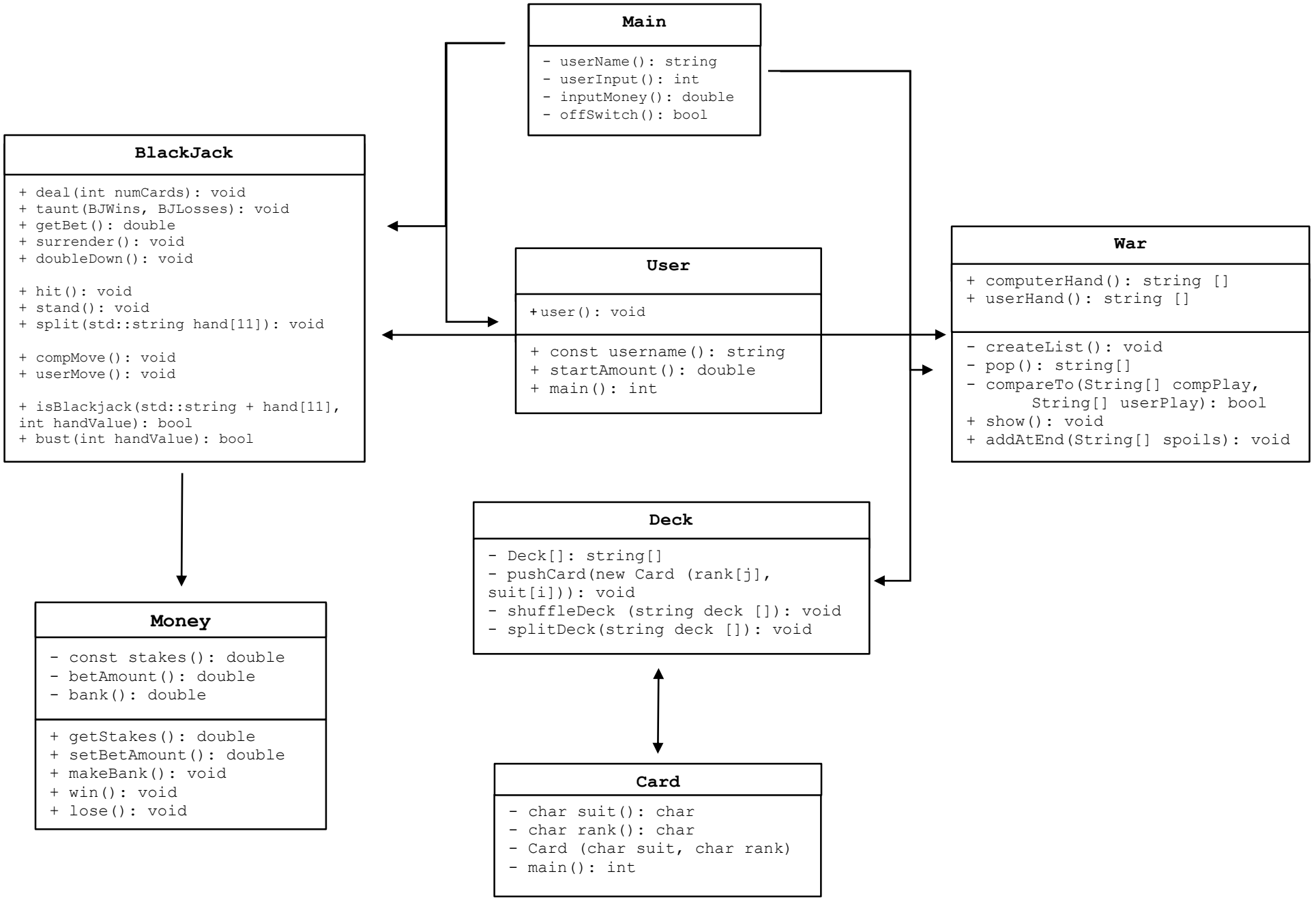
Gambling Program – Card Games



Gambling Program – Card Games:

The purpose of our project is to implement typical card games into a program that represent their real-world counterpart. The two games available are Blackjack and War where the user has to pay in order to play.

==========

Curtis Hiscock

Main

==========

```
Main()
-------------------------------------------------------
-  userName() : string
-  userInput() : int
-  inputMoney() : double
-  offSwitch() : bool
```

Notes:

I'm leaving out the arrows to cut down on formatting problems. Main should instantiate player object at the beginning of the program with name and money. Should call on the player object to display name and money. Instantiates each game object to begin the games. When we put together our UML, main should have arrows pointing to our player class and each game class.

The main method:

Prompts user to input a player name(): string and starting money(): double, then instantiates a person object with these two values. Userinput(): int is used to choose options with in the menu, and offSwitch(): bool is a switch that should trigger when prompted by the user to exit.

Within a while loop it displays a menu, with player name in the top left and remaining money in the top right. These two values are called from a "player" object. This menu displays 1) Play Blackjack, 2) Play War, 3) Exit.

Calling War and Blackjack should begin a game of war or blackjack, defined in class methods for War() and Blackjack().

Logan Peck

User

```
+ user(): void
-----------------------------------------------
+ const username: string
+ startAmount: double
-----------------------------------------------
+ main(): int
```

Notes:

The UML diagram above displays the user() class defined for our card game extravaganza user interface. The user of this program will be playing against the program or the machine. This class will be used to store the username, or "name", of the individual user that will be called in each of the two game classes specified "blackjack" and "war." The userName will be stored as a public const string; this is because the user will define this at the beginning of the program and will be unable to change it afterwards.

The way both of these games will work: users will be able to place "bets" during each round of gameplay. The "house" in the casino, taken place by the CPU or program against which the user is playing will have an amount of money to start as declared in the money() class and defined in the respective game class. The startAmount will be the amount of money that the user starts with when the program begins. In the main(), the user will be prompted to enter an amount of money that they will be bringing to the table to begin. That amount will then be set in the money() class.

==========

Rebecca Loredo

Deck and Card

==========

```
Deck()
-------------------------------------------------------
- Deck[] : string[]
- pushCard(new Card (rank[j], suit[i])): void
- shuffleDeck (string deck []): void
- splitDeck(string deck []): void
-------------------------------------------------------
Card()
-------------------------------------------------------
-  char suit;: char
- char rank: char
- Card (char suit, char rank)
- main(): int
```

Notes:

This will be the Deck class that will encapsulate a Card subclass.

The card subclass will handle the creation of each individual card within for loops, to be pushed into the Deck array. This will ensure that there is no card doubled. The deck class will then be able to shuffle the deck. The split method will split the array for the War class.

==========

Katie Harrington

Blackjack, feeling lucky???

==========

```
Blackjack()
-------------------------------------------------------
+ deal(int numCards): void
+ taunt(BJWins, BJLosses): void
+ getBet(): double
```

```
+ surrender(): void
+ doubleDown(): void

+ hit(): void
+ stand(): void
+ split(std::string hand[11]): void

+ compMove(): void
+ userMove(): void

+ isBlackjack(std::string hand[11], int handValue): bool
+ bust(int handValue): bool
```

Notes:

Both the computer and the user will have a hand that starts with 2 cards and the rest as NULL entries in a size 11 array (the largest hand that can stay under 21 is 11 cards with 4 aces, 4 twos, and 3 threes). Then play begins. The computer/dealer's moves are automated. The user will be able to choose through the userMove() method to hit, stand or split their hand. On the first turn, they can also choose to surrender or double down, giving up or doubling their bet. Then, as the rounds continue, the round counter keeps track of how many rounds there has been (so the user can't surrender or double down after the first turn and also for taunts). Then when someone either gets 21, blackjack, or busts, the game ends and the win/loss counters are updated and the win will be sent to the money class.

Also, based on the win/loss ratio, the program will congratulate/taunt the user each time a new game of blackjack is started.

==========

Edan Canning

$$$Money Class$$$

==========

```
Money()
-----------------------------------------------
- const stakes(): double
- betAmount(): double
- bank(): double
-----------------------------------------------
+ getStakes(): double
+ setBetAmount(): double
+ doubleBet(): void
+ win(): void
+ lose(): void
```

Notes:

The variables included are stakes: random double set at the start of program by constructor represents stakes of the game, betAmount: taken from the user and will determine loss or payout (relative to stakes), bank: user's total money in the bank.

Functions include getStakes(): returns the instance's stakes, setBetAmount(): sets the amount the user wants to bet, win(): gives the user his payout (in to his bank) relative to stakes. lose(): removes the user's betAmount (from bank), doubleBet(): doubles the user's bet.

==========

Stephen Slogar/Elba Chimilio

War

==========

```
War()
-------------------------------------------------------------------
+ computerHand(): string [ ]
+ userHand(): string [ ]
-------------------------------------------------------------------
- createList(): void
- pop(): string [ ]
- compareTo(String [ ] compPlay, String [ ] userPlay): bool
+ show(): void
+ addAtEnd(String [ ] spoils): void
```

Notes:

The War() class will take in a deck of cards from the deck class. A computerHand() and userHand() array will each be created from a deck split in half. The createList() will create a head node for a linked list. A for loop will push each item to a computerHand() linked list and a userHand() linked list from their respective arrays.

The pop() method takes the first 4 nodes off of the hand of cards and returns them in an array. The method compareTo() checks the last index of both arrays and compares the number to determine a winner. A while loop keeps the method running as long as the number of the two cards is the same. A spoils array stores all the cards while this loop is running. The show() method shows the user both cards and tells the user whether they won or lost. The addAtEnd() method takes the spoils array and pushes all of its elements onto the end of the winner's hand.