Project 6 Implementation Notes

LinkedStack and LinkedQueue

- The toString() implementation was interesting. I'm using an infinite while loop, to traverse the stacks and queues. Once the loop reaches the bottom/tail, this line throws an NullPointerException, which exits the loop:

  ```
  node = node.getNext();
  ```

  I finish assembling the String in the finally block and return it there. Eclipse doesn't like that I have a return inside the finally block, and I understand why, but for this implementation it works nicely.

Project6

- I created a LinkedQueue of LinkedStacks of Strings to test these constructs.

OUTPUT

```
Beginning contents of queue of stacks:
[head][top]C, B, A[bottom], [top]R, Q, P[bottom], [top]Z, Y, X[bottom][tail]

      Popping stack at front of queue: C
      Now at the top of the front of the queue: B
      Moving the front to the back of the queue...

Ending contents of queue of stacks:
[head][top]R, Q, P[bottom], [top]Z, Y, X[bottom], [top]B, A[bottom][tail]
```

StackHand

- StackHand is a LinkedStack of PlayingCards.
- It implements the HandOfCards interface.

BeatDealer

- The game instantiates a Deck of PlayingCards and uses the Deck methods to randomly deal cards. Deck does not mind dealing the same card to many players or even the same player many times, though.
- Players and a dealer are instantiated, each of which has its own StackHand.
- Player methods are used to add and play cards from each Player's hand.
- I setup the game to have 5 players with each having 3 cards. This is easily configurable.
- The players are enqueued to a LinkedQueue and repeatedly dequeued and enqueued again after playing their hands in each round.
- I added a getValue() method to the PlayingCard class which returns an int. This method translates ranks A, K, J and Q to a value of 10 and parses the int value from the String of other ranks.
- I wrapped the playing of each hand into a playHand() method, and wrapped the whole thing in a try block, in case the dealer or players run out of cards.

OUTPUT

```
New Hand Starting.
	Dealer's card: 2 of Clubs
	Dealer's total: 10

	Players cards:
		Player 1: 2 of Spades
		Player 2: 6 of Spades
		Player 3: King of Diamonds
		Player 4: 10 of Hearts
		Player 5: 2 of Hearts
	Players total: 30


	Players Win!


New Hand Starting.
	Dealer's card: Queen of Diamonds
	Dealer's total: 50

	Players cards:
		Player 1: 5 of Clubs
		Player 2: Ace of Spades
		Player 3: 2 of Hearts
		Player 4: 4 of Spades
		Player 5: Queen of Hearts
	Players total: 32
```

```
       Dealer Wins!


New Hand Starting.
       Dealer's card: 5 of Hearts
       Dealer's total: 25

       Players cards:
              Player 1: 2 of Hearts
              Player 2: 10 of Clubs
              Player 3: 6 of Spades
              Player 4: 3 of Diamonds
              Player 5: Queen of Diamonds
       Players total: 31


       Players Win!

The dealer won 1 of 3 hands.
The players won 2 of 3 hands.

Thanks for playing.
```

BeatDealer.java

```
import cards.Deck;
import cards.PlayingCard;
import jsjf.LinkedQueue;
import jsjf.exceptions.EmptyCollectionException;

public class BeatDealer {
	static LinkedQueue<Player> players;
	static Deck deck = new Deck(52);
	static Player dealer = new Player();
	static int numPlayers = 5;
	static int handSize = 3;
	static int playerWins = 0;
	static int dealerWins = 0;

	public static void main(String[] args) {
		players = new LinkedQueue<Player>();
		setupGame();
		for (int j=1; j<=handSize; j++) {
			playHand();
		}
		System.out.println("\nThe dealer won " + dealerWins + " of " + handSize + "
			hands.");
		System.out.println("The players won " + playerWins + " of " + handSize + "
			hands.");
		System.out.println("\nThanks for playing.");
		}

	public static void setupGame() {
		//creates 10 players with handSize cards each and enqueues them into the players
		//queue
		for (int i=1 ; i<=numPlayers; i++) {
			Player player = new Player();
			for (int j=1; j<=handSize; j++) {
				player.receiveCard(deck.dealOne());
			}
			players.enqueue(player);
		}

		//gives the dealer handSize cards
		for (int j=1; j<=handSize; j++) {
			dealer.receiveCard(deck.dealOne());
		}
	}

	public static void playHand() {
		int playersHandTotal = 0;
		int dealersHandTotal = 0;
		int diff;
		Player activePlayer;
		PlayingCard activeCard;

		System.out.println("\n\nNew Hand Starting.");

		try {

			/*calculates the dealer's hand
			 * wrapping in try block in case a player or dealer does not have a card
			 * to play
			 */
```

```
                        activeCard = dealer.playCard();
                        dealersHandTotal = activeCard.getValue()*numPlayers;
                        System.out.println("\tDealer's card: " +  activeCard.getRank() + " of "
                                + activeCard.getSuit());
                        System.out.println("\tDealer's total: " + dealersHandTotal);


                        /*calculates sum of players cards
                         * player is dequeued and enqueued again after showing hand
                         */

                        System.out.println("\n\tPlayers cards:");
                        for (int i=1; i<=numPlayers; i++) {
                                activePlayer = players.dequeue();
                                activeCard = activePlayer.playCard();
                                System.out.println("\t\tPlayer " + i + ": " +
                                        activeCard.getRank() + " of " + activeCard.getSuit());
                                playersHandTotal += activeCard.getValue();
                                players.enqueue(activePlayer);
                        }

                        System.out.println("\tPlayers total: " + playersHandTotal + "\n");



                        //determines who won

                        diff = dealersHandTotal - playersHandTotal;

                        if (diff >= 0) {
                                System.out.println("\n\tDealer Wins!");
                                dealerWins++;
                        }
                        else {
                                System.out.println("\n\tPlayers Win!");
                                playerWins++;
                        }
                }
                catch (EmptyCollectionException e) {
                        System.out.println("\n*****We ran out of cards!*****");
                }
                catch (Exception e) {
                        System.out.println("\n*****Something went wrong.*****");
                }
        }
}
```

Class Diagram