CSCI 345–Assignment 2
Implementation

Dr. Moushumi Sharmin (in collaboration with: Dr. Aran Clauson)

Points: 40 (10% + 5% of your overall course grade)

This part of the assignment is asking you to implement! You analyzed the problem, designed a solution, now it is time to implement the design.

Outcomes

Upon successful completion of this assignment, you will

- Convert your software design into a working program
- Demonstrate how the following class relationships are realized in Java
    - Association, Aggregation, Composition, Implementation, and Inheritance
- Have a text-based (command-based) implementation of Deadwood

Problem Statement

Implement your Deadwood design as a **console application.** This version of your game is indented to be playable but may not be aesthetically pleasing. The point of the program is to verify that your design correctly models the game play logic. For example, players can only move to adjacent rooms and your program must enforce this restriction.

The program **MUST** accommodate the following tasks/actions:

- Identify the active player
    - Display this players' information
- Location of the current players and all the other players on the board
    - Display location of all players and indicate the active player
- Moving from one location to another
    - Display the source and destination
- Working on a part
    - The active player works on a part
    - Display part-related information
- Upgrading level
    - Display current and target level
    - Support different types of upgrades
- Rehearsing
    - The active player rehearses
    - Display related information
- Acting
    - The active player acts on a role
    - Display related information

- Ending the current players’ turn
    - Active player can decide to end a turn
    - The **end** command is recommended even if the active player has no legal actions available
- An **End Game** command
    - This will help with testing as game can be ended at any given time

Example Interactions

Example 1:

If the active player is working a part, the interaction could be the following:

➢ Active player?

```
        The active player is Jane Doe. She has $15, 3 credits and 10 fames. She is working Crusty
Prospector, "Aww, peaches!"

> where is the player located?

        in Train Station shooting Law and the Old West scene 20
> act
        success! You got $1

> end
```

Example 2:

If a player is not working a part, the interaction might look like the following:

```
> who
        player blue ($1, 5cr)

> where
        Jail wrapped

> move Train Station
> where

        Train Station shooting Law and the Old West scene 20

> work Talking Mule
> end
```

We are showing additional information like the player's money and credits. **This would be nice but is not required**. Remember, **the point is to test the design and to have a robust working model, it is not to have a friendly user interface**. In addition, you may find that your design (the one you submitted) has weaknesses. This is typical of any significantly sized software project. It is difficult to anticipate every possible detail of the system. Feel free to modify your design. However, you **must update your design document (class diagram)** as you modify the design. We will use this document to understand your software.

Your program should accept a **single parameter: the number of players.** Like the rule book says, the game is designed for two to eight players.

Deliverables

- You should **submit your code** via canvas **and** give us access to your git repository. We will download your implementation from the repository, so please make sure we have full access to your repository. The file **Deadwood.java** should contain the main program. Our github account information is available on Canvas.
- You will **submit a report (canvas)** specifying **what types of Cohesion and Coupling** you utilized in your design and your rationale for using them. Also discuss your overall design choice and how that relates to SOLID design principles. This is a critical component. The grades you receive on this will be part of your Writing Proficiency (WP) points. [5% of overall grade]
- You will **submit a revised Class Diagram (canvas)** (the one that you implemented) and explain (**1~2 paragraphs**) why you chose this design.
- **Submit your Read Me file (canvas)**
- Email **peer evaluation** (individually, sharmim@wwu.edu). **Subject must include CSCI345 peer evaluation**

Due Date and Mini-Milestones

Mini-milestones are check points along the implementation period. We will use them to ensure you are making progress. *While your software is finally due on Friday, Feb 21st, you have one additional milestone when you are expected to submit partial code.* ***Do not miss the submission deadlines.***

Mid-Milestone: Submit your partially implemented program. You must have skeleton code for all your components (classes and interfaces) and must have complete implementation for at least half of your classes. We will compile your code. **If it doesn't compile, we will deduct 5 points**.

Final Submission:

- Complete code.
- *Read Me* file with instructions about how to compile and run your code.
- Modified Class Diagram.

- Report specifying use of cohesion and coupling and use of SOLID design principles
- **An email** (sharmim@wwu.edu) discussing peer evaluation. Example of good and not-so-good peer evaluations are available on Canvas.

Grading

20 points Your software correctly plays the game of Deadwood™ and uses Object-Oriented principles. We will check all functionality and concepts used. This is where we will check whether your followed object-oriented principles to the best of your ability.

2.5 points Quality of your code (e.g., meaningful comments, well defined methods, descriptive symbols, etc.).

10 points Detailed Description and Rational for Design Choices (explain based on SOLID design principles) and Cohesion and Coupling utilized in your implementation. **This report is important as this will be used to compute your writing proficiency (WP) points (partial).**

5 points Class diagram

2.5 points Peer Evaluation (Contribution Summary) [**must email individually**]

Major Deductions

Case 1: Your program does not compile. If it does not run, we can't give you any point.

Case 2: Your program compiles but breaks repeatedly. If every time we try to interact with it, it breaks, then we won't be able to test the correctness of the program.

Case 3: Your program runs but does not work correctly. For example, players can move to non-adjacent rooms, players can act in roles above their rank, etc.