Old MACdonald APCS1 pd5
Manahal Tabassum, Angela Kim, Clive Johnston 2017-01-10

Final Project Proposal

The UNOfficial Game of UNO

Overview

We want to simulate a game of Uno, abiding by standard Uno rules. This game will be played via terminal output. At the start, the user will be prompted with a multiplayer option and a player vs computer option. They will get to choose the number of players too.

1. Setup: Each Player will be dealt 7 Cards of a random assortment which may include cards of different colors and numbers, or even wild cards if they get lucky.
2. How to Play: For each turn a player will be able to either place a card, draw a card or say uno.
    a. They will only be able to say uno when they have only one card left.
    b. In order to place a card either the number or color has to match the card that was previously placed.
    c. If a player wants to place a wildcard or a +4, they can use it whenever they would like to.
    d. The computer, on its turn, will just randomly select a card that it has that is allowed to be played. So, it won't be too smart.
3. How to Win: Once one player runs out of all of their cards, they will be declared the winner.

Overview of Our Classes

- Super class Card will be abstract
    - It will have various attributes such as name.
    - It will specify the method signature of various methods such as a method to play the Card, and a method to see if the Card can be played on top of the current Card in the play pile. This is because all cards can be played but the way in which a wild card works for instance is different from that of perhaps a normal numbered+colored card.
- Various other classes, Number, Wild, Skip, Reverse, DrawTwo, DrawFour will all extend Card.
    - Each of these classes will provide an implementation for the abstract methods in Card, like play(), canPlay().
- Super class Player will have attributes like name and cardInventory
    - cardInventory will be an ArrayList containing the Cards each Player has. An ArrayList is a good choice because the Cards a Player has are always changing.
- Driver class Woo will bring everything together
    - It will create multiple Players, based on a number inputted by the user, and store them in an array
        - Inputs will be processed by Keyboard.java, which will be imported
    - It will keep track of which card is currently on the top file through a static variable.
    - It will cycle through the Players in the array, representing turns

        - Static variable order will keep track of what direction the turns are going in (based on if reverse cards have been played)
    - Every turn, the user will be presented with a print-out of their cardInventory, and be prompted to choose a Card to play. The Card will invoke its canPlay() method, and then its play() method if canPlay() returned true
    - There will be an array that keeps track of the remaining cards in the deck, dealing out the last element of the array whenever a Player draws a Card(s).

Concepts/Materials

- **Inheritance** is beneficial here because every type of Card has certain similarities, but they also differ from each other, and require unique atributes and methods.
- The **abstract** nature of Card will force each Card to have their own play(), and canPlay() methods. Every type of Card must be able to be played, but every single type of Card can be played under different circumstances, and has different actions, which makes abstractism a good choice.
- **ArrayLists** will also be used to store which cards every Player currently has. We will be able to add cards and remove them this way. Since we are utilizing inheritance, if each player has an ArrayList of Cards, they'd be able to have all the special ones and normal ones since they all belong to superclass Card.