# Solitaire Functional Specifications

**Overall Reasoning:**

Java – It’s my strongest language to date, and I feel comfortable writing in it.

Apache Commons Collections 4 Library – I learned about it after a friend was writing some code with it, and I found it more efficient to use than regular TreeMaps.

I/O Design – I wanted to make sure that the user had ease of access with playing the game, but that the game also set strict rules on inputs and outputs. The game is purposefully basic so as to not confuse the player, and directions are provided in concise, but effective language.

**Board**

**Card**

**CaseTest**

**Deck**

**InvalidCardException**

**InvalidMoveException**

**Main**

**Rank**

**RankMap**

**SolitaireTest**

**Suit**

**SuitMap**

**Board Class**

Specifications

The Board class implements CaseTest, allowing it to check for edge cases while the user is inputting commands, and thus controlling the types and amounts of inputs allowed via the console. Board.java uses the data structures ArrayDeque, ArrayList, and Stack, among others, to provide the proper error checking and operation during the course of gameplay.

**Card Class**

Specifications

The Card class simply constructs each card. Given that the deck is parsed from a text file containing a single number and a letter representing the rank and the suit on each line, the Card.java file converts that data into a proper Card object with a suit, a rank, and a color. The Card class makes use of two other classes created – RankMap and SuitMap.

**CaseTest Interface**

Specifications

The CaseTest interface provides direction for Board.java. During the early stages of my project, I only used the CaseTest as a basic guide for Board.java, but as the project progressed, I found it useful in helping outline other portions of code, and a similar framework was used to construct SolitaireTest and lineate the Exceptions.

**Deck Class**

Specifications

Put simply, the Deck class converts the given deck.txt file into a Deck object by essentially organizing the deck as an ArrayList of Card objects. A shuffle algorithm is used to randomly shuffle the cards each time the program is run.

**InvalidCardException Class**

Specifications

The InvalidCardException class was used extensively in the Card.java file, providing a framework to regulate the cards both in the deck and the cards in the game. It extends Java’s Exception.

**InvalidMoveException Class**

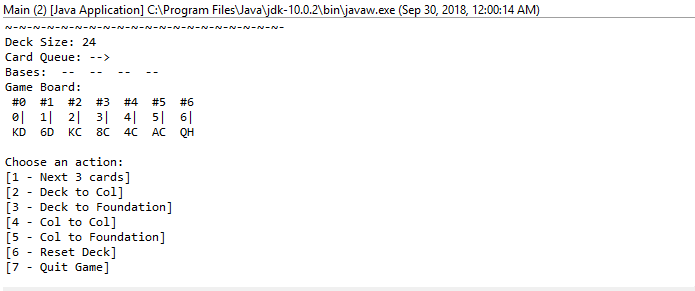
Specifications

The InvalidMoveException class was used mainly in the Board.java class to keep track of and either validate or invalidate certain moves that the player may have tried. It extends Java’s Exception.

**Main Class**

Specifications

The Main class is where all the magic happens. Using a simple switch-case statement to interpret the player’s commands, the Main class provides a simple, easy to use command-line I/O system.



**Rank** **Enum**

Specifications

The Rank Enum simply provides values for all the rank types of a Card, from *ACE* to *KING* and everything in between.

**RankMap Class**

Specifications

The RankMap class makes special use of the Apache Commons Collections 4 BidiMap and the TreeBidiMap in order to map the type Rank object keys to the numerical values.

**SolitaireTest Class**

Specifications

The SolitaireTest class is built to test all major use and edge cases within the Card, Deck, Board, Queue, and Print classes and methods. In the future, I may want to implement Junit testing, but as of now, try-catch blocks work equally as well.

**Suit Enum**

Specifications

The Suit Enum simply provides values for all the suit types of a Card, from *SPADES* to *DIAMONDS* and everything in between.

**SuitMap Class**

Specifications

The SuitMap class also makes special use of the Apache Commons Collections 4 BidiMap and the TreeBidiMap in order to map the type Suit object keys to the String values.