5 Homework 5: Uno Design

Due: 11/12 11:59pm

In this homework, you will design, implement, and test a software version of the Uno card game.

This homework does not fully specify the signature of every method, the types you should use in the class representations, and so on. You are expected to make reasonable inferences about what to implement and how to implement it.

The homework will be graded based on the principles of class design, the rules for Multi-Class Design Documentation, the correctness of your implementation, the quality and comprehensiveness of your testing, and the reasonableness of your inferences about issues that the homework does not explicitly specify.

Additional rules:

- Each class must be in the `cs410.uno` package.
- You cannot use this document as an excuse to avoid writing important information in the Class Design Documentation.
- Your work must be on a branch named `hw5`, which should be based upon the course repository's `hw5` branch. See Making the `hw5` branch.
- On the `hw5` branch, there must be a directory named `Uno`, and the contents of that directory must follow the Maven Directory Layout. It must be possible to build and test your project using Maven (`mvn compile` and `mvn test`, respectively).
- If your work is not in the correct branch, or not in the correct directory, or cannot be built with Maven, then it will not be graded and you will not receive credit for the assignment.

5.1 The Rules of Uno

The rules here are slightly simplified from the official Uno rules.

Uno is a card game played with non-standard deck.

It requires at least two players. The players are arranged in a circle, and players take turns in order around the circle. The order of players within the circle never changes, but the order that turns are taken (clockwise vs counter-clockwise) may change during the game.

Each player is initially dealt a certain number of cards from the deck.

After dealing each player their initial hands, the next card is dealt face-up to the table and becomes the start of the "discard pile". The undealt cards in the deck are placed face-down next to it and are called the "draw pile". During the game, players "play" cards by placing them face-up on the top of the discard pile, and they may draw new cards from the top of the draw pile.

The first player to get rid of all of their cards wins the game.

5.1.1 Taking a Turn

When it is a player's turn, if the player has any playable cards in their hand, they must play one of them, and then their turn ends. If the player does not have any playable cards, they must draw one card from the draw pile; if they draw a playable card, they must immediately play it, and then their turn ends. Otherwise they add the drawn card to their hand, and their turn ends.

Normally, when the player's turn ends, the next player in the circle then takes their turn, but some cards disrupt the normal order of play.

5.1.2 Uno Cards

The deck contains "normal cards" and "wild cards".

Each normal card has a color (red, yellow, green, or blue), and it has either a digit ("0" to "9") or a special instruction on its face. There are the following kinds of special cards:

- "Skip": The next player is skipped, and the player after the skipped player takes the next turn.
- "Reverse": The order of play around the circle is reversed. If it was previously clockwise, it now becomes counter-clockwise, and vice versa.
- "Draw Two": The next player must draw two cards, and then their turn is over, and play continues with the player after them. They do not get an opportunity to play a card that turn.

A wild card has no color, but the player declares a wild card's "effective" color when the card is played.

5.1.3 Playable Cards

A *playable card* is a card that can be discarded onto the top of the discard pile.

A normal card is playable if it matches either the color or the face of the top card on the discard pile. For example, a Red 7 can be played on top of a Red 3, or a Red Skip, or a Blue 7; but it cannot be played on top of a Blue 3.

A wild card is always playable.

If the top card on the discard pile is a wild card, it is treated as the color declared by the player who played it.

5.1.4 Running out of draw cards

If the draw pile becomes empty, it is refreshed from the discard pile. The top card of the discard pile is retained; the new discard pile consists of only that card. The other cards in the discard pile are shuffled and placed face-down, and they become the new draw pile.

5.2 Designing an Uno Package

This homework assignment partially specifies one class: `GameState`. The organization and design of the rest of the package's classes are up to you.

There is no way for the autograder to test your code's correctness. You must write your own tests sufficient to convince the human graders that your code behaves correctly. Your classes should be designed to support testing.

5.2.1 The `GameState` class

You must design a public class called `GameState`.

The `GameState` class must have a static factory method called `startGame` that takes the following integer arguments:

- `countPlayers` — number of players
- `countInitialCardsPerPlayer` — number of cards initially dealt to each player
- `countDigitCardsPerColor` — number of normal cards for each digit and color. For example, `2` means 2 Red "0" cards, 2 Yellow "0" cards, etc.
- `countSpecialCardsPerColor` — number of special cards of each kind for each color. For example, `1` means 1 Red Skip card, 1 Yellow Skip card, ..., 1 Red Reverse card, 1 Yellow Reverse card, etc. May be `0`.
- `countWildCards` — number of total wild cards. May be `0`.

After the `startGame` method ends, the game state should represent the situation immediately before the first player takes their first turn. That is, the players should be arranged, their initial hands have been dealt, and the discard pile and draw pile have been created.

You must also implement the following methods in `GameState`:

- `public boolean isGameOver()` — Returns true if the game is over.
- `public void runOneTurn()` — Runs one turn of the game.

See the declarations and comments in `GameState.java` for more details.

5.3 Making the `hw5` branch

These instructions assume that your local working repo has two remotes, `origin` and `course-repo`, as described in the instructions for Homework 3: Multi-Class Design.

Run the following commands in order:

- `git fetch course-repo`
- `git branch hw5 course-repo/hw5`
- `git switch hw5`
- `git push --set-upstream origin hw5`

Go to your gitlab page and verify that it now contains a `hw5` branch.