In this first Section of the ReadMe we will analyze the program by investigating what happens immediately upon runtime:

Part 1: The BootStrapper:

When the Project is actually run, the only file that will directly be run by clicking the button is the "DungeonCrawlerOther" file, if you will notice it is actually the only file in the entire package that actually has a main function. Upon running this File, the main function will begin execution. The EventQueue part may be ignored, the first important thing that happens is that DungeonCrawlerOther creates an instance of itself called "game", by calling the constructor.

JFrame Explanation: If you notice, at the beginning of the DungeonCrawlerOther it extends JFrame. "extends" means that it is a subclass of this class "JFrame". Being a subclass of something means that the subclass inherits all of the functions and properties of the superclass. So in our case, our "game" object will be a JFrame, with a little added on. The reason it is important that our "game" object is a JFrame. Is because JFrame is the class that Java uses to represent User Windows like the ones you see any time you open a customer program like Word or Google Chrome, that come with a close, minimize, and maximize button in the top right.

Now back to tracing the execution;The Constructor for our freshly created "game" window will now create the "GameScreen", which is the true core of our project. The creation of GameScreen will set off a long chain of events that is described in Parts 2-5, and after this entire chain of events has finished occuring, DungeonCrawlerOther will do the following:

It will add the created GameScreen onto the "game" window meaning that now our Game Screen is visible to the user on the game window, and the user can begin interacting with it through the "game" window. after this some small settings related to the window are configured, we give permission for the user to resize the window, And the title of the Actual Window is set. After this point, this BootStrapper (DungeonCrawlerOther) Class becomes irrelevant and is never used or referred to again, it's only importance is to bring the game into existence.

Part 2: GameScreen:

So Far, When the project is run, the bootstrapper file(DungeonCrawlerOther) is executed, and a new instance of GameScreen is created, GameScreen's constructor is run.

The "myBufferedDimension" variable is initialized, this variable determines how large the bufferedImage will be. A "StatusScreen" object is created, which for now just contains a string with a text staus. Then the Listeners for Key and Mouse Events on the Game Screen are set up, so that the GameScreen will know when a user has clicked on the screen, or when the user has pressed a key while focused on the game window. The background color is set to black, amd the screen is made focusable. Then a BufferedImage object is created, called "buffImage".

"buffImage" Explanation: "buffImage" is the image that we generate before actually changing anything on the screen, the user can not see it, it is purely internal. Once all the necessary things are drawn onto this "buffImage" we then plaster it on the screen, and clear the "buffImage" so that we can draw in any changes that may occur in the game.

Now back to the execution; Once "buffImage" is made, we also create an object "gbi" which is the Graphics object derived from "buffImage"

"gbi" Explanation: In java, you cannot modify Images or Buffered Images directly, which is why they have a Graphics object associated with them, that lets you modify them. By running the various functions of gbi, such as functions that let you draw Strings, rectangles, and other Images. We will be able to draw onto, and make changes to "buffImage" which will eventually be displayed on the actual screen.

Now back to the execution; Finally a GameStateManager is created, meaning the constructor is called, with the argument "this" because we are actually going to pass the GameScreen object that we have just finished constructing, to the GameStateManager.

Part 3: GameStateManager:

So far, upon running the project, the bootStrapper created a JFrame, which added a JPanel to itself known as the "GameScreen", and now the "GameScreen" is creating the GameStateManager.

First an array "GameStates" is made that contains the pointers to each of the available Game States.

"GameState" Explanation: GameState is an Abstract class that I created, that means it's only purpose is to be a Super Class, meaning that I can not make any objects that are only GameStates, they have to be a type of GameState, InventoryState is a type of (AKA a subclass of) GameState, MainMenuState is also a type of GameState. GameStates come prepackaged with some essential methods for any state such as draw(), keypressed() et. Cetera, but the bodies of these methods for all the different types of GameState are different. This is important because It means I can broadly make a command to draw, and without me knowing which State the game is currently in, the game will be able to redraw the screen based on whatever state it is in, even though they all have different draw functions.

Now back to the execution; The first element added to the array is a newly created GameState Object that is of the GameEngine type, this can be considered the "ADVENTURESTATE" object, even though right now it is not labeled as such, it is just the zero'th element in the gameStates array. In order to create the GameEngine object, the Constructor is called with the GameScreen and the GameStateManager as parameters.

Part 4: GameEngine(may later be known as AdventureState):

So far, upon running the project, the bootStrapper created a JFrame, which added a JPanel to itself known as the "GameScreen", and now the "GameScreen" is creating the GameStateManager, which is in turn creating our GameEngine/AdventureState.

First the DungeonColumns, and DungeonRows variables are being populated with how many rows and columns we would like our current level to have. Then we take the GameScreen that was passed in the arguments, and store it in a variable, so that the GameEngine has access to it. Next we set up an array called "myTiles", which will consist of Pointers to all the Tiles that will be present in the current level.

"Tile" Explanation: Tile is a class that I made that represents Tiles of the dungeons, and has the corresponding data.

Next we take the GameStateManager that was passed, and put it into a varibale, so that the GameEngine has access to it (In order to do things such as switching to a different State/Menu). Next we populate "myTiles" by creating all the tiles and putting them in the array. After this we create a new Unit, who is called "myHero", which will be placed at 0,0 of myTiles (very soon we will have an explicit Hero Class which will extend Unit). Then we create two new walls which will be placed at 2,2 and 3,3 of myTiles, and then we create the StatusScreen "myStatus" which currently just contains a String with a brief Textual Status.

Part 5 (Last Part): GameStateManager Continued:

So far, upon running the project, the bootStrapper created a JFrame, which added a JPanel to itself known as the "GameScreen", and now the "GameScreen" is creating the GameStateManager, which is in turn created our GameEngine/AdventureState. The GameStateManager will now finish constructing.

Now that the First State is made, we construct and Add two more Game States to our "GameStates" array. These are the InventoryState, and the MainMenuState. Once these are created, and added, we set the "currentState" to ADVENTURESTATE, which invloves actually changing the variable, then drawing the Tile Map and Status Screen onto the "buffImage", and then plasteting the "buffImage" onto the GameScreen, by calling the repaint() function of our GameScreen JPanel.

Act 1 Conclusion:

After all of this has happened, the runtime thread of execution is finally over, the Program does not do anything else indepently at this point, now all the objects are in memory, and the program is essentially on idle, waiting for stimuli from the user, in order to modify these objects which will in turn move the game forward.

Here is the basic flow of events:

Project Starts

DungeonCrawlerOther calls it's own constructor

GameScreen's Constructor gets called

GameStateManager's Constructor gets called

GameEngine's Constructor gets called

GameEngine's Constructor Finishes

InventoryState's Constructor gets called

InventoryState's Constructor Finishes

MainMenuState's Constructor gets called

MainMenuState's Constructor Finishes

GameEngine/ADVENTURESTATE is set as the current State

GameStateManager's Constructor Finishes

GameScreen's Constructor Finishes

DungeonCrawlerOther adds the GameScreen into the user Window

DungeonCrawlerOther, does some minor configurations

DungeonCrawlerOther's Constructor Finishes

DungeonCrawlerOther sets the instance of itself (AKA the user Window) to be visible.

Act Two: Reacting to Stimuli

Because the GameScreen has both a Key Listener, and a Mouse Listener, when ever a key is pressed or the mouse is clicked, the keyPressed or mousePressed functions of the GameScreen are called, and in the bodies of these functions, the GameStateManager's corresponding functions are invoked, and the GameStateManager, will have the current GameState react to the Key Press however it is supposed to, and then draw the corresponding changes on the screen.