CMSC 202 — Fall 2013
Project 3 — Monte Carlo Solitaire

Assigned	Sunday, October 20
Program Due	Monday, Nov 4rd by 8:00am
Weight	8%
Updates	As a point of clarification, the `Game` class's `getSuit(Coordinate)` and `getRank(Coordinate)` methods should return `null` if no card is present at the specified coordinate.

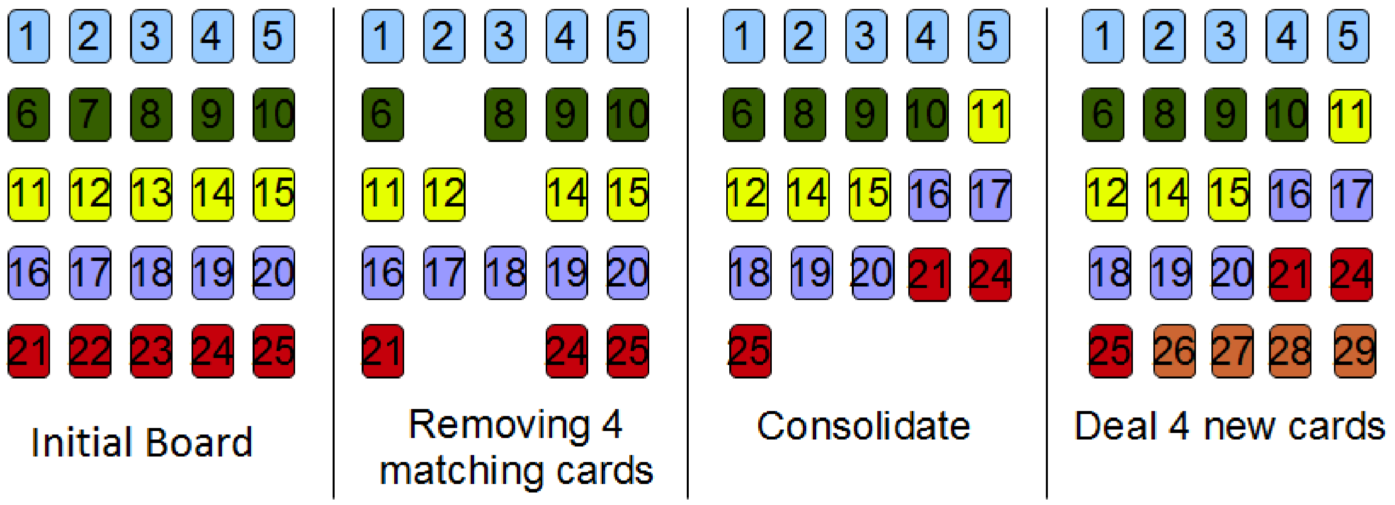
Objectives

- To gain experience with object-oriented program design
- To gain experience using composition
- To gain experience interfacing with classes designed and implemented by someone else
- To appreciate the value of writing reusable and easily modifiable code
- To reinforce the use of preconditions, postconditions, and class invariants
- To implement and appreciate the value of unit testing

Project Description

As an employee of the ***UMBC Game Company***, you have been assigned to the programming team responsible for designing and implementing a version of " Monte Carlo Solitaire." Monte Carlo Solitaire is played with a standard deck of 52 playing cards (4 suits and 13 ranks). Initially, 25 cards are played face up, in 5 rows with 5 cards in each row. These cards are called the "tableau." The player removes pairs of adjacent cards (horizontally, vertically, or diagonally).

When no more pairs can be removed, the tableau must be "consolidated." Consolidating the tableau will attempt to shift the remaining cards to fill in the holes in the tableau. The tableau will traverse itself row by column looking for any holes. When a hole is found, the tableau will move the next card on the traversal into the hole that needs to be filled. If there is no card in the tableau that can be used to fill that hole, a card must be dealt from the deck and placed into the hole in the tableau. The consolidate operation has completed when the tableau has completed the traversal of itself, or there are no cards left in the deck. To help you visualize the consolidate operation, use the following image of the tableau.



Online Demo

You can play an online version of the Monte Carlo Solitaire application you will be building below.

Launch

If you have Java installed you should be able to download and run a demo of the app by simply clicking on the button above. If you have any issues running the app that way, try downloading the p3.jar file directly and run the demo with the following command: `java -jar p3.jar`

Game FAQs

1. ***How is the game number chosen and how is it used?***

   When the game initially starts, a game number is randomly chosen by the GUI. A new game can be created by either selection a game number at random (Game → New Random Game) or by specifying a specific game (Game → New Game…). The game number is used to initialize the random number generator used to shuffle the cards.

2. ***What does the "Control → Hint" menu option do?***

   When "Control →Hint" is selected, a pair of cards that may be removed are highlighted (their borders change color). If no pair of cards can be removed, a message indicating no move is available is shown.

   Implementing "Hint" is worth 5 points extra credit.

   Be sure that you set `Game`'s `isHintImplemented()` method accordingly.

3. ***What does the "Help → Monte Carlo Solitaire" menu option do?***

   The "Help → Monte Carlo Solitaire" option displays a pop-up window that briefly explains the rules of the game.

4. ***What does the "Game → New Random Game" option do?***

   The "Game → New Random Game" option abandons the current game, selects a new random game number, seeds a random number generator, and starts the new game.

5. ***What does the “Game → New Game…” option do?***

   The “Game → New Game…” option abandons the current game, prompts the user for a new game number, seeds the random number generator with that selection, and starts the new game.

6. ***What does the “Game → Replay Current Game” option do?***

   The “Game → Replay Current Game” option restarts the current game from the beginning.

7. ***How do I win the game?***

   By removing all the cards from the tableau, hence earning a score of 52.

8. ***What's the tableau?***

   The cards that are face up in a rectangular arrangement (usually 5 × 5).

9. ***How do I remove cards from the tableau?***

   Clicking on a card highlights that card by changing its border color. If the next card clicked is touching the first card and is of the same rank, the cards are removed. If the second card is not touching the first card or is not of the same rank, a beep is sounded. Suits are not relevant when choosing cards to remove.

10. ***When are two cards touching?***

    Two cards are touching if they are next to each other horizontally, vertically, or diagonally. Cards at the top and bottom of a column, or the ends of a row, or the opposite corners of the diagonal are ***NOT*** considered touching. That is, the columns, rows, and diagonal do not wrap.

11. ***What do I do when I can't remove any more cards?***

    Push the card to the top-left or select the “Control → Consolidate” button. This ***always*** moves all cards in the tableau left and up towards the top of the tableau. If there are any cards left in the deck, then this button also deals cards from the deck to refill the bottom of the tableau.

12. ***What if there aren't enough cards in the deck to fill the bottom of the tableau when I “consolidate”?***

    If there are too few cards in the deck to complete the tableau, the remaining cards are dealt to complete as much of the tableau as possible.

13. ***How does my score change?***

    One point is awarded for each card removed from the tableau.

Project Policy

This project is considered an ***OPEN*** project. Please review the open project policy on the course website.

Class & Enum Descriptions

Our application consists of many classes and enums. Some of the classes and enums have already been written by other team members. Some of the classes must be written by you. Some classes must support a specific API, and still others may be designed and implemented as you see fit.

View all Javadocs

- ***Project3 Class***

  Project3.java contains the GUI framework and the project's main. These have been written by someone else. The GUI code in `Project3.java` calls methods from the Game class that you will design and implement. Do not submit `Project3.java`.

- ***ClickableImage and ClickableImageGrid Classes***

  ClickableImage.java and ClickableImageGrid.java are support classes used by `Project3.java`. Do not submit `ClickableImage.java` or `ClickableImageGrid.java`.

- ***Suit Enum***

  Suit.java contains the enum definition of Suits used to identify a card. For example, `Suit.HEARTS` is used to represent the suit hearts. This class is provided so that there is no possible discrepancy among all classes that refer to a card's suit. The four suits in a standard 52-card deck are `Suit.HEARTS, Suit.SPADES,` `Suit.DIAMONDS`, and `Suit.CLUBS`. Do not submit `Suit.java`.

- ***Rank Enum***

  Rank.java contains the enum definition of Rank used to identify a card. For example, `Rank.TWO` is used to represent a card with a rank of two. This class is provided so that there is no possible discrepancy among all classes that refer to a card's rank. The ranks of the cards in a standard 52-card deck are `Rank.ACE`, `Rank.TWO`, `Rank.THREE`, ..., `Rank.JACK`, `Rank.QUEEN`, and `Rank.KING`. Do not submit `Rank.java`.

- ***Game Class***

  Game.java must be written by you. This class contains data and methods used by `Project3.java` to drive the GUI. The API is provided so it is clear how `Project3.java` will utilize `Game.java`. This class is will prevent `Project3.java` from compiling until it is implemented.

- ***Coordinate Class***

  Coordinate.java must be written by you. This ***immutable*** class encapsulates the row and column coordinates of a card in the 5 × 5 tableau. This class is also used by `Project3.java` and will prevent `Project3.java` from compiling until it is implemented.

- ***Other Classes***

  Design and implement any other class(es) that you feel is(are) necessary to complete a good object-oriented implementation of this project. Yes, there will be some other class(es).

You are not permitted to make any changes to Project3.java, ClickableImage.java, ClickableImageGrid.java, Suit.java, or Rank.java.

However, you are welcome to add additional members/methods (public or private) to the other classes referenced in the Javadocs. Additionally, you are welcome to add additional helper classes as you see fit.

Project Requirements and Specifications

1. This project will require you to create a new project based on one that we have started to develop. Download p3.zip and use Eclipse's import tool to create a new project from an Existing Project.

2. You must write `main()` in the Coordinates class, the Game class, and any class you design and implement as a means of unit testing. As discussed in class, each `main` should instantiate an object and invoke all public methods defined for the class, printing the results to `System.out`.

3. For any given game number from the online demo GUI, your application should duplicate the tableau for that same game number.

4. In order to duplicate game #12345 as required above, it is necessary that the deck of cards be initialized appropriately before shuffling. Initialize the deck of cards in the order: A♣, 2♣, ..., Q♣, K♣, A♦, 2♦, ..., Q♦, K♦, A♥, 2♥, ..., Q♥, K♥, A♠, 2♠, ..., Q♠, K♠.

5. The algorithm below, which shuffles an array of items, must be adapted and implemented to shuffle the cards to start a new game of solitaire.

   A. A[0], A[1], ..., A[N-1] is an array of items
   B. Let n = N-1
   C. Pick a random index, k, between 0 and n (inclusive)
   D. Swap the values of A[k] and A[n]
   E. Decrease n by one
   F. Repeat from step (c) until n is less than 1

   To choose a random index (step c), use the `nextInt` method of the `Random` class found in the `java.util` library. Note: Read the description of the `nextInt(int n)` method carefully!

6. Per the course standards, all class comments must contain the class invariant and all method comments must include pre- and post-conditions.

7. Make all .java files you create part of package “proj3”.

8. You may not use any objects from the java Collections, which includes ArrayLists or any ArrayList like objects.

9. You may use any classes you've already written, provided you do not violate the above constraint.

Hints and Tips

- We have alluded to the possibility of other classes being required to earn full design points. Make sure you carefully read the project description so that you can identify the objects in the problem correctly.

- Your code may assume we are using a standard 52-card deck.

- Any multi-dimensional array can be implemented as a one dimensional array.

- Develop your classes slowly. As you write new methods, write code in your class' `main` to test them.

- Add precondition checking to each of your methods last. Save, compile.

- ***DO NOT*** try and implement an ***ENTIRE*** class at once. Use incremental development. Develop one method at a time, write code that will thoroughly test it, run and test it, then move on to another method.

- ***Do not cheat — you WILL be caught.*** Do not show your code to another student. Use the tutors, TA's, and instructors when you need help.

- This is a GUI-based project. Therefore, to test your code once you've moved it to your GL account, you must use a Linux machine in one of the labs to run it. You will not easily be able to run your program on GL from home using an SSH client (e.g. PuTTY, TeraTerm, etc.).
- Follow the K.I.S.S. Principle
- ***Testing***
  1. Basic Cases — these might be tests such as creating a game, removing cards, viewing the score, or other simple tasks related to any class.
  2. Complex Cases — these might be tests such as consolidating after removing cards, restarting a game, and so on.
  3. Atypical Cases — these might be tests that attempt to break your code based on input or class invariant violations. These might attempt to create a new game with a negative value, remove non-adjacent cards, and etc.

Grading

See the course website for a description of how your project will be graded.

Project Submission

1. **submit** `Game.java`, `Coordinates.java` and the .java files for any class(es) you design and implement.
2. ***DO NOT*** submit any `.java` files from the provided zipfile. These files should not be changed by you and so need not be submitted. If you submit these files, they will be removed from your submittal directory.
3. **submitls cs202 proj3** to verify that your files are in your submittal directory.

The order in which the files are submitted doesn't matter. You need not submit all files at the same time. You may resubmit your files as often as you like, but only the last submittal will be graded and will be used to determine if your project is late. For more information, see the projects page on the course website.

You can check to see what files you have submitted by typing

```
submitls cs202 proj3
```

See the Project Submission page for detailed instructions.

Remember — if you make ***any*** change to your program, no matter how insignificant it may seem, you should recompile and retest your program before submitting it. Even the smallest typo can cause compiler errors and a reduction in your grade.

Avoid unpleasant surprises!