Project Details: Option 2 - Dungeon Adventure

Your program must incorporate the MVC design pattern.

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 Java 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.

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 the 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 is independent of the other)
    - Vision Potion - can be used to allow the user to see eight rooms surrounding the current room as well as the current room (location in the maze may cause less than 8 to be displayed)
- Must contain a toString() method that builds a 2D Graphical representation of the room (NOTE: you may use any graphical components that you wish). The (command line) representation MIGHT BE 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

```
***
*i|
*-*
```

Adventurer (Hero)

- 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 toString() method that builds a String containing:
    - Name
    - Hit Points
    - Total Healing Potions
    - Total Vision Potions
    - List of Pillars Pieces Found

NOTE: The Adventurer/Hero 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 (probably a 2D array, but doesn't have to be)
- 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) o The type of data structure you use to represent your maze is up to you
- 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 the location of the Adventurer in the Dungeon
- Contains a toString() method that builds a String containing information about the entire dungeon.

DungeonAdventure

- Contains the main logic for playing the game. It controls gameplay.
- Introduces the game describing what the game is about and how to play
- Creates a Dungeon Object and an 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
- Add an inheritance hierarchy of dungeon characters. These classes will be used to represent the hero (player) and monsters
    - DungeonCharacter is the parent/super class for the hierarchy. It should have the following characteristics
        - is abstract
        - constructor for initializing all fields provided by the class
        - character name
        - health points/hit points
        - damage range (min and max)
        - attack speed (1 is the slowest)
            - when battling the attack speeds of the two opponents will be compared
            - a character can get multiple attacks per round of battle based on speed: a character that is twice as fast gets two attacks per round, a character that is three times as fast gets three attacks, etc.
            - you have the freedom to adjust how this works
        - chance to hit (when attacking the opponent)
        - gets/sets as necessary for accessing/changing fields
        - an attack behavior (method)
            - this method passed the opponent to attack
            - if a character can attack (based on chance to hit), damage is generated in the min to max range for the character and applied to the opponent
            - provide a means to report success of attack or failure as necessary (this might be done inside the method or elsewhere depending on your design)
        - anything else you deem necessary (be creative and have fun :-)
    - Hero
        - Inherits from DungeonCharacter
        - Is abstract
        - A hero never gets fewer attacks than a monster (you can change this if you wish)
        - A hero has a chance to block an attack. This can be an integer or double.

        - Has a constructor that initializes all fields specific to Hero and calls the DungeonCharacter constructor to initialize the fields defined in DungeonCharacter
        - Heroes have a regular attack and also a special skill (skills for specific heroes will be defined below)
        - Any other fields or methods you deem necessary
    - Warrior
        - Inherits from Hero
        - Special skill is Crushing Blow that does 75 to 175 points of damage but only has a 40% chance of succeeding (you can adjust all numbers)
        - gets, sets, and any other methods you deem necessary (you may want to override the **attack** method to fit your Warrior – or not)
        - suggested statistics for Warrior (should be set up in constructor(s))
            - hit points: 125
            - attack speed: 4
            - chance to hit: 0.8 (80 percent)
            - minimum damage: 35
            - maximum damage: 60
            - chance to block: 0.2 (20 percent)
    - Priestess
        - Inherits from Hero
        - special skill is **heal** (choose a range of hit points that will be healed)
        - suggested statistics for Priestess (should be set up in constructor(s))
            - hit points: 75
            - attack speed: 5
            - chance to hit: 0.7 (70 percent)
            - minimum damage: 25
            - maximum damage: 45
            - chance to block: 0.3 (30 percent)
        - any other fields and methods you deem necessary
    - Thief
        - Inherits from Hero
        - Special skill is **surprise attack** -- 40 percent chance it is successful. If it is successful, Thief gets an attack and another turn (extra attack) in the current round. There is a 20 percent chance the Thief is caught in which case no attack at all is rendered. The other 40 percent is just a normal attack.
        - suggested statistics for Thief (should be set up in constructor(s))
            - hit points: 75
            - attack speed: 6
            - chance to hit: 0.8 (80 percent)
            - minimum damage: 20
            - maximum damage: 40
            - chance to block: 0.4 (40 percent)
        - any other fields and methods you deem necessary

- Monster
  - Inherits from DungeonCharacter
  - Is abstract
  - constructor - should call base/super constructor
  - get, set, and any other methods (this includes overridden ones) you deem necessary
  - a **heal** method that is based on chance to heal and then range of heal points for monster
  - chance to heal (a Monster has a chance to heal after any attack that causes a loss of hit points -- this should be checked after the Monster has been attacked and hit points have been lost -- note that if the hit points lost cause the Monster to faint, it cannot heal itself!)
- Ogre
  - Inherits from Monster
  - instance variables as you deem necessary (none may be necessary!)
  - gets, sets, and any other methods you deem necessary (you may want to override the **attack** method to fit your Ogre – or not)
  - suggested statistics for Ogre (should be set up in constructor(s) -- choose a name for your Ogre)
    - hit points: 200
    - attack speed: 2
    - chance to hit: 0.6 (60 percent)
    - minimum damage: 30
    - maximum damage: 60
    - chance to heal: 0.1 (10 percent)
    - minimum heal points: 30
    - maximum heal points: 60
- Gremlin
  - Inherits from Monster
  - instance variables as you deem necessary (none may be necessary!)
  - gets, sets, and any other methods you deem necessary (you may want to override the **attack** method to fit your Gremlin)
  - suggested statistics for Gremlin (should be set up in constructor(s)-- choose a name for your Gremlin)
    - hit points: 70
    - attack speed: 5
    - chance to hit: 0.8 (80 percent)
    - minimum damage: 15
    - maximum damage: 30
    - chance to heal: 0.4 (40 percent)
    - minimum heal points: 20
    - maximum heal points: 40
- Skeleton
  - Inherits from Monster

- instance variables as you deem necessary (none may be necessary!)
- gets, sets, and any other methods you deem necessary (you may want to override the **attack** method to fit your Skeleton)
- suggested statistics for Skeleton (should be set up in constructor(s)-- choose a name for your Skeleton)
    - hit points: 100
    - attack speed: 3
    - chance to hit: 0.8 (80 percent)
    - minimum damage: 30
    - maximum damage: 50
    - chance to heal: 0.3 (30 percent)
    - minimum heal points: 30
    - maximum heal points: 50

Game Play

- Player chooses a Hero (the game will ask the user for the name of the hero)
- Monsters are randomly placed in rooms of the dungeon
- Stronger/special/more monsters should be placed with pillars and exit
- Previous rules for Dungeon Adventure are still in place but you can modify things based on your team's vision for the game
- Provide the ability to save and load a game
    - you can provide a single save, multiple saves, let the user choose the names of the save files, whatever you deem best :-)
    - serialization should be used to save if possible (it will make life much easier on you if you can get it to work)
- Once the game is over provide the ability for the player to start a new game
- Store data for your monsters in a SQLite database (Monster name and statistics that go with that monster)
    - Retrieve this data at the start of your program and use it to generate monsters as you place them in the dungeon
- Extra Credit possibilities
    - Creativity
    - good OO (use of patterns where it makes sense)
    - difficulty levels
    - audio/video
    - custom assets (images, etc.)
    - 3D maze
    - Multiple heroes (a party)
    - Additional potion types (perhaps a bomb that can be used for massive damage against a monster)
    - ???

Project Deliverables

- A zip file with the following items
    - Source file solution for your project (Eclipse/IntelliJ folder)
        - Code should have thorough Javadoc comments
        - There should be a robust set of unit tests
        - Be sure support files (pictures, media) are included in folder
    - UML class diagram that represents your solution (.pdf format)
        - Final version of SRS (.pdf format)
        - List/capture of all user stories on Pivotal Tracker (those completed as well as those that are unfinished) (.pdf format)
        - Log of commit history of git/GitHub for all team members (.pdf format)
        - A project synopsis that contains the following (.pdf format)
            - team member names
            - breakdown of what each person worked on
            - total hours contributed by each person on project (refer to Toggl for this information)
            - discussion of problems you had to overcome
            - discussion of shortcomings your project has (if there aren't any state so)
            - discussion of items you feel should be considered for extra credit on project
    - NOTE: Extra credit is available for anything above and beyond the basic project requirements as specified previously. Other possible items include:
        - an installer program
        - posting program to cloud
        - ??

Rubric

Project Deliverables		
Criteria	**Ratings**	**Pts**
Misc items not covered in other criteria		30 pts
SQLite used in project, properly wired so it works when graded on instructor's machine		20 pts
Save/Load ability to save/load (note: serialization not required, but is nice if you can make it work)		15 pts
SRS Updated version of draft that reflects final version of project		20 pts
UML class diagram Properly represents final set of classes, packages, and interfaces in project. Uses proper UML symbols as introduced in class.		15 pts
Unit tests Thorough tests for the Model portion of code. View and controller tests are nice but not required.		30 pts
Pivotal Tracker Used properly to plan iterations (user stories with points applied, user stories checked off as complete when finished, etc.)		10 pts
git history A clear history of activity (commits) to the GitHub repository.		10 pts
Project synopsis Contains all details specified on project specs (breakdown of work done, list of issues/concerns, total hours for project via Toggl including breakdown of work done, future work plans, etc.). List of extra credit features.		30 pts
Basic functionality Fulfills all basic requirements (minimal game play specs, utilizes MVC, naming conventions followed, modular design that includes highly cohesive code and decoupled classes and methods, thorough Javadoc comments, etc.)		70 pts
Extra credit -audio -video -extra features/design -installer		0 pts
		Total Points: 250