University of Puerto Rico

Rio Piedras Campus

Faculty of Natural Sciences

Computer Science Department

Ángel G. Romero Rosario

801-18-2064

High Level Programming Languages Class

Profa. Patricia Ordóñez

Design Document for Rummy Card Game Program

Part 1 :

Relationships between classes.

Card.java //This class implements the CardInterface; an object constituted of a rank, a suit and image. It is the "card" the player will be playing with. The game centers around this class.

Deck.java //This class implements a Deck constituted of the game Card's objects. It saves 52 Card(s) which can be taken by players on every turn. It has various functions that allows the program to move the cards through the Table Object.

Hand.java //This class represents the 9-10 Card objects taken from the Deck or Pile every player has throughout the game. It implements the HandInterface and it allows players to sort the Card objects and their functions.

Set.java //This class is a representation of a Hand but smaller. It represents 4 cards of the same rank. It inherits the Hand class with all its functions.

Table.java //The whole game runs around this class object. The Table object implements Card and Deck and it will implement Hand, Set and Pile. As it is now, Table creates its own pile, hand and sets based on the other classes and functions. It also implements the GUI.

Part 2 :

Class objects that I am going to implement:

public class MyStack(); //This class implements all the basic function of a stack such as pop, top, push, etc.

MyStack.java functions:

1) **public Object pop();** //This function removes and returns the last object inserted in the Stack.
2) **public Object top();** //This function returns the last object inserted on the Stack class without removing it.
3) **public int getSize();** //This function returns the quantity of objects inside the Stack object.
4) **public Boolean isEmpty();** //This function checks if the Stack object is empty and returns true if it is, false otherwise.
5) **public void push(Object C);** //This function receives an object and adds that object to the top of the Stack Object. Returns nothing.

public class Pile extends MyStack(); //This class extends the MyStack class and works as a pile stack for the player's game. It has functions that allows to take the first card on the Pile, to add a card on the Pile top, to look for the Pile's size. This object will be a different one from the deck but at the same time it will have similar functions.

Pile.java functions:

1) **public Card takeFirst();** //This function returns the top card from the Pile Object using pop().
2) **public Card TopCard();** //This function returns the top card without taking it out of the Pile.
3) **public void addCard(Card C);** //This function adds a card object to the Pile Object using push().
4) **public int size();** //This function returns the size of the Pile object.
5) **public Boolean isEmpty();** //This function returns true if Pile object does not have any Card objects. False otherwise.

New functions:

public boolean Play(); //This function will be implemented to create an automatic player. First it takes a card from the deck or the stack randomly. After that, it checks if there is a set of the same rank. If true, it lays that set. Then it checks if there is another set, if false, puts one random card on the stack and the player turn is over. Then return true because player played.

public void displayWinner(int p1Points, int p2Points); //This function receives the total points from both players and it displays on screen who won or displays "It's a tie!" if the player's points are the same. Function returns nothing.

Modified functions:

- **public void sort();** //This function from Hand.java was modified to have much control over the program. It sorts the hand in ascending order by rank using the selection sort algorithm. It receives nothing and it returns nothing. It will work on all the other sort() functions.

- **public Deck();** //This constructor for the Deck.java object will be filled with Card objects once initialized and then it will shuffle() those objects.

Design Patterns:

Singleton Design Pattern: I am going to use this DP to make the Pile class. This way, the program will only have 1 stack that can only be accessible by a getter function. I can achieve this by making the object private.

Composite Design Pattern: (Tentative) I can modify the Card class by adding a private "deck" of those Cards inside the Card class. I can then add the functions to manipulate the Deck and a getter to that Deck object.

MVC (Model View Controller) Design Pattern: In table.java, this Design Pattern is used for notifying all the functions that manage the imported "data" from the *xswing* and *awt* libraries. In table there is the JFrame and the ActionListeners being representations of the Model, the view and the controller being the Table object.