DUNGEON ADVENTURE

PROGRAM PURPOSE:

The purpose of this is assignment is for you to practice problem solving, working more on teams, working with multiple classes, maze generation, and maze traversal. You will optionally be able to incorporate GUI components as part of your solution.

PROGRAM DETAIL:

This is an adventure game where a hero is randomly placed within a dungeon, which is randomly generated. The adventurer needs to find the four Pillars of OO (Abstraction, Encapsulation, Inheritance, and Polymorphism) and take them to the exit to win the game. Some features of the dungeon will prove a hindrance to the adventurer's task (pits), while some will prove helpful (healing and vision potions). Your task is to write a correct and well documented Python 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

- 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 pillars in game and they will never be in the same room
- 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)

- Must contain a _ _ str _ _ () method that builds a 2D Graphical representation of the room (NOTE: you may use any graphical components 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
    - X - Pit
    - i - Entrance (In)
    - O - Exit (Out)
    - V - Vision Potion
    - H - Healing Potion
    - <space> - Empty Room
    - A, E, I, P - Pillars

Example: Room 1,1 might look like

```
* - *
| P |
* - *
```

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

Adventurer.java

- Has a name
- Contains at least the following:
  - Hit Points - initially set to 75 - 100 upon creation (randomly generate - you can change the range)
  - The number of Healing Potions
  - The number of Vision Potions
  - The which Pillars found
- Ability to move in Dungeon (you might decide to place this behavior elsewhere)
- Increases or decreases the Hit Points accordingly
- Contains a _ _ str _ _ () method that builds a String containing:
  - Name
  - Hit Points
  - Total Healing Potions
  - Total Vision Potions
  - List of Pillars Pieces Found

NOTE: The Adventurer and the Dungeon will need to interact. When the Adventurer walks into a room if there is a potion in the room, the Adventurer automatically picks up the potion. Likewise if there is a pit in the room, the Adventurer automatically falls in the pit and takes a Hit Point loss. These changes obviously affect the room. For example, the Adventurer walks into a room that contains a Healing Potion. The Adventurer will pick up the potion, changing the Adventurer's potion total, as well as changing the room's potion total.

Dungeon

- Creates/contains a maze of Rooms
  - The maze should be randomly generated
    - You must incorporate an algorithm to ensure traversal of the maze from entrance to exit is possible once the maze has been generated. If the maze is not traversable, then generate a new one 😊

    - The maze should have ‘dead ends’ (places that lead no further)
    - The type of data structure you use to represent your maze is up to you
        - A list of lists has some advantages
        - A linked list of linked lists has other advantages
        - You could represent your maze via rooms that have references to other rooms (this would be much like a linked list structure but without all the basic linked list functionality which you do not need for this assignment)
- Places the Entrance, the Exit, and the Pillars. NOTES: the entrance and exit are empty rooms. The Pillars cannot be at the entrance or the exit. No two Pillars may be in the same room.
- (Possibly) Maintains location of the Adventurer in the Dungeon
- Contains a _ _ str _ _ () method that builds a String containing information about the entire dungeon.

DungeonAdventure

- Contains the main logic for playing the game
- Introduces the game describing what the game is about and how to play
- Creates a Dungeon Object and a Adventurer Object
- Obtains the name of the adventurer from the user
- Does the following repetitively:
    - Prints the current room (this is based on the Adventurer's current location)
    - Determines the Adventurer's options (Move, Use a Potion)
    - Continues this process until the Adventurer 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 conclusion of the game, display the entire Dungeon

WORKING AS A TEAM:

You will turn in one assignment per team with readme.txt 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.

Extra credit (up to 10%) MAY be given for additional features, originality, creativity, and well-designed code. This is at the discretion of the grader (your instructor) so is not guaranteed. If you feel you did things that warrant extra credit, clearly describe these things as part of your documentation.

Extra credit GUI (up to 10%): You can earn an additional 10% for utilizing a GUI interface for your program.

TO TURN IN:

- All zip file named based on the team member last names
- All Python source files (PyCharm project folder is fine, too)
- A UML class diagram that represents the classes in your solution (and their relations to one-another)
- Output captures (or tests) that show
  - Maze generation works (generates a passable maze, if impassable maze is generated then a new maze is generated)
  - Example of losing game
  - Example of winning game
  - NOTE: for these tests you can (should?) make your maze smaller to test things more easily.
- Readme.txt that describes
  - what work each person did
  - estimate of time spent on project
  - any shortcomings the project has
  - any other information you might think is useful for grading

Have fun! Both of your instructors (Kevin and Tom) look forward to playing your game!