CSCI 345–Assignment 3
Implementation

Aran Clauson

Introduction

You analyzed the problem, designed a solution, now it is time to implement the design.

Outcomes

Upon successful completion of this assignment you will

- covert your software design into a working progrm
- demonstrate how the five class relationships are realized in Java
  - Association, Aggregation, Composition, Implementation, and Inheritance
- have an implementation of Deadwood™ that you can play in a terminal.

Problem Statement

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

The program will accept a very limited vocabulary. The following table is the complete language:

command	action
who	The software identifies the current player and any parts that the player is working.
where	The software describes the current player's room and any active scenes.
move *room*	The current player moves to the indicated room.
work *part*	The current player takes the indicated role.
upgrade $ *level*	Upgrade the current player to the indicated level.

upgrade cr *level*	Upgrade the current player to the indicated level.
reherse	The current player rehearses.
act	The current player performs in its current role.
end	End the current player's turn.

Emphasized words denote arguments. The *room* and *part* are string arguments and are the rest of the line. *Level* denotes an integer from two to six.

The **end** command must be provided even if the current player has no legal actions. For example, if the current player is working a part, the interaction could be the following:

```
> who
red ($15, 3cr) working Crusty Prospector, "Aww, peaches!"
> where
in Train Station shooting Law and the Old West scene 20
> act
> end
```

I'm 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 working model, it is not to have a friendly user interface.

For example, consider a player that is not working a part. The interaction might look like the following:

```
> who
blue ($1, 5cr)
> where
Jail wrapped
> move Train Station
> where
Train Station shooting Law and the Old West scene 20
> work Talking Mule
> end
```

How did I know that I could take the Talking Mule part? I looked at the XML file.

You may find that your design 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 as you modify the design. I am going to use this document to understand the 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. While it does not really matter, I suggest naming the players with the dice colors: blue, cyan, green, orange, pink, red, violet, and yellow. These match the dice images on Canvas.

Deliverables

Your code will be submitted via GitLab (`gitlab.cs.wwu.edu`). Make sure that I have *developer* permissions to your repository. Submit the `git` clone command to canvas. I will use this command to download your implementation.

Your repository should be structured as follows:

```
repo-name/
└── src/
    ├── Deadwood.java
    └── ...
```

I don't care what you name the top-level directory. The file `Deadwood.java` contains the main program. That is to say, the following command should start a two-player game of Deadwood™.

```
$ pwd
/path/to/repo-name/src
$ javac Deadwood.java
$ java Deadwood 2
```

Organize the rest of your Java code in whatever way you think is the easiest to understand.

Grading

- Your software correctly plays the game of Deadwood™.
- Quality of your code (e.g., meaning full comments, well defined methods, descriptive symbols, etc).
- How well your design matches your implementation.