For your final project, you will take the code from the Heroes and Monsters refactoring assignment and add the following features and patterns.

1. The hero is now randomly placed within a dungeon that is a 5 x 5 2D array (bigger is ok). The hero needs to find the four Pillars of OO, and take them to the exit to win the game. Some features of the dungeon will prove a hindrance to the hero's task (pits and monsters), while some will prove helpful (healing potions). Your task is to write a correct and well-designed Java or C# program that will simulate this adventure. NOTE: You are welcome to implement a variation on this theme provided you adhere to the spirit of what is being asked on this assignment.

CLASS DETAILS:

Room.java

- Contains default constructor and all methods you deem necessary -- modular design is CRUCIAL
- Contains the following items/behaviors
    - (Possibly a) Healing Potion - heals 5-15 hit points (this amount will be randomly generated -- you can modify the range)
    - (Possibly a) Pit - damage a pit can cause is from 1-20 hit points (this amount will be randomly generated - you can modify the range)
    - (Possibly an) Entrance - only one room will have an entrance and the room that contains the entrance will contain NOTHING else
    - (Possibly an) Exit - only one room will have an exit and the room that contains the exit will contain NOTHING else
    - (Possibly a) Pillar of OO - four pieces in game and they will never be in the same room
    - (Possibly a) Monster
    - Doors - N, S, E, W
    - 10% possibility (this is a constant that you can modify) room will contain a healing potion, vision potion, and pit (each of these are independent of one another)
    - Vision Potion - can be used to allow user to see eight rooms surrounding current room as well as current room (location in maze may cause less than 8 to be displayed). The Vision Potion allows you to see the rooms that are immediately around you (this is up to eight rooms depending on your location in the dungeon). This potion only lasts for a single turn. Example:

- The hero is currently in room 1,1. If the hero drinks the Vision Potion, then the following rooms are visible for a single turn.

    - Room 0,0 Room 0,1 Room 0,2
      Room 1,0 Room 1,1 Room 1,2
      Room 2,0 Room 2,1 Room 2,2

- Must contain a toString method that builds a 2D Graphical representation of the room (NOTE: you may use any graphical components in Java that you wish). The (command line) representation is as follows:
    - * - * will represent a north/south door (the - represents the door). If the room is on a boundary of the maze (upper or lower), then that will be represented with ***
    - East/west doors will be represented in a similar fashion with the door being the | character as opposed to a -.
    - In the center of the room you will display a letter that represents what the room contains. Here are the letters to use and what they represent:
        - M - Multiple Items
        - P - Pit
        - I - Entrance (In)
        - O - Exit (Out)
        - V - Vision Potion
        - H - Healing Potion
        - E - Empty Room
        - X - Monster

  Example: Room 1,1 might look like
  * - *
  | P |
  * - *

  Room 0,0 might look like
  * * *
  * E |
  * - *

Hero.java (add these new items as necessary)

- ~~Something to allow the hero to be given a name that is used throughout the game~~
- ~~Contains at least the following:~~
    - ~~The number of Healing Potions~~
    - ~~The number of Vision Potions~~
    - ~~The number of Pillars of OO found~~
- Something to move/track hero location around the Dungeon
- ~~Contains a toString method that contains fundamental hero information:~~
    - ~~Name~~
    - ~~HitPoints~~
    - ~~Total Healing Potions~~
    - ~~Total Vision Potions~~
    - ~~Total Pillars of OO Found~~

NOTE: The Hero and the Dungeon (described below) will need to interact. When the Hero walks into a room if there is a potion in the room, the Hero automatically picks up the

potion. Likewise, if there is a pit in the room, the Hero automatically falls in the pit and takes a Hit Point loss. All these changes obviously affect the room. For example, the Hero walks into a room that contains a Healing Potion. The Hero will pick up the potion, changing the Hero's potion total, as well as changing the room's potion total.

Also note that when a hero enters a room, if a monster is in the room they will automatically battle. Provided the hero wins the battle, there is a chance that the monster will drop a healing potion.

Dungeon.java

- ~~Creates/contains a 5 X 5 2D Array of Rooms (you can make this larger if you wish)~~
- Places the Entrance, the Exit, and the Pillars of OO Pieces. NOTES: the entrance and exit are empty rooms. The Pillar pieces cannot be at the entrance or the exit. Pillar pieces must not occur in the same room.
- Maintains location of the Hero in the Dungeon
- Contains a toString method that builds a String containing information about the entire dungeon.

DungeonAdventure.java

- ~~Contains the main method~~
- ~~Provides an introduction to the game describing what the game is about and how to play~~
- Creates a Dungeon Object and a Hero Object (based on user choice)
- Does the following repetitively:
    - Prints the current room (this is based on the Hero's current location)
    - Determines the Hero's options (Move, Use a Potion)
    - Continues this process until the Hero wins or dies
    - NOTE: Include a hidden menu option for testing that prints out the entire Dungeon -- specify what the menu option is in your documentation for the DungeonAdventure class
- At the end of the game, display the entire Dungeon

WORKING AS A TEAM:

You will work in teams of up to three students. You will turn in one assignment per team with the DungeonAdventure class containing a **DETAILED** account of each team member's contribution. There is enough work to go around where each team member should be responsible for a class. Note that you must use git and GitHub to version your code and coordinate with one-another. In addition, your code should have unit tests applied to all methods other than basic getters.

Additional specifications

1. Create a HeroFactory and a MonsterFactory. These are simple factories that have a createHero and createMonster method, respectively. The create methods will be passed necessary information for the factory to build an appropriate hero or monster.
2. Utilize the Strategy pattern to build an Attack interface. Create appropriate concrete classes that represent different ways to attack based on the original specifications of the game. All DungeonCharacters should utilize the Attack interface. Use the Flyweight pattern to store a pool of all attacks. When constructing a hero or monster, an appropriate attack should be assigned from the Flyweight pool.
3. (EXTRA CREDIT – 5 points – be sure and document you did this): Utilize the Memento pattern to allow the user to save the state of the game. As part of saving the state, you should write the saved state to a file. Provide a means to retrieve that state (load a saved game) as part of the game functionality. Note that using Serialization (Serializable in Java) can make this process very simple.
4. Add (at least) two more heroes and two more monsters to the game. Assign attributes and behaviors however you wish.
5. The type of interface you use is up to you. A console interface is fine, but you are welcome to make a GUI version if you wish. If you choose to make things GUI, don't spend too much time on it -- make sure the basic infrastructure above is taken care of first. All the above items really constitute the Model for your game. Your interface is your View and should be kept separate from what the Model is going if possible. You could utilize a Controller to handle the interactions between the two.
6. Place all files in a package named **dungeon**.

TO TURN IN

1. A UML Diagram that represents your solution (.pdf format)
2. All source code
3. Capture/printout of the log of history using git/GitHub
4. Captures of output from running your program that demonstrate program correctness. The output captures do not need to be exhaustive, just provide enough to show the basics of your program are in place. If your program is console-based, the output capture can be placed in a plain text file. If you create a GUI interface, then include screen shots in .pdf format.
5. Place the above three items in a zip file named with team member last names followed by finalproject349 (e.g. capaulsteinerlemelinfinalproject349.zip). Just one team member needs to submit the project – be sure it is someone trustworthy!

GET STARTED ASAP!