APPLICATION 4: LIST – CARD GAME

TOPIC(S)

List application: multi-player card game.

READINGS & REVIEW

- Carrano, Data Structures and Abstractions with Java, Chapters 7-9, 12-15
- List ADT & implementation lectures

OBJECTIVES

Be able to:

- *Utilize one or more Lists as part of an implementation of a multi-player card game.*
- *Produce a high-level, text description of your game and the entities which are used in the real world to play the game.*
- *Create a base class and derived classes to implement more specialized behaviors (proper use of inheritance).*

INSTRUCTIONS

1. **You will work in self-selected groups of 2-3 members (self-enroll in Blackboard).**
2. ***Read the assignment and ask questions about anything that you don't understand (before you start).***
3. For your application:
    a. Be sure to follow Good Programming Practices.
    Your code must comply with the Coding Standards/Guidelines posted on Blackboard.
    b. Keep in mind the Guidelines on Plagiarism.
4. Do a Write-up (Analysis / Summary). Each group member must submit this individually – it goes in the ADT assignment submission, not the application assignment!
5. Prepare a PowerPoint presentation. Your group will present during the class following the posted submission due date. Your group will submit the PP presentation file as part of the group submission – it goes in the project root folder.
6. Submit your work by the posted due date/time. Late submissions will incur a 25% penalty per day. Non-working projects (major or minor bugs) will incur at most a 10% penalty.
7. If you'd like my assistance, please zip your entire project folder and attach it to an email message with a brief description of your question/problem. I will typically respond within a few hours. Do not expect a response after 9p (5p on the due date).

PART 1: CARD GAME MODEL & ALGORITHMS

Practice modeling real-world objects. Specifically, those entities needed to play a multi-player card game.

(1) Write algorithms to model playing a multi-player card game.

- Your game may be a real game or one of your own creation.
- The computer can be one of the players.
- You need to describe the behaviors for:
    - A card
    - A pile of cards
    - A deck of cards
    - A player's hand
    - A player
    - The game
    - Any other entities you may need (e.g. table)

How many players participate? Fixed number? Minimum required? Maximum supported? If applicable to your game, can you run out of cards? If so, now what?

Are there other objects that affect playing the game?

How will users interact with your game?

Additional things to consider (not required for this assignment):

- Do you allow betting? If so, what additional objects do you need in your model? How do they behave? How do they affect the required components (first list above)? If you don't include betting initially, what effort might be required to add it later?
- The assumption for this model is that all players are physically located in the same place. What complications might arise if you allowed players to participate remotely?

Feel free to add your own questions.

The deliverables for this portion of the assignment are:

- a written description of your model in terms of the required components and their respective behaviors. The only requirement for the format is that your group's identifying info must appear at the top of the page. The file can be any text document type (e.g. .docx, .txt, .pdf) – that is, anything that I can open and print.
- UML diagrams of all classes
- The answers to the questions (above)

You can use my sample train simulation algorithms write-up (on Blackboard) as a guide for the write-up or choose any other format that simply describes your model.

Each group will turn in their part 1 write-up, and the instructions for their game, on **Tuesday, 4/9/2019** at the beginning of class. This submission must be printed (single- or double-sided).

You will also submit the write-up with the Eclipse-compatible project – put it *in the project root folder*. The write-up must be named: DS - List App write-up - Group##.docx.

Questions? Ask.

Have fun!

Dave

PART 2: APPLICATION/CARD GAME IMPLEMENTATION

The final objective of this project is to create a multi-player card game. **All classes described below are *required.* Create your classes in the** *order given, as specified. (Analogy: you must have a solid foundation before you can build a house).* In order to allow creative freedom, some of the **details** of selecting instance variables and methods for classes are **left to the student**. However, the instructor will provide **direction and assistance as needed**.

If done properly, your classes will track your algorithms. If they don't, revisit the algorithms and revise them or your code or both.

CARD

Write a Card class to represent a playing card.

- As usual, provide constructors and *accessor* methods for all instance variables, and a *toString()* method for displaying the card. Note that the instance variables for Card typically can't change so they should *final*.
- Have it implement *Comparable* (which will require you to write a *compareTo()* method) so it will be easier later to compare cards. You should also override the *equals()* method.
- You may want to use enums for face/rank, suit, color, etc. (samples which you may use are provided)

Test your Card class before you proceed! ***For this course this can be JUnit tests or including a main() method in each class, with object creation and method calls.***

PILE (OF CARDS)

Write a Pile class to represent a general pile of cards. Your pile must be/utilize a List. You can use any List implementation (from the Java library class) but you must limit yourself to use only List and Iterator methods.

As usual, provide *constructors*, *accessor* and *mutator* methods for all instance variables, and a *toString()* method

Include methods representing **operations** on piles of cards, such as

- Shuffling and Sorting
- Removing and adding cards.
- Searching
- Splitting
- Grouping by face/rank or suit.
- Other?

Other methods may be added later if you determine that you need them.

Thoroughly test your Pile class methods before you proceed (will save you time and frustration later).

DECK AND HAND

***extend* classes Deck and Hand from Pile**. Add instance variables and behaviors (methods) specific to each. For example, Deck would have a *deal()* method. As usual, provide constructors, accessor and mutator methods for all instance variables, and a *toString()* method. *Once you write a card game class you may determine that these classes need additional instance variables and methods. The default constructor for a Deck will probably instantiate all of the Cards in the Deck.*

Thoroughly test your Deck and Hand classes before you proceed!

PLAYER

Write a Player class. Include instance variables related to players, such as name, hand (instance of your Hand class), and score. Include methods related to players. These may also be added later as you write your game class.

Thoroughly test your Player class before you proceed.

GAME

Determine a multiple-player card game that you'd like to implement that is complex enough to require:

- **Objects** from your Player, Card, Deck and Hand classes
- **Operations** provided in your classes, such as shuffling, sorting, searching, dealing, determine win,…

I must approve your game choice.

Write a game class – name it according to the game you implement (e.g. GoFish). Include instance variables related to your type of game, such as *Players* and at least one *Deck.*

Depending on the game you are implementing, the methods you have might include *initialize()* to set up for a new game, and *play()* to run the game.

You have creative freedom to develop additional classes if they make your implementation cleaner and/or more elegant.

Have fun playing your card game!

The assignment includes a zip file containing:

1. A PowerPoint document/template for the presentation.

   Unless there is a compelling reason to do otherwise, please use PowerPoint for your presentation. *If you wish to use a different program, you need to get my approval in advance*. You do not need to use the provided templates, however, your documents must include all of the information in them including headers (if included), identification blocks, and footers. *Make sure you update all identifying information in the footers (typically the assignment and group numbers – the file name, date, and paging update automatically).*

2. PDFs of game rules.

GOOD PROGRAMMING PRACTICES

For all classes (required except as noted):

- In your IDE, you must rename your project "DS - List App - Group ##" where ## is your group number – do not include the double quotes – you must match the spacing and hyphenation precisely. Warning: You must type the project name – do not copy/paste – Word does not use the correct hyphen characters. (I provided an Eclipse project containing template classes – except the game – rename the project to reflect your group number.)
- [***Required***] Design your solution using UML diagrams before you begin coding.
    - Provides understanding of the big picture and how classes relate to one another.
    - Some IDE's will then create the framework for your classes right from the diagrams.
- Use mnemonic/fully self-descriptive names for all class members (methods, variables, parameters, etc.).
- Make your instance variables **private.**
- Include **constructors** to initialize your instance variables.
- Derived class constructors should **leave initialization of super class instance variables** to the super classes' constructors:
    - Remember the call to the super classes constructor is: ***super()*** or ***super(<init1>, <init2>,..).***
- If appropriate, include **accessor** and **mutator** methods for all instance variables *(please ask if you're not sure what these are)*.
- Include a ***main()*** method for testing (unless it's an application rather than a utility class) and test before you proceed…

Add comments to your code, not just so it's easier for other readers, but also so it's easier for you to remember your logic.

WRITE-UP

ANALYSIS

The write-up is an individual assignment. Each group member must complete the write-up included with the ADT assignment and submit it to the List ADT & App write-up assignment, not the application assignment on BB.

SUBMITTING YOUR WORK

1. Make sure the course number, assignment (App 4/List: <name of your card game>), your group number are in all your project files. For the Powerpoint presentation, list each team member's name and the specific class(es) they 'own.' ***For the source modules, include the name of the class owner – the individual primarily responsible for the design, implementation, testing, and maintenance – do not list all members in all classes – you must delegate responsibilities.*** If any other group member makes *significant* contributions, list their name(s) too.

2. Your presentation file name must be consistent with the template:
    a. Required: update/insert your group number
    b. Optional: update the date/version number

3. Your Java class files must be 'properly' named/match the class defined within.

4. You should include a comprehensive set of unit tests for your classes (except for your application). I suggest that the entire team define the test 'suite'. For simplicity, include a *main()* method for each class to run your tests; create private utility methods as appropriate for the testing. Or, you may implement the tests using JUnit.

   Style requirement: For unit tests using a *main()* method, *main()* must follow all public and private methods which implement the API/ADT. *private* utility methods for testing must follow the *main()* method.

   Style requirement: For console applications, you may read a file containing the configuration parameters controlling the simulation. All data/text/input files must open from the default location (typically the project root folder). ***You are not permitted to specify a path to the files in your application.***

   Style requirement: You are not permitted to use global variables.

   Style requirement: You are not permitted to use separate, distinct variables for each player, hand, etc.; you must use an appropriate collection instead. You will likely use temporary variables to instantiate and manipulate particular instances. Consider this carefully in your design so you can ask the user(s) how many players (for example).

   Style requirement: You must include Javadoc comments in all classes. Generate the Javadoc for the entire project (if you generate it, it must be in a \doc folder 'next to' the \src folder in the project directory tree).

5. Include documentation for the rules of the game you implement. Make sure the rules and the implementation match! Any text or PDF format is acceptable. Do not include an image without prior approval. (Note: The Rank and Suit enums do not have complete Javadoc comments – you do not need to complete them.)

6. Spell check and grammar check your work!

7. Make a **.zip file** for your project. ***.rar, .tar, etc. are not acceptable!*** Instructions for creating a zip file are posted on Blackboard.

   - Include your entire project folder (including subfolders).
   - Your PowerPoint presentation and game rules must go in the project root folder. **Do *not* attach them directly to the Blackboard submission.**
   - In Blackboard, **attach** your solution file to the submission for this assignment – click on the assignment title to access the submission page. **Only one team member submits the project/code and PowerPoint presentation for the entire team**.

8. Each group member must complete the write-up included with the ADT assignment and submit it as part of the ADT assignment, not the application assignment on BB.

GROUP PRESENTATION

1. All group members are expected to participate in the presentation. The presentations will be in class on Tuesday, 4/16/2019.

2. Make sure your group number and all of your names and roles are on the title slide and the group number is in the footer of all following slides.

3. You may use any theme, template or style you wish (recommendation: keep it appropriately professional). Make sure the font size is large enough to read from the back of the room.

4. Minimally, your presentation must include all of the information in the provided template.

5. Presentations should take approximately 3-5 minutes, including a demonstration of your application and Q&A.

6. ***Do not read the slides to the class***. The slides should typically be short, bulleted lists of talking points. Face the class when speaking and speak loudly enough to be heard in the back of the room.

7. ***Do not include code in the slides***. If you want to show selected portions of your implementation, do so using your IDE or copy the sections of code into another application for display purposes only (again, make sure you set the font to a large enough size – try 20 or 22 pt - to be readable from the back of the room).

8. ***Make sure the presenter's computer is adequately charged and has the correct versions of the PowerPoint presentation and code. Prior to coming up, open the presentation and your IDE (so you can demonstrate your application) and make sure they run properly and that the font size is large enough to be seen/read at the back of the room (use 22 pt).*** I will provide HDMI and Thunderbolt cables/connectors.

GUIDELINES ON PLAGIARISM IN COMPUTER SCIENCE

- ***Plagiarism/cheating will result in immediate failure for this course.***
- ***The Provost's and Registrar's offices will not permit you to withdraw from a course which you failed due to cheating.***
- ***An Academic Review Board may impose additional penalties.***
- ***You will not be allowed to attend class for the remainder of the semester.***

CLEAR PLAGIARISM

A clear case of plagiarism exists if a student **passes off someone else's work as his or her own**. Examples of such behavior include (but are not limited to) the following:

- A group of students each performing a separate part of an assignment. Each student in the group then hands in the cumulative effort as his or her own work. This is different from members of a group dividing up the work then submitting it as the product of the group's efforts (as is the case with this assignment).
- A student making cosmetic alterations to another's work and then handing it in as his or her own.
- A student having another person complete an assignment and then handing it in as his or her own.
- A student handing in (as his or her own) a solution to an assignment performed by someone else from a previous offering of the course.

These are all cases of indisputable plagiarism and are characterized by the submission of work, performed by another, under one's own name.

PERMITTED COLLABORATION

Collaboration and research, where they do not constitute plagiarism, should be generally encouraged. These efforts constitute honest attempts at obtaining a deeper understanding of the problem at hand. They can also serve as a validation of a student's approach to a problem. Some examples are as follows:

- A student researching existing, public approaches to a problem presented as an assignment.
- A group of students discussing existing, public approaches to a problem presented as an assignment.
- A group of students discussing the requirements of an assignment.
- A student discussing the merits of his or her proposed solution to an assignment with the instructor or teaching assistant.

Provided no plagiarism is committed, these are all valid (and encouraged) activities.

THE GRAY AREA

The differences between the examples of plagiarism and encouraged collaboration above are those of originality and acknowledgment. In general, any work submitted without acknowledgment which is not the original work of the indicated author constitutes plagiarism. Some examples which illustrate this point follow. Note that while some of the examples do not constitute academic misconduct, they may be of questionable academic value. Also note that anyone (knowingly or through negligence) contributing to someone else's academic misconduct is also guilty of the same.

- It is acceptable to use solution proposals presented by the instructor or teaching assistant. The acknowledgment is implicit. Explicit acknowledgment is not usually required.
- It is not acceptable to use publicly available work without acknowledgment.
- It is not acceptable to use a full or partial solution obtained from or by another through any means (verbal, written, electronic, etc.), without that person's permission.
- It is not acceptable to collaborate on a single solution without acknowledgment.
- It is not acceptable to discuss solutions or partial solutions to a problem and then incorporate them without acknowledgment.
- It is acceptable to implement a standard solution to a problem without acknowledgment, but it is not acceptable to incorporate someone else's implementation without acknowledgment. Here standard solution means a commonly used data structure or algorithm.
- It is not acceptable to re-submit an assignment from another course or a previous offering of the same course without acknowledgment, regardless of authorship.
- It is not acceptable to make a solution available as an aid to others.
- ***You may not request solutions on any internet site (e.g. stackoverflow.com).***

This set of examples helps define the bounds between encouraged behavior and misconduct, but does not constitute an exhaustive set.