SWEN 225 Assignment One

Assumptions

- Users can move around within a room without using up any steps.
- Players can block others from moving into a room by positioning themselves in front of a door.
- We don't show the user the contents of the envelope if they guess incorrectly in case the next person sees.
- Weapon pieces can only be placed in rooms and are off the board to start.
- Character pieces cannot occupy the same cell as a weapon piece.

Design

Our design focusses on delegating sections of the code into portions designed for a specific task. This starts with packages. We read through the description of Cluedo trying to find actions or items that fit neatly together. We concluded that there were four such groupings: cards, moves, pieces, and high-level logic. These groupings lend themselves to a design where high-level logic takes care of the general direction/rules of the game whilst cards, moves, and pieces handle the more specific gameplay.

After splitting the code into packages, we separated the code out further into classes. These classes reflect specific actions, items, or high-level rules (e.g. turnOrder and Envelope) in the game. There were some ideas that clearly lent themselves to abstract classes/interfaces (cards, moves, and pieces). These ideas encompass specific implementations (e.g. character cards, weapon cards, room cards) that are treated similarly by high-level logic (cards are dealt, displayed, etc.). This abstraction was very useful for moves as we could design different move implementations with their own isValid() and apply(Board) methods that could just be treated as a Move by the high-level logic.

Within the classes themselves we tried to split the tasks into general methods with one specific role. By dividing the code like this we significantly increase how much code we can reuse between tasks and dramatically increase readability. This "divide and conquer" strategy also makes solving problems substantially easier. For example, the Cluedo method separates user input into distinct methods for choosing moves, pieces, rooms, etc. but uses the general yesNoQuestion() and getInput() methods to ensure the users' inputs are valid. The code would be a mess to write, read and debug if we hadn't taken this approach.

We set out from the beginning to restrict access as much as possible in order to decrease the likelihood of one of us using the other person's code incorrectly. Some strategies we used were private fields/methods, getters and setters, and "helper" classes to limit scope (e.g. CustomMove).