ScreenManager class
A class that is created and called when the program starts in order to set up the maps and current rooms loaded in addition to moving between the screens.

This class implements functions that print each screen and determines what will be displayed on those particular screens. When the program is first started, the ScreenManager will be created and display the main menu, awaiting input from the player to move to the next screen depending on the input. Depending on the input, corresponding objects will be created or updated then the next screen will be printed such as the roomIdle screen, battle menu, or inventory. This class will manage the main inputs, prompting the player to input their choice, then take the input and pass it to other classes such as the Map, Room, or Inventory classes.

Map class
There is only one Map class present during the program which will contain a vector of Room objects that will be the line of Rooms the player will have to travel through in addition to an index indicating the current Room the player is in.

This Map class implements functions including displayMap which will be called during a screen transition and will iterate through all Rooms in the Room vector to determine if they are cleared, unknown, or the current room. These states can then be printed to the screen in singular boxes that represent the Rooms. This class also implements a moveRooms function that is called when the player goes to the next or previous room so the currentRoom variable is updated in preparation for the next displayMap call.

Room class
A Room has a vector of Enemy pointers which will contain all the current enemies located in that Room. The Room will also be created with a vector of Item pointers that contains the available items in the Room that the player will be able to find by choosing to look around the Room. Finally, the Room has a clear boolean variable which will indicate if the Room has enemies and is safe or not, and contains a string called roomInfo that contains unique information about the Room that the player can examine.

The Room will be constructed with an imputed roomInfo as well as a randomly generated number of enemies and items once the player enters the Room. Right after the Room is constructed, startBattle will be called which will create a BattleManager and proceed to the battle menu which will continue until the Enemy pointer vector is empty and the Room will be set to clear. The lookAroundRoom function will randomly select an item from the Item pointer vector and give a success chance for the player to obtain the item. Finally, the getRoomInfo function is implemented by printing the roomInfo string to the screen, then prompting the player for the next choice of action.

ItemDatabase class
Only one ItemDatabase is present in the program that will hold all programmed items in the game. All items are contained within a two-dimensional vector, with each row containing a singular item and each column containing a property of that item including the name, description, property, and idNum.

To access the items in this database, the returnItem function is implemented by taking two integers as parameters, the first indicates the row, or item, and the second indicates the property, or column, which returns a string that can then be converted to other data types when an Item is constructed for the player to obtain.

Item base class
An abstract base class that other items of three types, Weapon, Usable, and Equip, will inherit from and contains basic information that all items will have including its name, description, property, and its idNum.

It has a set of getter functions to return all of these internal values for each item and no setters are required as these values for items should not be changed after the item is constructed.

Weapon class that inherits from Item
A Weapon has an integer that represents damage. A Weapon will be constructed similarly to its inherited Item having a name, description, and property which will influence the damage number.

A Weapon will implement the totalDamage function which takes an Enemy* as an input which will then subtract the damage value from the weapon from that Enemy's health. A chooseWeapon function will also be implemented so a specific Weapon object is indicated from the inventory so only one Weapon damage is applied.

Equip class that inherits from Item
A Equip has a boolean equipped that will indicate whether that item is equipped to the Player or not and will influence whether the player gets a boosted stat or not.

IncrStat and decrStat functions are implemented that check if the item is equipped to the passed in Player, and if they are, then the incrStat function will increase the corresponding stat and the decrStat function will decrease the corresponding stat. The equipItem and unequipItem functions will set the value of the equipped boolean based on the player inputs in the inventory menu.

Usable class that inherits from Item
A Usable will have a statChanged integer in addition to the other variables set by the inherited Item that will indicate which stat will be affected by the item being used. The amount that will be affected is determined by the property integer in the inherited Item class.

The useItem function is implemented by passing in a Player object which will then take the statChanged integer to determine which stat to update on the Player and the value changed will be determined from the property integer.

Entity base class
The Entity class is an abstract base class which both the Player class and Enemies will inherit from. This class has both standard variables for statistics such as health, defense, attack, and speed as well as implementations for getter and setter functions for those variables that inherited objects will have.

The implemented functions will include getters for all individual statistic variables and one setter function that will take in arguments for each of the statistics gathered from player's choices or equipable times. An updateHealth function will also take an integer as an argument which will be called any time damage is dealt or a heal item is used and will update the Entity's health accordingly.

Player class that inherits from Entity
The Player class has a string name that will be input upon construction of the object in addition to an Inventory that is linked to the Player which they will be able to access through this item. In addition to other statistics, the Player also has maximum and current magic through integers that will be the cost for certain items.

The Player constructor takes the Player name as the only argument to initialize the Player object. Getter functions are implemented in order to retrieve any extra statistics that would be needed to be used in battle functions and a setter is implemented through the updateStat function which takes two integers as arguments indicating the stat that would be changed and the value that it is changed by. Finally, the chooseAction function is implemented by allowing input for the Player to choose an action from accessing their inventory, to attacking with a weapon, or attempting to flee.

Enemy abstract class that inherits from Entity
The Enemy class is an abstract base class that all enemies will inherit from. This class contains the functions and variables that all enemies will need to have. The constructor will accept a name from the player that determines which enemy to fight next.
Each enemy will have its own name (Witch, Spider, Golem, Skeleton, and Boss). Each enemy also contains their own number of actions to execute while fighting the player. The number of actions is determined by the level/area of the dungeon the player is located at. The enemy's action will be determined by the player's input. Each enemy will have their own unique "expValue" which is what increases the player's abilities every time an enemy is defeated. Each enemy gives its own unique "expValue" amount. The higher level the enemy, the more "expValue" the player will receive.

The implementation functions will be two getter functions and one virtual function that each enemy will uniquely use. Enemies will use the same getter functions to return the enemy's name and expValue when called. Enemies will use the same "action" virtual function to combat with the player. Action is unique for each enemy.

Witch class that inherits from Enemy
The Witch has an integer magic that is initialized from the constructor which takes the maximum magic as an integer argument.

A Witch has an implementation of the action where it has a certain number of possible actions and one is randomly chosen. Then appropriate damage is applied to the Player with the action being output during the turn output.

Spider class that inherits from Enemy
The Spider inherits from the Enemy class which will construct based on the enemy name, numActions, and exp value input.

A Spider has an implementation of the action where it has a certain number of possible actions and one is randomly chosen. Then appropriate damage is applied to the Player with the action being output during the turn output.

Golem class that inherits from Enemy
The Golem inherits from the Enemy class which will construct based on the enemy name, numActions, and exp value input.

A Golem has an implementation of the action where it has a certain number of possible actions and one is randomly chosen. Then appropriate damage is applied to the Player with the action being output during the turn output.

Skeleton class that inherits from Enemy
The Skeleton inherits from the Enemy class which will construct based on the enemy name, numActions, and exp value input.

A Skeleton has an implementation of the action where it has a certain number of possible actions and one is randomly chosen. Then appropriate damage is applied to the Player with the action being output during the turn output.

Boss class that inherits from Enemy
Gets the Boss stats, generates random numbers associated with Boss's move kit. Could be an attack against the player, something that makes the boss stronger. Then says some dialogue to the player through the implementation of the displayDialogue function which outputs text to the screen.

A Boss has an implementation of the action where it has a certain number of possible actions and one is randomly chosen. Then appropriate damage is applied to the Player with the action being output during the turn output.

Inventory class
The Inventory class has a vector of Item pointers which will be sorted alphabetically in addition to a vector of Equip objects that will represent the currently equipped Items the Player has. The integer max items will be a limit on the size of the inventory, and the current number of items will be stored in the integer numItems.

The Inventory class displays the players inventory, which will be sorted alphabetically through the implementation of displayInventory which will help print to the screen. Then there will be slots where the player can equip items to bring to battle with them through the selectItem where they can view the descriptions and choose certain items through output on the screen.

BattleManager
The BattleManager class has a vector of Entity pointers which is a list of all Entities (Player and Enemies) currently in battle which are sorted by their speed stat. It also has an integer numTurns to indicate how many turns have passed in the battle.

The startTurn function is implemented by taking the action from the argument Player and going through the list of sorted Entities, performing their actions in that specific order. The getPlayerInput will assist in prompting and printing choices for the player in battle and taking that input and applying it accordingly for actions, inventory, or fleeing.