Jinghu Lei

U84972785

Trianta Ena Object Specification

**Game Logic Classes:**

Game:

* Extends: Object (None)
* Implements: Comparator<Hand>
* Important Methods:

public Game(int maxVal, int dealVal, int maxPlayers)

* + The main constructor for the game class, which initializes a max value (ex. 21/31), the target deal value (ex. 17/27), and the max number of players.

public static void select()

* The entry point into the games. It asks the user to select the game to play, for which the game class will they monitor.

public int initialize()

* Initializes the game parameters and selects the number of players through input, which is then passed into createPlayers(…).

public void createPlayers(…)

* Generates the player list and dealer/computer dynamically and if more than one player randomly assigns the dealer. Then it initializes the object fields with the correct starting values.

public void play() throws EndGameException

* The continuous loop that counts and starts new rounds as long as players want to continue or the players are not all broke.

public abstract boolean startRound() throws EndGameException;

* Encapsulates the entire logic for one round, up to the game to implement. Can throw an EndGameException to stop the game if the user enters “quit”

public abstract void deal();

* Decides how the dealer deals their card for each game. Both games have a deal phase before the round actually begins.

public void playerTurn() throws EndGameException

* The entire player turn loop, which tracks input and calls the checkActionPlayerHand for each input made.

public void dealerTurn() throws EndGameException

* The entire dealer turn loop. It is automatic if a computer, or can take input if the dealer is a human (although it still forces to hit up to at least 27).

public abstract void checkAction(Player/Dealer)Hand(Player player, Hand hand) throws EndGameException

* Checks the action of the player or dealer hand (2 separate methods) and performs the correct action based on a switch statement. Must be implemented for each game because the allowed moves are different.

public void checkResult()

* Checks the result of the round and calls the win/tie/lose methods as necessary.
* Changes: To add Trianta Ena while keeping most of my methods intact, I generalized my Blackjack methods and inserted them into the Game class. The Game class then became the parent for both Blackjack and Trianta Ena since they have very similar actions and objects. The game class then provides all of the general game logic that is shared between both of the two such as players, dealers, checking if a hand is over a max, etc. The 90% of the methods that the *Blackjack* and *Trianta Ena* classes need to implement are the abstract methods in this parent class and some methods they need to override with more functionality.

Blackjack:

* Extends: Game
* Implements: None
* Important Methods:

public static void start()

* + Starts by initializing the game variables for Hand (e.g. the max hand value) and updates its singleton instance of the game to a new *Blackjack* object. Then it creates the players and calls play. It acts as the entry point into the game after the *select* method from the Game class. Acts as the top most method that all EndGameExceptions for this game propagate to. It handles the exceptions by calling the exit() once one is caught.

public static void exit()

public void printPlayerControls()

public void printDealerControls()

public void play() throws EndGameException

public void deal()

public boolean startRound() throws EndGameException

public void checkAction(Player/Dealer)Hand(Player player, Hand hand) throws EndGameException

…

* Changes: I abstracted most of the *Blackjack* methods out so they can fit into the parent *Game* class. Instead of having hard coded 21 values, it turned into a variable MAX\_VAL that can be set. The only two completely new methods from the Game class are the *start* and *exit* methods. These control the start and ending of the complete game.

Trianta Ena (*New*):

* Extends: Game
* Implements:
* Important Methods:

public static void start()

* + Starts by initializing the game variables for Hand (e.g. the max hand value) and updates its singleton instance of the game to a new *Blackjack* object. Then it creates the players and calls play. It acts as the entry point into the game after the *select* method from the Game class. Acts as the top most method that all EndGameExceptions for this game propagate to. It handles the exceptions by calling the exit() once one is caught.

public static void exit()

public void printPlayerControls()

public void printDealerControls()

public void play() throws EndGameException

public void deal()

public boolean startRound() throws EndGameException

public void checkAction(Player/Dealer)Hand(Player player, Hand hand) throws EndGameException

public boolean oneWinner()

public void printRanking()

public void convertBanker(ArrayList<CardPlayer> ranking)

EndGameException (*New*):

* Extends: Game
* Implements:
* Important Methods:

**Player Classes:**

CardPlayer:

* Extends: Game
* Implements:
* Important Methods:
* Changes:

Player:

* Extends: Game
* Implements:
* Important Methods:
* Changes:

Dealer:

* Extends: Game
* Implements:
* Important Methods:
* Changes:

Computer:

* Extends: Game
* Implements:
* Important Methods:
* Changes:

**Card Classes:**

Deck:

* Extends: Game
* Implements:
* Important Methods:
* Changes:

Hand:

* Extends: Game
* Implements:
* Important Methods:

Card:

* Extends: Game
* Implements:
* Important Methods:

**Main Method Class:**

App:

* Parent Class: Object (None)
* Interfaces: None
* Purpose:

As part of the Maven file architecture, the main method should be contained in a separate App.java file. This separates the entry point of the application from the rest of the source files.

* Important Methods:

public static void main(String[] args)

* Calls the static Game.start() method, which initiates an user input sequence for deciding what game (currently Blackjack or Trianta Ena) that the user wants to play.