Assignment 4: Freecell (Part 3)

Due Jun 4 by 8:59pm **Points** 0

A Freer FreeCell

Starter files: code

1 Purpose

In this assignment we will reap the fruits of our design, by introducing a new variant in the game of Freecell: the ability to move several cards at once. You will see how your earlier design of this program will facilitate adding this feature with (hopefully) minimal change in code.

All *new* classes and interfaces for this assignment that pertain to this new model variant should be in the `cs3500.freecell.model.hw04` package. All classes written in previous assignments, even if improved upon, should remain in their respective packages.

A starter file is provided above whose sole purpose is to ensure your code is in the correct packages with the correct visibility.

2 Multi-move Freecell

A more common version of Freecell is one where the player can move several cards at once from one cascade pile to another (while it is also possible to move several cards from a cascade pile to a foundation pile, we will ignore this feature in this variation).

Moving multiple cards must obey two conditions. The first condition is that the cards should form a valid *build* , i.e., they should be arranged in alternating colors and consecutive, descending values in the cascade pile that they are moving from. The second condition is the same for any move to a cascade pile: these cards should form a build with the last card in the destination cascade pile.

Note that the ability to move multiple cards is not a special feature, but a convenience to the player. A multi-card move is basically several single-card moves, using free open piles and empty cascade piles as "intermediate slots." Thus a multi-card move may not be legal even though it obeys the above two conditions, if there aren't enough of these intermediate slots available. More specifically, it can be proved that the maximum number of cards that can be moved from one cascade pile to another when there are N free open piles and K empty cascade piles is $(N+1)*2^K$. Your implementation of this variation should enforce all three of these conditions.

In order to use an empty cascade pile as an intermediary for multi-card moves, we will allow any card to move to an empty cascade pile (not just a king, as some Freecell versions mandate).

3 Design Constraints

In this assignment, you must introduce this new feature in your game while respecting the following constraints:

- You are not allowed to change the controller interface (`FreecellController`) or implementation (`SimpleFreecellController`) at all from **Assignment 3** , except to fix bugs or design issues pertaining to that assignment.
- You are not allowed to change the interface of the model (`FreecellModel<K>`) at all from **Assignment 2** .
- You must create a separate model implementation while keeping `SimpleFreecellModel` from **Assignment 2** working as it was. That is, models that represent both variations of the Freecell game (single and multiple card moves) must co-exist. You are free to name the new model implementation whatever you like.
- You must provide a factory class `FreecellModelCreator` , that works as specified below.

You *may* have to refactor your earlier design to do this. This is OK, but should be **properly documented and justified** . Again, you are not allowed to change existing interfaces or add new public methods.

4 The FreecellModelCreator class

Write a class called `FreecellModelCreator` and put it in the cs3500.freecell.model package. The class should define a public enum `GameType` with two possible values `SINGLEMOVE` and `MULTIMOVE` . It should offer a static factory method `create(GameType type)` that returns either a `SimpleFreecellModel` or an object of your multi-card-move model, depending on the value of the parameter.

5 Deliverables

- Your new "multi-card move model" class
- Your factory class
- Your controller and model classes and related interfaces from **Assignment 2** and **Assignment 3** (possibly modified only to remove bugs and design limitations)
- Tests for your multi-card move model and the related game in a JUnit test class. It is a good idea to include your earlier tests as well, for regression testing. We will.

6 Grading standards

For this assignment, you will be graded on

- Whether you had to modify any previously written interfaces,

- the extent to which your implementations share code as opposed to repeating code,
- whether your code implements the specifications (functional correctness),
- the clarity of your code,
- the comprehensiveness of your test coverage, and
- how well you follow the prescribed style.