return** **true**;

}

**else** **if**(dealer.hasBlackjack()) //If dealer has a blackjack.

{

losses++;

bv.dealerBeatsYouNotification(wins, losses, ties);

**return** **true**;

}

**return** **false**;

}

/\*\*

\* Checks who has a bigger hand. Always returns a winner.

\* Sends appropriate callback to the {@link #bv}.

\*/

**public** **void** checkForWin()

{

**if**(player.handValue() > dealer.handValue()) //If player is over dealer (checkForBust insures not a bust)

{

wins++;

bv.youBeatDealerNotification(wins, losses, ties);

}

**else** **if**(dealer.handValue() > player.handValue()) //If dealer is over player

{

losses++;

bv.dealerBeatsYouNotification(wins, losses, ties);

}

**else** //Must be a tie

{

ties++;

bv.bothTieNotification(wins, losses, ties);

}

}

/\*\*

\* Ends the round and finds a winner.

\*/

**public** **void** checkWinConditions()

{

**if**(!checkForBust()) //If nobody bust.

**if**(!checkForBlackjack()) //If nobody has a blackjack.

checkForWin(); //Always returns a winner.

gameInProgress = **false**;

}

/\*\*

\* This round is a loss and starts a new round.

\*/

**public** **void** quitGame()

{

losses++;

gameInProgress = **false**;

bv.quitGameNotification(wins, losses, ties);

}

}

***BlackjackGCard Class:***

**package** unit6.blackjackProject;

/\*\*

\* Models a blackjack card. The value of a blackjack card is computed.

\* **@author** Mark Jones

\*/

@SuppressWarnings("serial")

**public** **class** BlackjackGCard **extends** GCard {

/\*\*

\* The highest possible value of an ace.

\*/

**public** **static** **final** **int** ***HIGH\_ACE\_VALUE*** = 11;

/\*\*

\* The lowest possible value for an ace.

\*/

**public** **static** **final** **int** ***LOW\_ACE\_VALUE*** = 1;

/\*\*

\* The value of all face cards (jack, queen, king)

\*/

**public** **static** **final** **int** ***FACE\_VALUE*** = 10;

/\*\*

\* Creates a new blackjack card.

\* **@param** r - The rank of the card.

\* **@param** s - The suit of the card.

\*/

**public** BlackjackGCard(Rank r, Suit s) {

**super**(r, s);

}

/\*\*

\* Convenience method for making a deck of BlackjackGCards.

\* **@return** the new deck

\*/

**public** **static** Deck makeDeck() {

Deck deck = **new** Deck();

**for** (Suit s : Suit.*values*()) {

**for** (Rank r : Rank.*values*()) {

deck.add((Card) **new** BlackjackGCard(r, s));

}

}

**return** deck;

}

/\*\*

\* The value of a Blackjack card, counting an ACE as 1.

\*/

**public** **int** value() {

String rank = getRank().toString();

**if**(rank.matches("j|q|k")) **return** ***FACE\_VALUE***;

**else** **if**(rank.equals("a")) **return** ***LOW\_ACE\_VALUE***;

**else** **return** Integer.*parseInt*(rank);

}

}

***BlackjackHand Class:***

**package** unit6.blackjackProject;

/\*\*

\* A BlackjackHand is a Hand with BlackjackCards in it.

\* This is simpler than parameterizing Hand<BlackjackCard>.

\* A BlackjackHand knows how to value itself, determine if

\* a hand is a blackjack, etc.

\*

\* **@author** Dr. Mark A. Jones

\*/

@SuppressWarnings("serial")

**public** **class** BlackjackHand **extends** Hand {

**public** **static** **int** *BLACKJACK\_VALUE* = 21;

/\*\*

\* Creates a BlackjackHand.

\*/

**public** BlackjackHand() {

**super**();

}

/\*\*

\* Returns the highest legal value for a blackjack hand.

\* **@return**

\*/

**public** **int** handValue()

{

**int** value = 0;

**int** highAces = 0;

//Adds values of the entire hand.

**for**(Card g: **this**)

{

//If card is an ace add its high value, else use normal value.

**if**(g.getRank().toString() == "a")

{

value += BlackjackGCard.***HIGH\_ACE\_VALUE***;

highAces++;

}

**else**

value += ((BlackjackGCard)g).value();

}

//If the value is over 21 and there are still high aces.

//Set those high aces to low ones.

**while**(value > 21 && highAces > 0)

{

value += BlackjackGCard.***LOW\_ACE\_VALUE*** - BlackjackGCard.***HIGH\_ACE\_VALUE***;

highAces--;

}

**return** value;

}

/\*\*

\* Determines whether a hand is a "blackjack" (a two-card hand worth 21).

\* **@return** true if a blackjack hand, false otherwise

\*/

**public** **boolean** isBlackjack() {

**return** size() == 2 && handValue() == *BLACKJACK\_VALUE*;

}

}