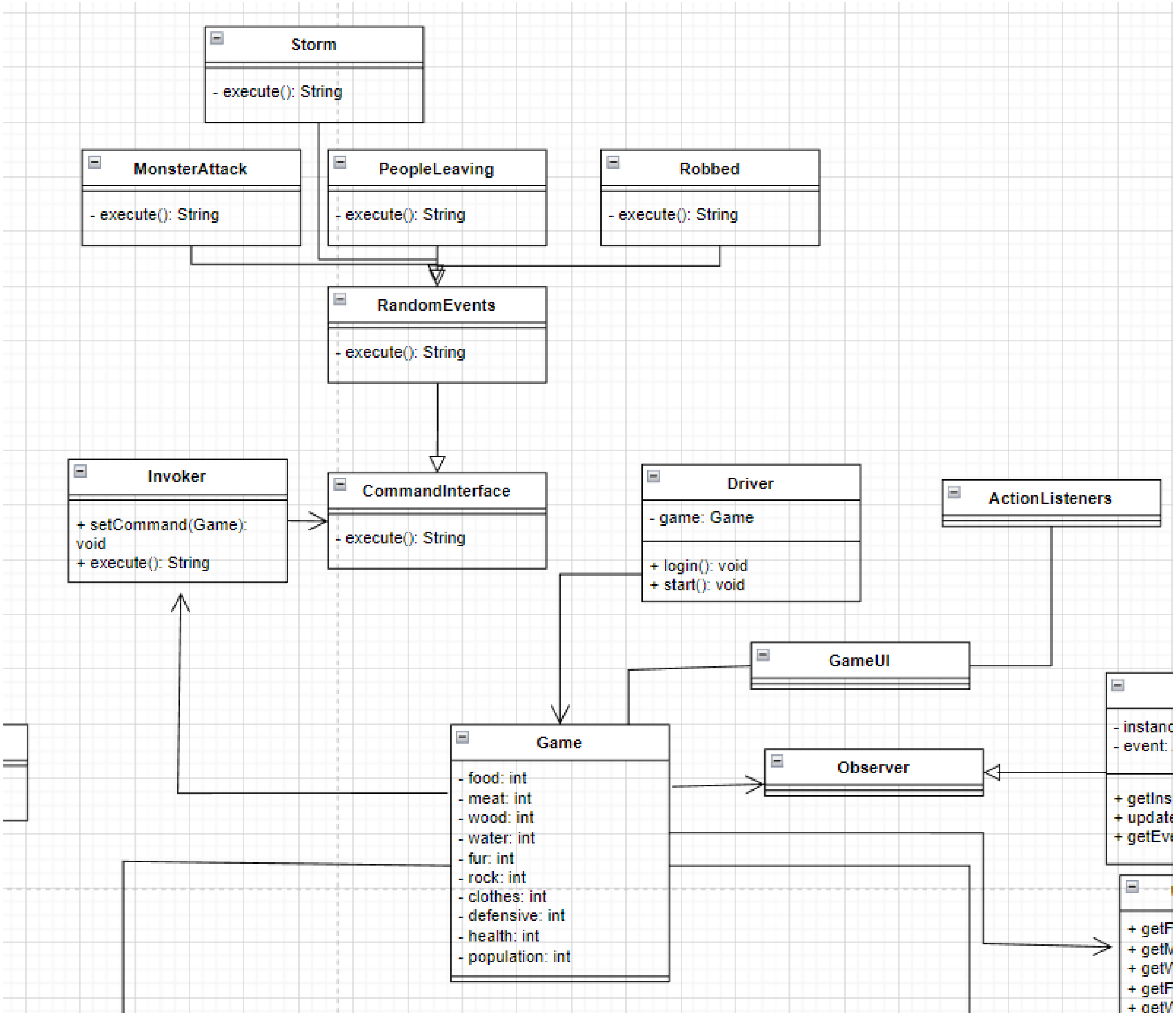
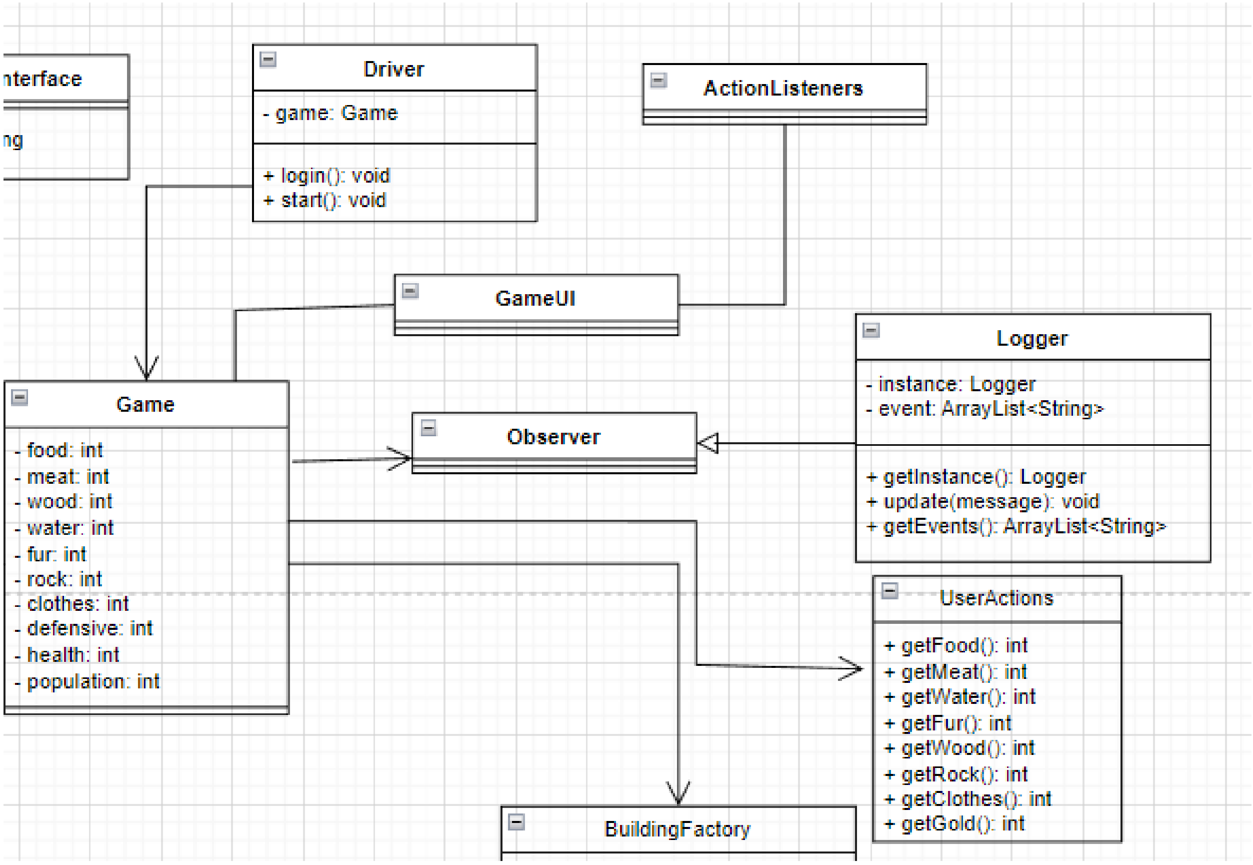
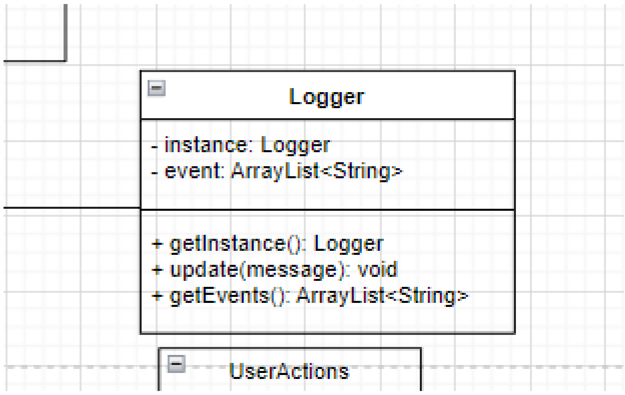
Project name: Defend Your Village
Team member: Anuragini Sinha & Lin Shi
Final State of System:
Within our proposed, *Defend Your Village* (a text-based survival game); we completed all of the features we offered in Project 5. These features include a login system, saving a game, user actions to gather resources, construct buildings, assign villagers to jobs, villagers collecting the resources, magic items that will boost production, a tradecart (shop), and random events. However, since the JFrame was a little challenging to mess around with, we did not spend as much time formatting the game exactly the way we wanted it. Additionally, we did not have enough time to balance the time. Overall, we completed all the features proposed in Project 5; however, we did not have enough time to make the game better and more fun.
Final Class Diagram:
. T Pattern:

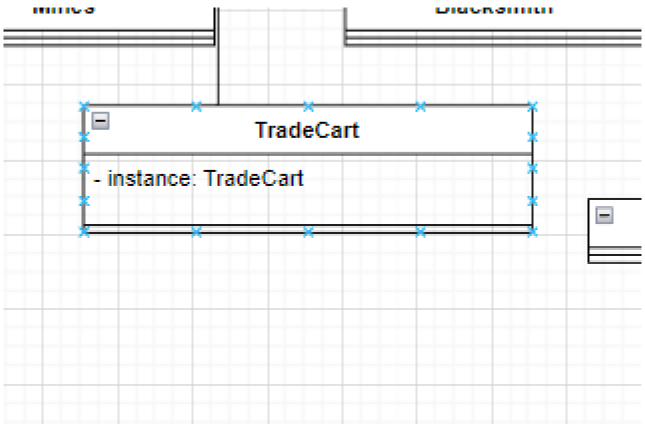


Observer Pattern:

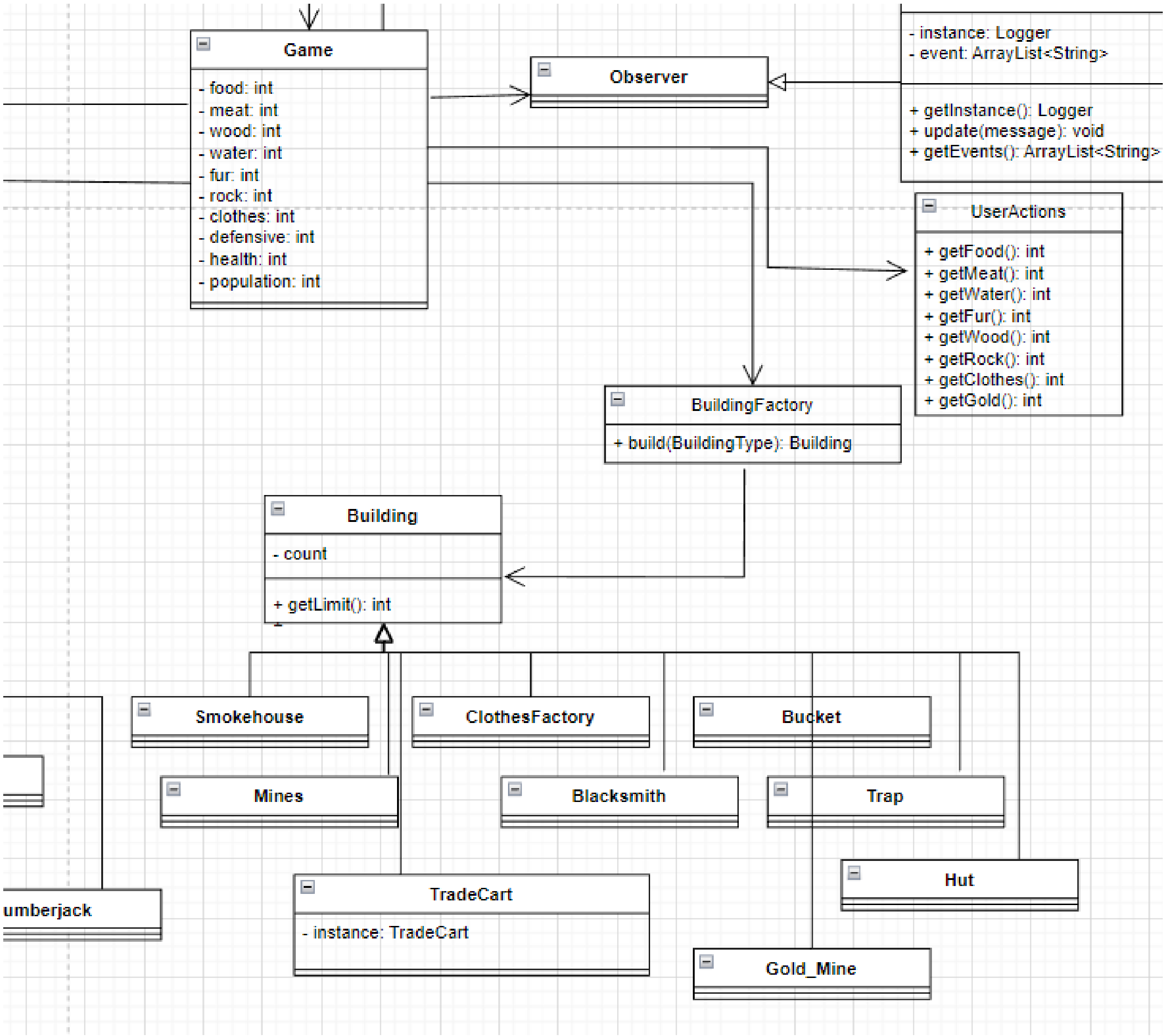


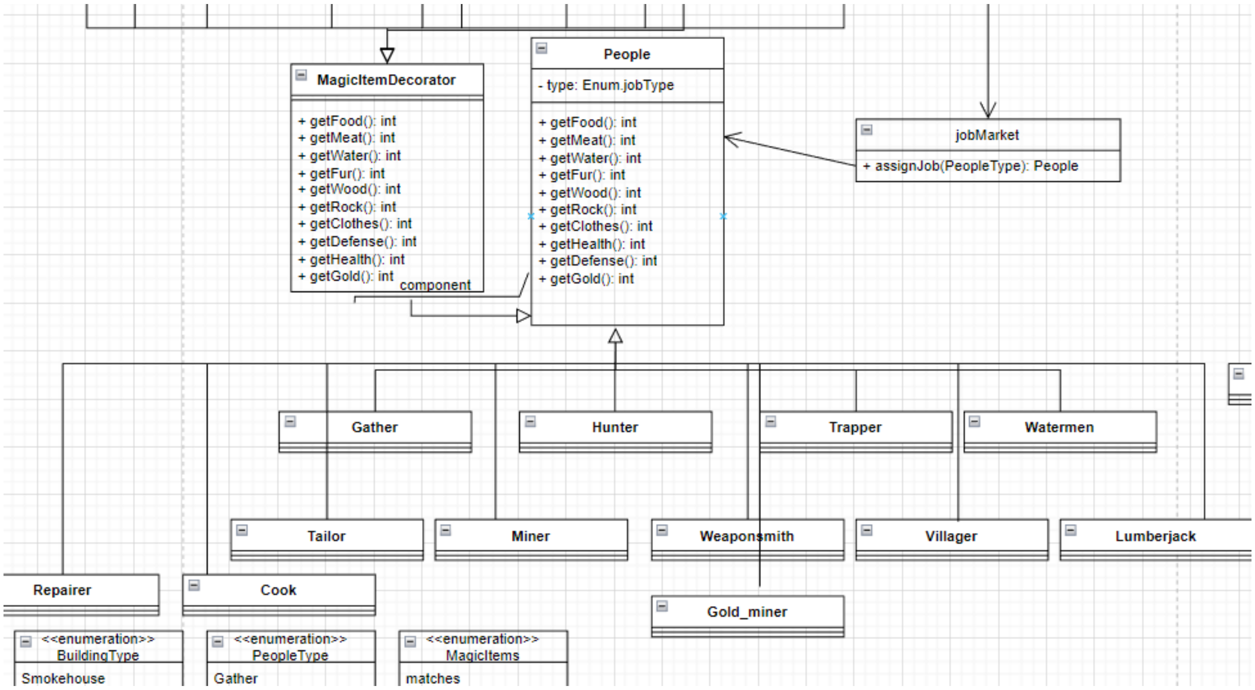
Singleton Pattern:



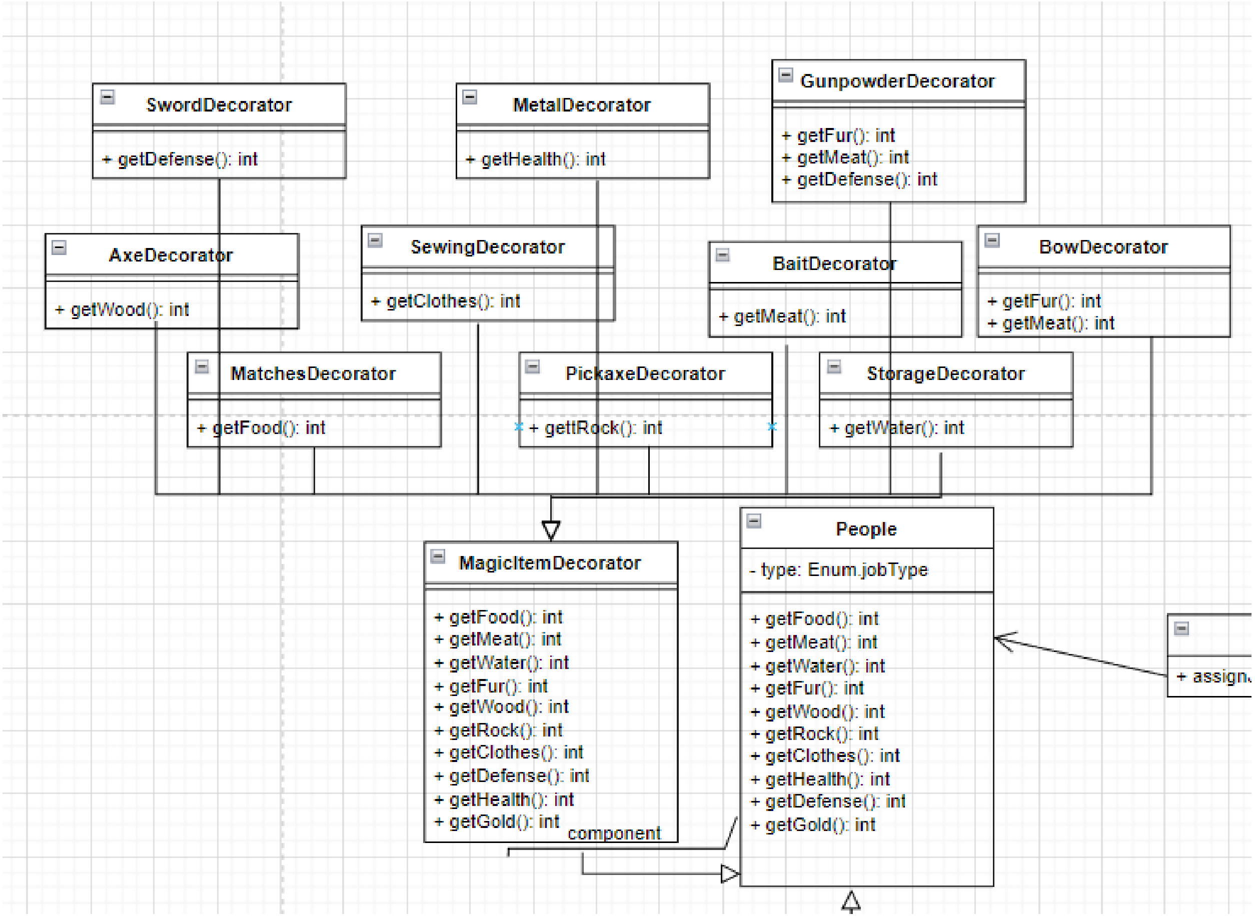


Factory Pattern:

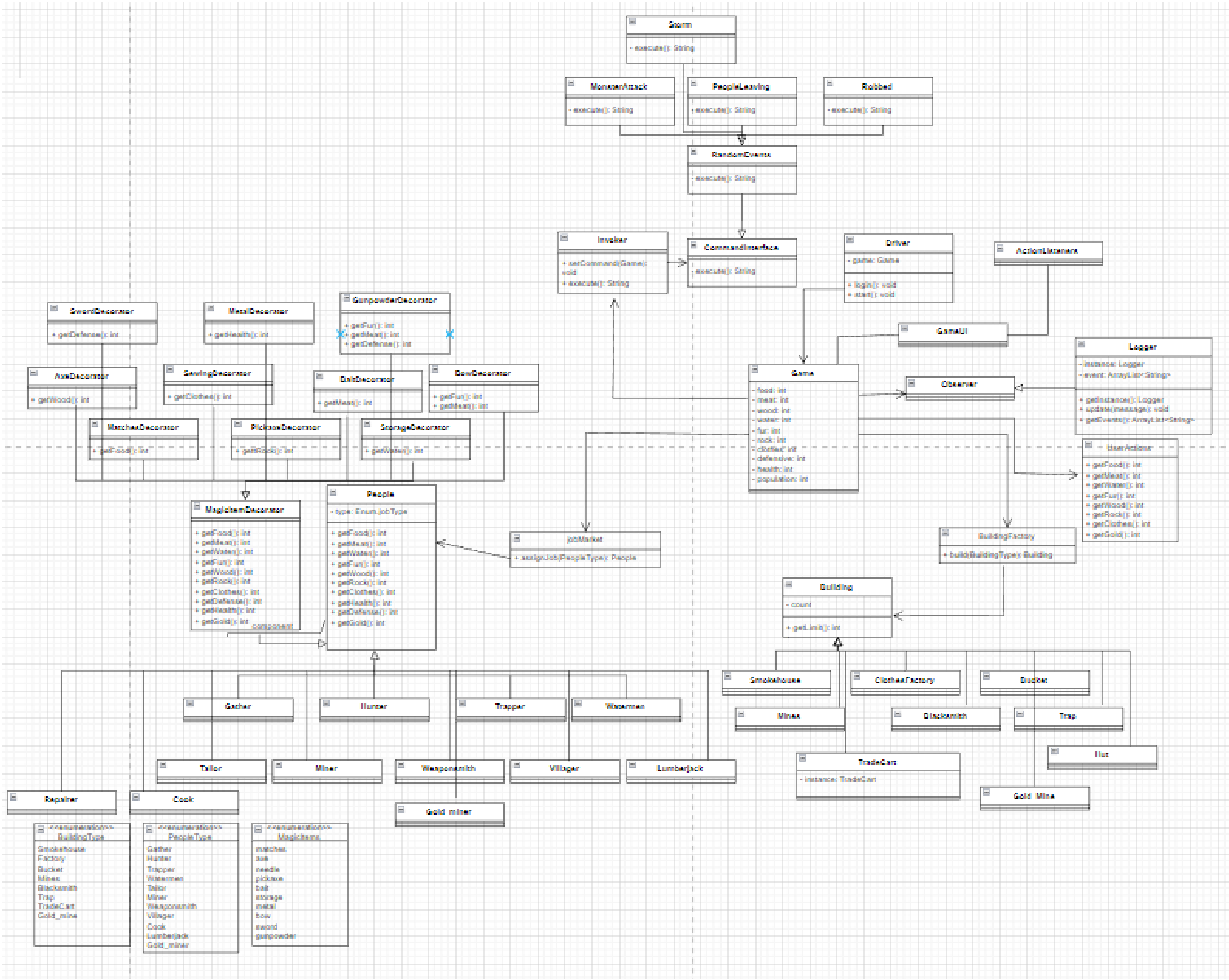




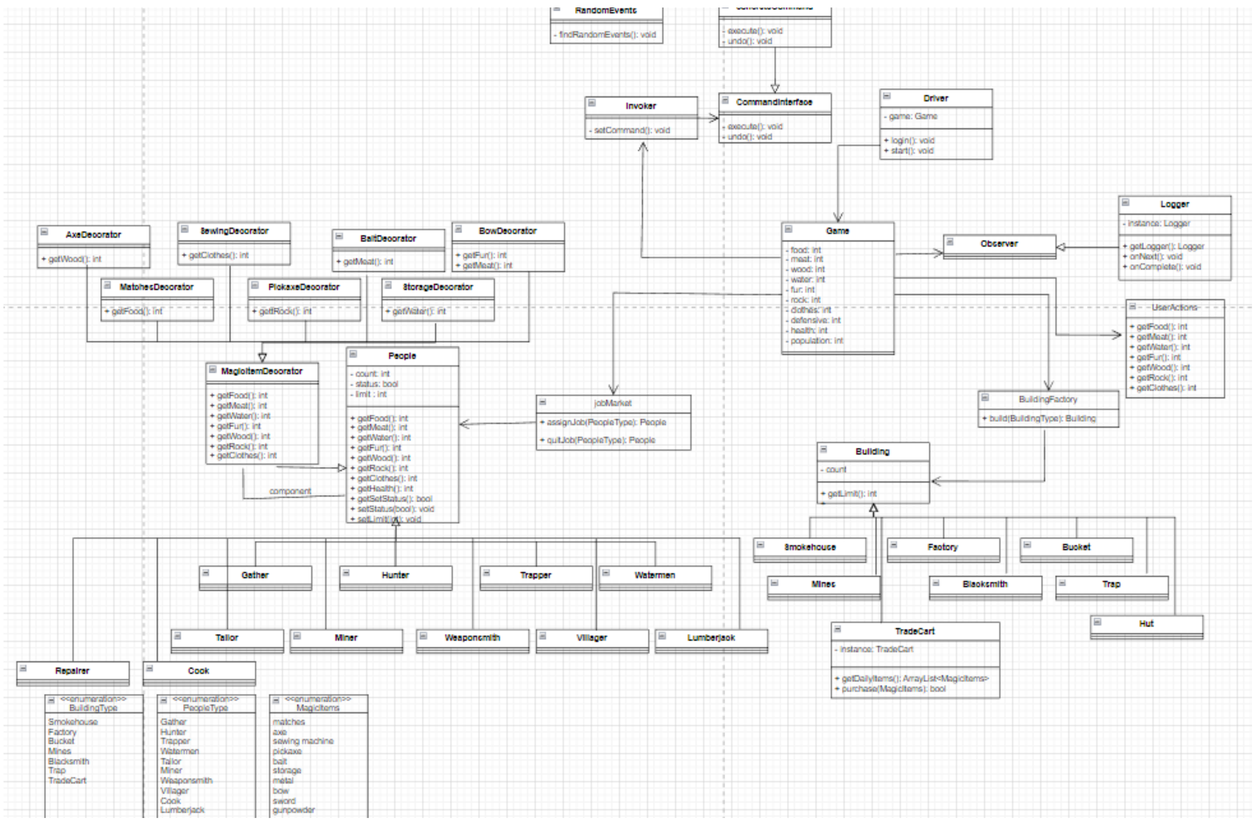
Decorator Pattern:



Current UML:

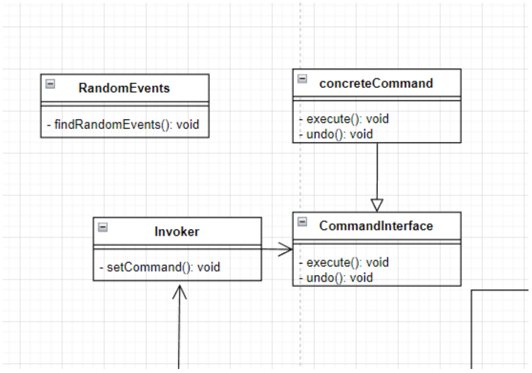


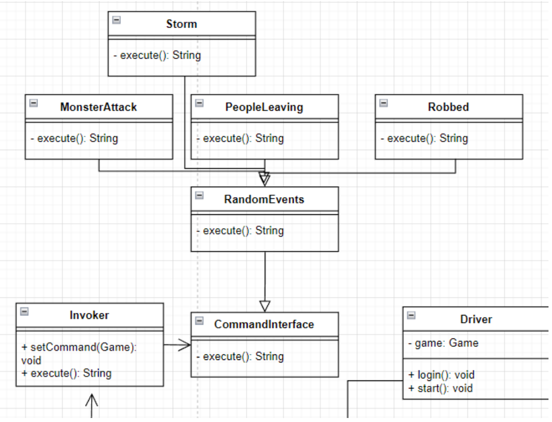
UML proposed in Project 5:



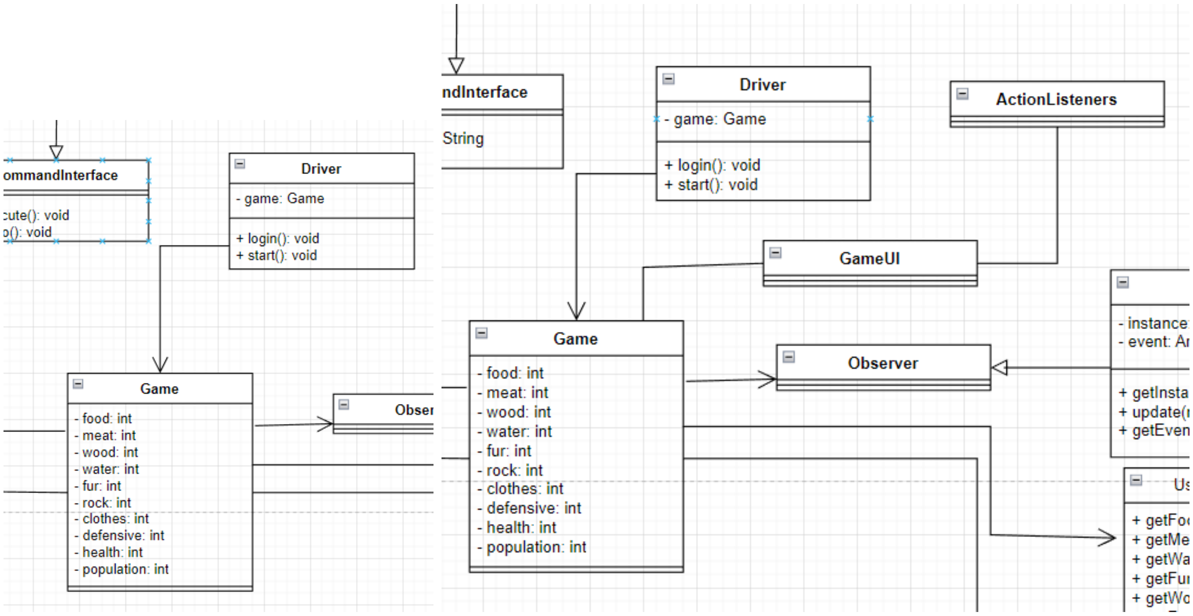
Discussion (What has changed since Project 5):

The structure of our project did not change much from Project 5 to Project 7. However, there have been a few changes while implementing the game. First, the Random concrete events has been listed (project five on the left and Project 7 on the right):





At the time of the design, we were unsure of the random events that may occur; therefore, we did not include all the instances. Second, we divided the Game class into GameUI and Game, and GameUI connects to a list of ActionListener.



When implementing the code, we quickly realized that the GameUI was going to be extremely long; therefore, dividing it into another class would be the best option. Additionally, each button requires an ActionListener, so we moved all of our ActionListener into its own class. Furthermore, in order for us to purchase from the tradecart (store), some kind of currency must be added. Therefore, we added the gold miner, gold mine, and variable gold throughout the program. Furthermore, we added a few extra magic items as we were working through the project. Lastly, there were a few naming changes and function parameter changes. Overall, we did not change the functionality, patterns, and features of the game from Project 5 to Project 7.

Third-Party code vs Original Code Statement:

When using the UI tools, a lot of documentation was used to look at the different functions that are available to us. All the UI tools used in this project are part of the swing package (https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html). Additionally, while getting started, we used an online tutorial for inspiration: (https://www.ryisnow.online/2021/04/java-for-beginner-text-adventure-game.html). Furthermore, the slides and the design patterns book by the gang of four were used for pattern implementations.

Statement on the OOAD process for the Semester Project:

One of the critical design processes that we analyzed and designed for the game involved saving an instance of the game. For instance, when users return to the game, their information is saved on a text file. Our user interaction diagrams noted this and was done in our game. After the save button is clicked, the game's current status is saved on a txt file called gamestats.txt. Another key design process that was taken into consideration of user actions in the game. Each day the user is allowed three actions, which was taken care of in the decorator class. The decorator classes were also responsible for randomly assigning values for each resource the user must collect for that day. These values were randomized between 1-5 or 5-10. The final key design process was the random events in the game. This was done using a command pattern that would alert the user about random events that have taken place in their village, such as an attack. As mentioned before, the appearance of these random events was documented using the command pattern. Furthermore, these random events were captured in our UML and code through different classes titled “Storm” or “Monster attack.”