Two locations where you implement defensive programming

- Card.java
  - Assert statement checking the string is not equal to null in the card constructor
  - Assert statement checking that the card is both of an existing suit and rank
- Stack.java
  - Assert statement checking that input string is not null in the stack constructor
- Throughout our code we are throwing exceptions to check for different cases in which our program might fail due to unexpected inputs. For example if the top for the stack is less than zero then the code will throw an exception saying that it is not possible to pop card from an empty stack.

Two locations where you might have corrected code smells

- SimpleServer.java -- long method
  - Our server class is fairly long, despite the name indicating that it is simple.
- Card Util -- Primitive Obsession

Locations that are responsible for inter-processes/threads communication

- In the simple server class we have a main thread which accepts connection from the incoming clients and then creates Clienthandler threads which takes care of the incoming client connections.

Three locations where you implement three design patterns

- MVC pattern - MainView and HandView are the views in the Model View Controller pattern. The Card, Player,Stack and GameState together make up the Model for our program. The Player Controller communicates with the GameState and the Player then updates the view.
- Singleton Pattern - A single class(SingletonSocket.java) creates an object and ensures that only one object is created, in this case a socket object.

- Composite Pattern - Our GameState class follows a composite pattern. It is composed of the Player, Stack and the Card class. Every game state has one stack , three players and one card. The stack in turn will have at least one card in it and each player will have ten cards in hand.