**MiniProject2 Report**

**Developing a Gambling Problem**

Nathaniel Foster

Department of Engineering, Johns Hopkins University

EN.605.201.84.SP20: Intro to Programming Using Java

Dr. Jamal Simms

April 5, 2020

General Program Design:

The ***BlackjackGameSimulator*** relies on several classes to simulate games of Blackjack until it receives an input to end the program. The following assumptions are leveraged for this implementation of a Blackjack game simulator: a new deck is used for each round; only one deck of card is used at a time; the dealer will stay at a soft 17; and a card is never re-used. The ***BlackjackGameSimulator*** class handles all inputs and outputs to the user via the console/terminal. Based on user inputs, it creates an instance of the ***BlackjackGame*** class.

The ***BlackjackGame*** class is responsible for defining storage of the components of a Blackjack game and facilitating interactions between these components. A Blackjack game has a deck of cards and a list participants which includes the dealer and player.

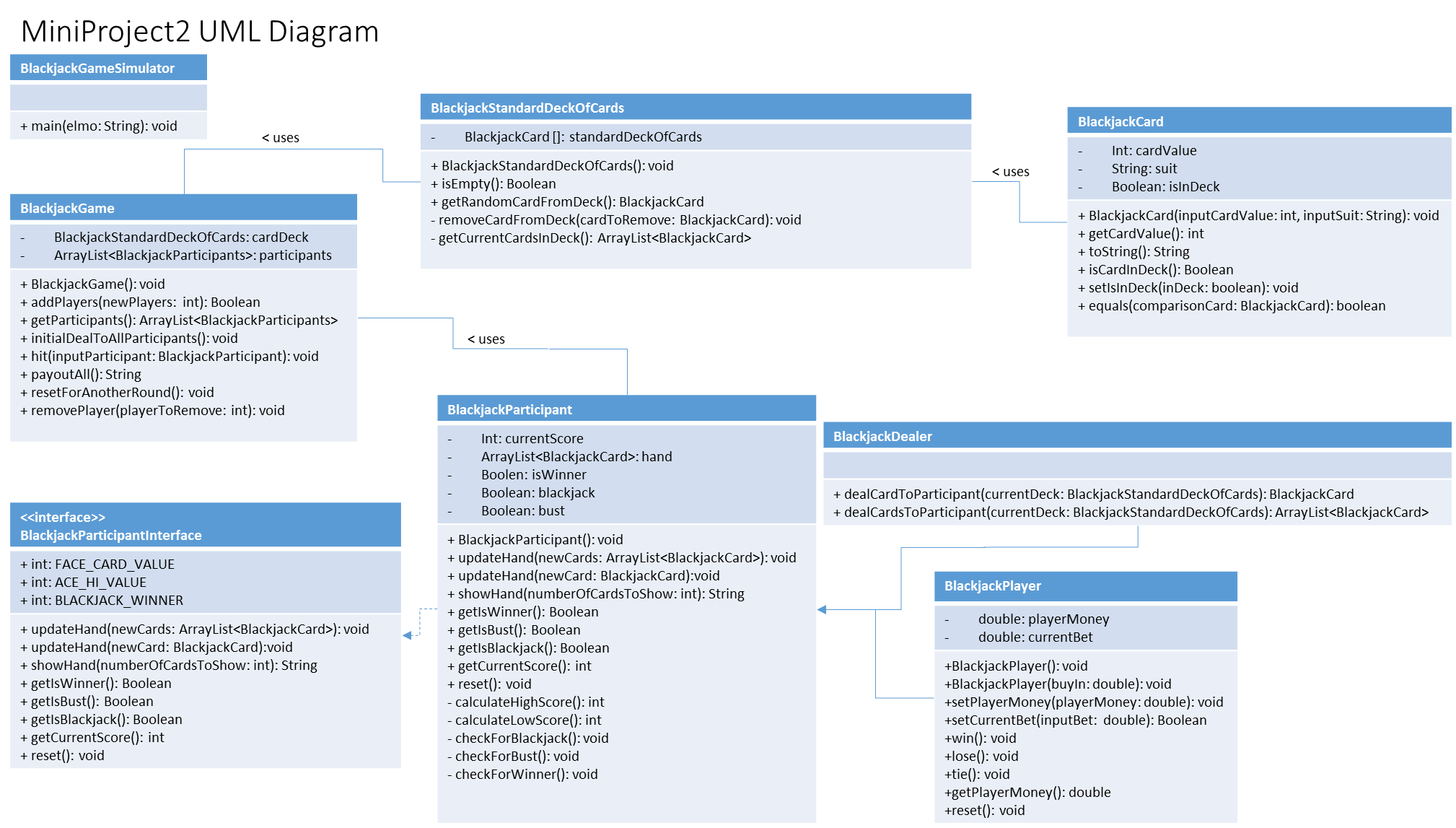
The dealer and players are maintained in a common ***ArrayList*** instance of type ***BlackjackParticipant***. The ***BlackjackParticipant*** class is the parent class to both the ***BlackjackDealer*** and ***BlackjackPlayer*** classes. These classes define attribute and behaviors specific to being a player or dealer. Dealers are able to deal cards if given a deck of cards. Players are able to make bets as well as win and lose money. ***BlackjackParticipant*** is an abstract class implementing the ***BlackjackParticipantInterface***. It defines all the common attributes and behaviors for updating the attributes. All participants have a current score, a hand of cards, a flag for being a winner, a flag for being a bust and a flag for Blackjack. The dealer is always the first positon in the list since a single dealer must exist in every Blackjack game. The rest of the participants are assumed to be players. This greatly simplified implementation as the dealer could easily be included or excluded from processing by just skipping the first position in the list.

For this implementation, only one deck of cards is stored in a blackjack game since only one deck of cards is used at a time. A deck of cards is defined by the ***BlackjackStandardDeckOfCards*** class. An instance of the ***BlackjackStandardDeckOfCards*** is made of an array of 52 cards. A card is defined by the by the ***BlackjackCard*** class. An instance of the ***BlackjackCard*** class consists of the card value (int: 1 - 13), the suit of the card (String: “Clubs”, “Diamonds”, “Spades”, “Hearts”), and a status of whether or not it is in a deck (boolean: true, false). By default, a card is assumed to be part of a deck. A separate card class was created as it greatly simplified the implementation of the program in lieu of using a multidimensional array to store all the card attributes.

Approach:

An iterative approach was taken to the development of the program that built up to the fully designed program. The first iteration was designed to only support without even the dealer playing using a single deck of cards. The next version then supported one player and one dealer playing against each other still with one deck of cards. This next feature added was the ability to play with more than one player. The number of players at first was limited to 25 to avoid running out of cards during the initial dealing of cards. This limitation was later removed by simply using a new deck of cards once the entire deck was distributed. This design however still fell far short of the specifications called out by the project. The next step was to allow for multiple games to be played and for players to have the option to quit the game or be removed if he or she is out of money. This was first done in a rudimentary way that created a whole new game in the event that the user wanted to player another game. This approach required the user to re-enter all player information. This information included the number of players and starting amounts of money for each player. Depending on the number of players, entering all this information again can be quite a hassle so this approach was still considered unacceptable albeit one step closer to the final design. A routine was added at the end of each round to iterate through all the players if they have any money left and prompt each one for a choice to quit or continue playing. Players that ran out of money or chose to quite were removed from the game. If the players want to play another game, their data can be saved for the next game or a whole new game can still be created to add new players. If all the players quit, the

UML Diagram:



\*BlackjackParticipant implements BlackjackParticipantInterface – dashed line may be hard to see

Citations/References:

1. Java, Ninth Edition

Herbert Schildt - 2014