RPG Game

(Inheritance, polymorphism)

In this homework we are going to continue building upon the previous one utilizing the concepts of inheritance and polymorphism, as we have seen, the classes player and enemy share many instance variables and methods, so it is convenient to derive them from a base class called **Character**.

So your first task is to separate the shared variables and methods in a Parent **abstract** class called **Character** and drive the **Player** and **Enemy** classes from it. Identify the abstract methods in the **Character** class and give default implementation to the other methods. Override the abstract methods in the subclasses.

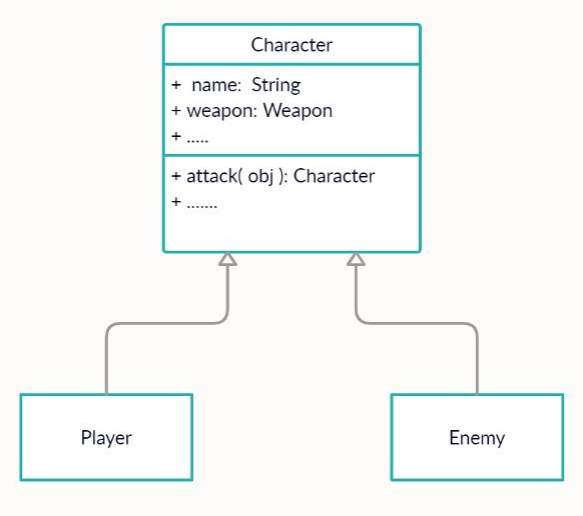
Gold Modification

Add an integer instance variable to the Player class that keeps track of the player's amount of gold. After each battle, generate a random gold reward that the player receives, in addition to the experience point award. Be sure to also modify the Player viewStats() method to display the current amount of gold the player owns.

The Shield Class

The Shield class is a new instance variable added to the enemy class, it will give the enemy a health boost.
Class variables:

1) **String type**: the type of the shield. *"Metallic, wooden.. etc"*
2) **int resistance**: this value is used to boost the enemy's health *"total =enemy health + resistance"*. You should modify the enemy takeDamage method in order to take the shield resistance into account. The takeDamage(int damage) method must decrease the resistance first, the shield is considered broken when the resistance value reaches 0. Also be sure to include the resistance value when printing the enemy's health at the end of each battle.



**Note that the attack method receives an object of type Character.*

Copy constructor and Equals method

Implement the Copy constructor and equals methods in the player and enemy classes following good OOP practices, without privacy leaks.

Multiple Enemies

Before the fight starts there is a 30% chance that the enemy copies itself up to 4 times.
Modify the game so that the player can encounter several enemies at once. Update the Map **checkForEnemies**() method to return a **Character** array. Overload the Player attack method to instead receive an array of enemies **attack**(**Character**[ ] enemies). The method applies damage to the *first enemy* in the array for now.

You encountered a Zombie, a Zombie!
1) Attack, 2) Run: 1

You attack a zombie with a Sword
You attack for 9 damage!

A zombie attacks you with a dagger
You are hit for 4 damage!

A zombie attacks you with a dagger
You are hit for 2 damage!

Bob's health = 16
Zombie's health = 11
Dark soul's health = 20

Bomb Class

In order to apply damage to multiple enemies at once it makes sense to have a Bomb weapon. The bomb is a subclass of the weapon class and has one additional variable called numberOfEnemies, this number will specify the number of enemies affected by the bomb. The damage on each enemy is then calculated as usual using the damageRange instance variable.

Class variable:
Int numberOfEnemies;

Upon constructing the player object you are free to choose between a basic weapon or a bomb.
Note that the Player attack method should check the weapon's type, in case of a bomb the damage should be applied to a certain number of the enemies specified by the bomb's *numberOfEnemies*.

// numberOfEnemies = 2
You encountered a Zombie, a Zombie, an Zombie!
1) Attack, 2) Run: 1

You attack a zombie with a Bomb
You attack for 9 damage!

You attack a zombie with a Bomb
You attack for 6 damage!

A zombie attacks you with a dagger
You are hit for 4 damage!

A zombie attacks you with a dagger
You are hit for 5 damage!

An zombie attacks you with a dagger
You are hit for 5 damage!
…….

Grid Map

Our simple map allowed the player to move freely in the environment by incrementing the position values. In this homework we are going to create a GridMap class that extends the Map class. The 2D grid is defined with two variables **gridRows** and **gridColumns**, each cell in this grid is a location the player can move to. Some locations are free but others are occupied with obstacles. We are going to provide the game with an obstacle map which is an integer array like:

Class variables:

1) **int gridRows;**
2) **int gridColumns;**
3) **int[ ] obstacleMap =** { 0, 0, 0, 0,
   0, 1, 0, 0,
   0, 0, 1, 0, // number of rows and columns are equal to 4
   0, 1, 0, 0 };

The player is free to move to locations marked by **0**, locations marked by **1** contain obstacles, therefore the player can not move to those locations. Override the move() method in GridMap class to navigate the grid taking the obstacle map into account. You should notify the Player if he/she tries to move to an obstacle or reaches the boundaries of the obstacleMap. The playerXPos and playerYPos respectively represent the current column and row number of the player.

Configuration file

So far we were hardcoding the values needed to create the entities of our game, a more robust way to initialize any game is to have a configuration file which contains the necessary data to construct the game objects.

1. As a beginning we are going to read the values required to create the GridMap object from a configuration file located in the game directory.
2. Add the method " String readFile(String filename) " to the **Utility** class, the method should open an input stream (Scanner or BufferedReader), read the configuration text from the file and return a string.
3. Upon constructing the GridMap object this method should be called, and the returned string should be evaluated.

The obstacle configuration data format:

4 3 : 0 0 0 : 0 1 0 : 0 0 1 : 0 1 0

Where 4 is the number of rows, 3 is the number of columns, the rest of the numbers represent the elements of the obstacle map.

As we have mentioned above, this text should be evaluated during the GridMap object creation, You need to throw your own exceptions with proper messages and catch them in the main method of the GameDemo class.

Some error cases:

4 3 : 0 0 0 1 : 0 1 0 : 0 0 1 : 0 1 0 throw ConfigurationFormatException *message 'wrong number of columns'*

4 3 : 0 0 0 : 0 4 0 : 0 0 1 : 0 1 0 throw InvalidConfigurationException *message 'incorrect value'*

…. etc