- Player
- Human
- Computer

Having chosen "Computer" as our advanced requirement, we recognized that there were many similarities between the "Human" and "Computer" players. They will likely execute the same types of moves and actions and share many attributes and methods. In seeing this overlap, we determined that creating a new Abstract entity called "Player" and leveraging the inheritance model for this relationship would be appropriate.

This "Player" entity will capture many of the similar responsibilities between "Human" and "Computer", and thus allow us to only implement them once.

- Action
- StartGameAction
- QuitGameAction
- ChooseOpponentsAction
- MovePieceAction

We also recognized that a "Player" may create various types of actions which are executed by the game. Therefore, we chose to capture these various actions under an abstract class called "Action". This not only eliminates code repetition (adhering to DRY principles), but also reduces the number of dependencies coming from the "Game" class (which would otherwise need to have dependencies with each concrete action class).

- Move
- SetMove
- AdjacentMove
- RemoveMove
- FlyMove

Our reasoning behind the design surrounding "Move" and its concrete classes, follows the same reasoning behind the "Action" classes. With various phases and contexts within "Nine Man's Morris" allowing for different types of movements, we decided that following a command pattern in implementing "Moves" would be very appropriate. There would otherwise be far too many responsibilities consolidated within one class which would need to execute on the game's state (Board, Pieces, Position) across a wide range of outcomes.

- Game

A "Game" class is typical and appropriate for most implementations of this nature, and Nine Man's Morris is no exception. There must be a central point from which all choices and outcomes are coordinated through or executed on. Nine Man's Morris as a concept in our design, cannot itself exist without boards and pieces, which the "Game" is ultimately responsible for initializing. "Game" is also home to all the central logic which bedrocks Nine Man's Morris.

A human player is also not capable of playing the game well, without seeing some visual representation of the game state. Thus, "Game" is logically needed as an intermediary to pass consolidated meaningful information onto "Display" for visual representation (as display by itself knows nothing).

- Display

Where “Game” homes the program’s logic, “Display” homes the game’s graphics. Display’s responsibilities are restricted to only the visual technicalities, hence its separation from the “Game” class. “Display” could be technically over-loaded, as it may setup graphical APIs (Java2d, Swing, SDL), in addition to other display elements. Therefore, it may need to be further compartmentalized and broken to better fit object-oriented design. This will be fleshed out later, when the team becomes more experienced and knowledgeable of the requirement’s nuances.

- Board

The “Board” is an essential component of the Nine Man’s Morris Game. It is important to note the composition relationship between “Board” and “Game”. This is because we likely need to access much of the “Board” class’s behaviors, within the “Game” class. A litmus test of the Liskov Substitution Principle on the “Game” and “Board” relationship (which is failed) can also be used to rule out inheritance as a potential relationship type. Furthermore, thinking intuitively, a “Board” and “Game” relationship is not like a typical inheritance relationship like “Car” and “Vehicle”.

- Piece

Pieces too are essential components of the Nine Man’s Morris game.

- <enum> Mills

We recognized that a key mechanism of Nine Man’s Morris, was that board pieces in “mills” would not be permitted to move or be removed. Therefore, it is important to have some sort of identifier for “Pieces” to hold, to be able to recognize their mill status.

- Position

“Position” will be a class that denotes various board location points. One important design note is that “Position” may have associations with itself, through which its neighbors’ positions are noted. This is potentially helpful in identifying open spots for adjacent moves and chains of pieces which could form mills.