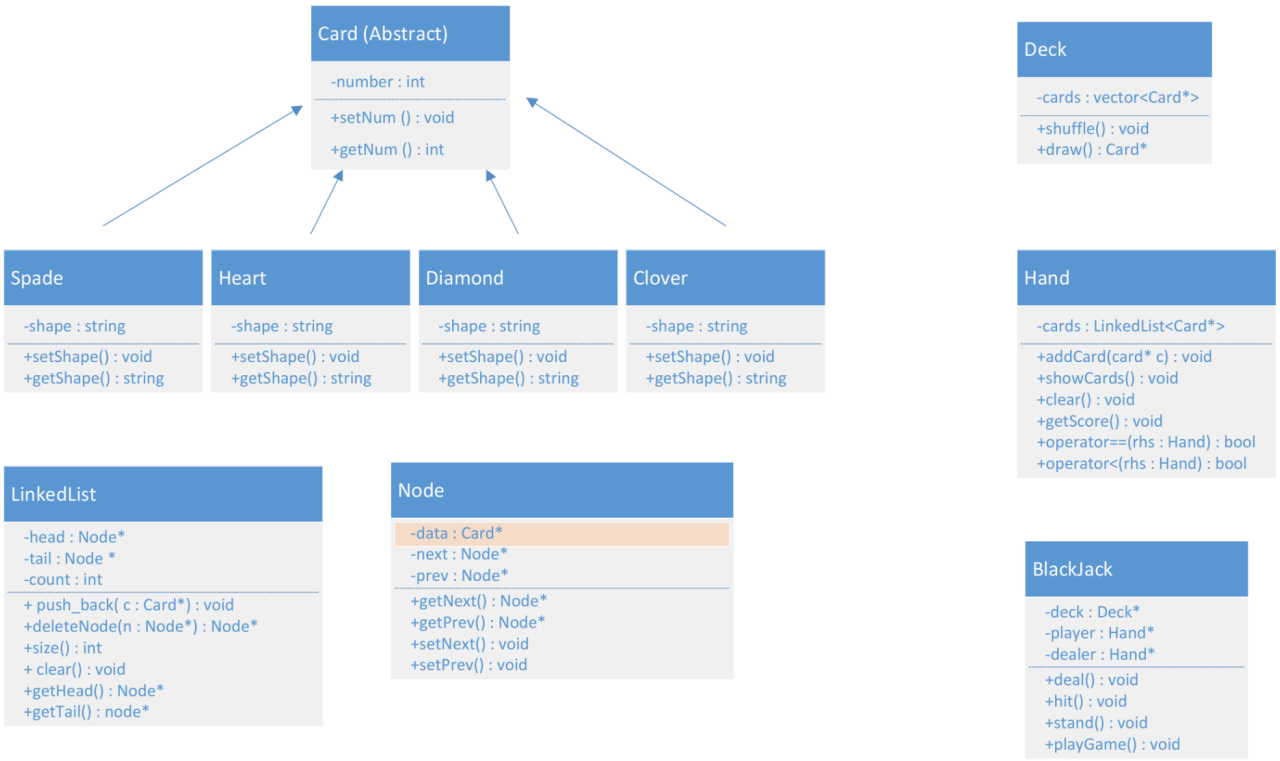
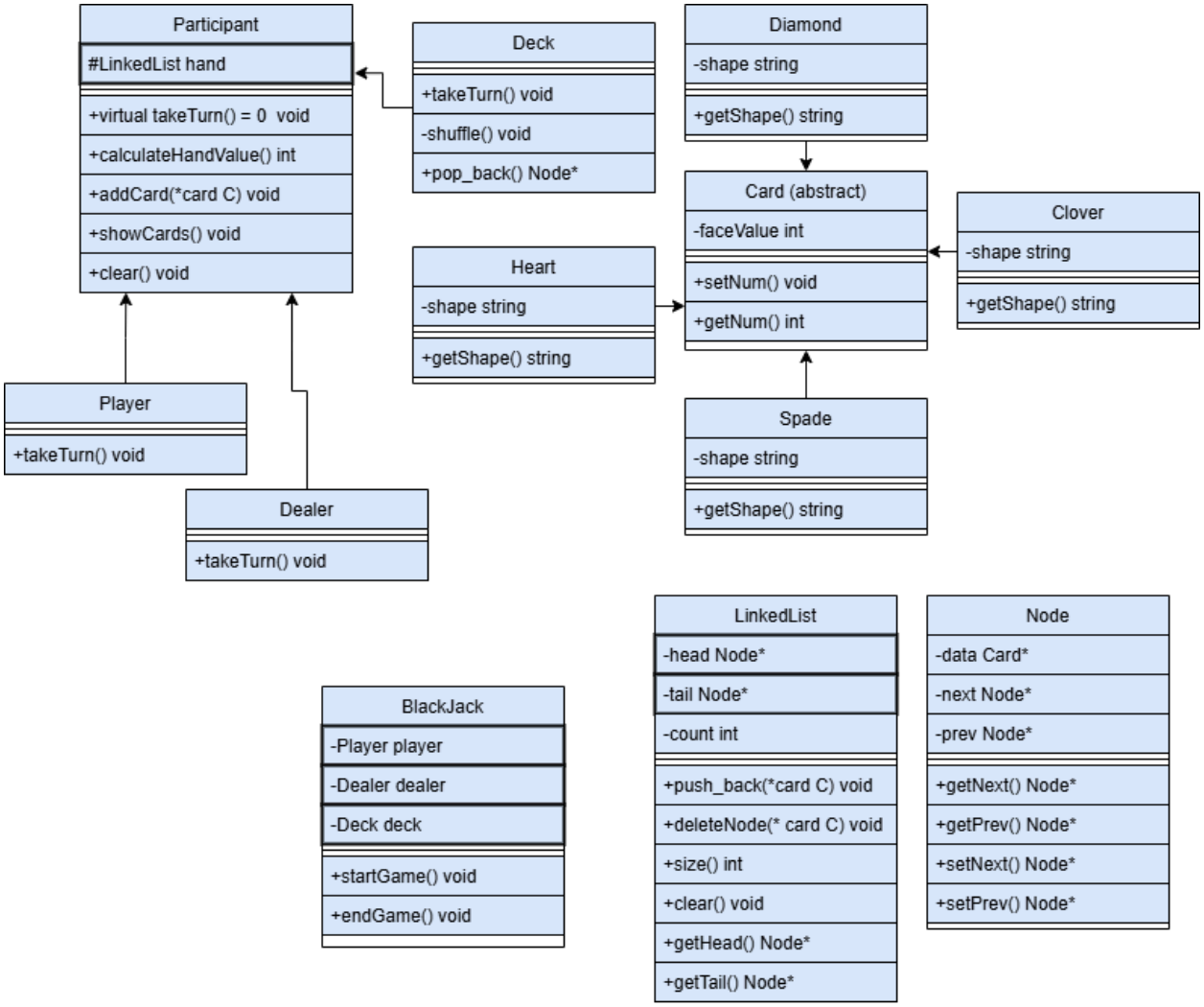
Summary of Project Goals

Implement a simple game of Blackjack with a deck of 52 cards played by one user. Use linkedlists to store and shuffle cards. Use Card objects to represent cards and their face values. Use OOP design to combine multiple objects together into a cohesive program

UML Diagram (old)



New



Meeting Project Guidelines

- *Use of algorithms and data structures*

Cards held by the dealer, player, and the card deck in play are stored and accessed from a linkedlist of type Card, making use of the linkedlist data structure

Cards in the player's hand can be sorted for display purposes, making use of sorting algorithms. Cards in the deck will have to be shuffled during gameplay, which will also use its own algorithm

- *OOP Design*

Cards held by the dealer and player are stored in a linkedlist of Card objects. This makes use of polymorphism to store suite cards derived from the Card base class.

The game itself is managed by an object of the BlackJack class which will control game flow, creating and handling the hands of the dealer and player.

Players and dealers are derived from a Participant class. Participants have a hand (a linkedlist of cards) and different methods to interact with their hand. Derived classes of Player and Dealer will override these methods to play different roles during gameplay

Stages of Development

Implement the basics of each class

- Data members, setters/getters, function prototypes

Implement LinkedList and Card and test storing cards of different values and suites into the same Linkedlist

Implement and test a sorting algorithm and a shuffle algorithm for the LinkedList of cards

- Check that sorting actually places nodes in order of data
- Check that sorting doesn't break the list (all nodes connect properly)
    - Check that sorting doesn't somehow remove or add nodes
- Check that shuffling actually changes the order of nodes
- Check that shuffling doesn't break the list
    - Check that shuffling doesn't somehow remove or add nodes

Implement player and dealer methods like:

Implement the calculateHandValue() method and test its logic

- Correctly identifying when the player or dealer is over 21
- Correctly identifying when the player or dealer is at exactly 21
    - Correctly identifying when the player or dealer is at exactly 21 at the start of the game
- Optionally, correctly and dynamically updating the face value of aces in the player/dealer's hand

Implement the addHand() method

- Correctly gives caller a random card with a face value within range and an appropriate suite
- Correctly adds the card to the player/dealer's linkedlist

Implement part of the BlackJack class to start the game

- Check that player and dealer are created properly
- Check that player and dealer have a functioning Linkedlist
    - Check that player and dealer have the right amount of starting cards
- Test different textual prompts to direct gameplay

- Optionally, allow reading into a file to load high scores and specific configurations (number of cards in deck, dealer behaviour)

Implement part of the BlackJack class that ends the game

- Provide text prompts/directions on exit
- Optionally, load highest score and specific configurations into text file