## Section 2

I started with this section because I couldn't get a conceptual handle on how to design the game object model yet. For this section I decided to have the `Character` do most of its updating work on the server. I think this simplified the work of having the identical display for each client. When the client sent keyboard inputs and then after the updating work was done on the server I sent a list of the `Character`s to each client and then the clients would draw each character to the sketch. The `Character`s were basically just a class storing some properties.

I used JSON from GSON to pass data between the client and server. I tried to use the object streams and serialization but couldn't solve some bugs. So at first I think it was better to just stick to what I knew: just passing strings would be easier. This simplified the message passing quite a bit and didn't have much a performance hit.

## Section 3

I moved on to this section without having done section 1 because I needed a better understanding of how the abstract concept of the game object model will work with a concrete implementation. So at first I implemented all the requirements of this section and then abstracted out from there.

The design is basically the same as section 2 but instead of sending a list of characters I'm sending a class with some fields converted to JSON. I have two messages: `ServerClientInitializationMessage` is sent only once and initializes all the unchanging values the server and client need to know about, this got rid of message passing the same data all the time and got rid of duplicate code on the client and server, and `ServerClientMessage` is the primary message I'm sending between client and server, it contains only lists and arrays of primitive types. I'm only sending the primitive data that changes because when I sent lists of entire classes performance really suffered.

Once I completed the required functions of this section I realized I had a lot of code sections I could abstract out into parent classes and interfaces. For example I had a void `update` and `draw` method in the `Character` and `FloatingPlatform` classes. So I moved these methods up into a `Movable` component. There were also some redundant fields and helper methods I moved up as well. Abstracting this out will promote reuse and flexibility in the future.

I also had drawing and physics functions coupled directly to the server and client classes. So I moved drawing functions to the `Drawing` component and had the `Movable` component reference it, as the character and floating platform needed it. I used to have all the Processing drawing functions execute in one code block on the client but now there are different drawing snippets native to each game object, for promoting modularity and future scaling of more complex drawing. I moved the physics collision stuff to the `Physics` component and moved all the physics collisions to execute on the server thread because it alone has references to the positions of all the game objects.

There are several “objects” that are drawn to the sketch but I haven’t made them “game objects.” Things like the boundaries and the pit. I haven’t done this because they never need updating and making them game objects I think would introduce needless complexity, for this assignment at least.

## Section 1

As I had a hard time visualizing how a concrete game object model would look like in the java language I coded the other sections first and determined to move out duplicate code and coupling into a more abstract middle layer and see what kind of reasonable game object model came out of that. So I kind of reverse engineered the model from the components I made in section 3 and I ended up not changing much.

After I made the components in section 3. On looking at my components especially Movable I couldn’t see a way to make this into a more abstract model and so I ended up creating two marker interfaces: `GameObject` and `Component`. These implement `GameObject`: `Character, FloatingPlatform, Server`, and `Client`. And these implement `Component: Drawing, Immovable, Movable,` and `Physics`. These interfaces basically serve as a way to “tag” these classes to their designed function. If they weren’t tagged it may be difficult to see what layer of the game engine they reside on.

I didn’t find anything I could put in these marker interfaces to convert them to an regular interface or an abstract class with predefined methods and such because the components and game objects I used had mostly unique methods and fields. So it made more sense to use some inheritance and composition to get more custom behavior.

Essentially the game object model is something of an inversion of the generic component model. Instead of inheriting from GameObject the objects in my software object model are inheriting from components, have some reference to components via composition (`Physics` is a reference to a few game objects), and implement `GameObject` as an interface. Using some inheritance from components was the only way I could see to get method and field code reuse among game objects. I kind of wish I had some “functions as first class citizens” ability so I could do something like this: MovableComponent.update = function() { /* do stuff specific to this game object instance */ }. I think if I could assign functions to an interface method I could have envisioned implementing a more standard version of the generic component model.

I think in the future since this model is more inheritance based than composition based I may run into some problems deciding what properties to divide to get a sensical inheritance graph. Although I’m still unclear of any better way to get code reuse when my game object need some unique behavior. Maybe in future assignments my implementation would lend itself to exposing more opportunities for abstracting out (moving duplicate code) into a more composition based model.

Now that you have designed and implemented a number of engine components, write a 2–3 page paper **summarizing your design**. That is a minimum of 2 FULL single-spaced pages. Think creatively about what you have done. **Why did you make the design decisions you made? What did they enable for this assignment? What will they enable in the future? What didn't work out the way you had hoped, and why**? The most successful writeups will contain evidence that you have thought deeply about these decisions and implementations and what they can produce and have **gone beyond what is written in the assignment**. As an appendix to your paper, please include all relevant screenshots to support your discussion. The appendix does not count toward your 2–3 page requirement. Your points for the writeup will represent 1/2 of the points allocation for each of the above sections

https://github.com/cblupodev/csc481hw2
https://moodle-courses1617.wolfware.ncsu.edu/mod/resource/view.php?id=203233

implement a generic component model in code. did you use interfaces, abstract classes, how did you represenet game objects explicitly, what kinds of components did you crete, what methods where in the component classes, did you use inheritance, how did you use inheritance to create different types of components