#5.1

In a component-based architecture, the system's elements are viewed as loosely interconnected components that offer services to one another. Similarly, a service-oriented architecture operates on a comparable principle, albeit with its components implemented as services, frequently deployed on distinct computers and communicating over a network. While both architectures share similarities, a distinguishing feature of a service-oriented architecture is the heightened separation between its components.

#5.2

This application is designed to be straightforward and self-contained, eliminating the need for remote services or databases. Complex architectural designs such as client-server, multitier, component-based, and service-oriented approaches are likely excessive for its purpose. While these architectures could theoretically be implemented within the application itself, the basic nature of the task, such as playing against a simple computer opponent, does not warrant their use.

Given the simplicity and compactness of the application, a monolithic architecture would likely suffice. Additionally, a data-centric approach is suitable, especially considering the straightforward nature of tic-tac-toe. Building tables to store moves and optimal responses aligns well with this approach.

The user interface will be event-driven, primarily to respond to user actions. While it's possible to extend this event-driven paradigm to include the computer opponent's actions, it may not be necessary for such a basic application.

Although distributed components could be employed to parallelize move sequences, tic-tac-toe's inherent simplicity renders this feature unnecessary.
In summary, the most suitable architecture for this application would likely be a simple monolithic structure, leveraging rule-based and data-centric principles.

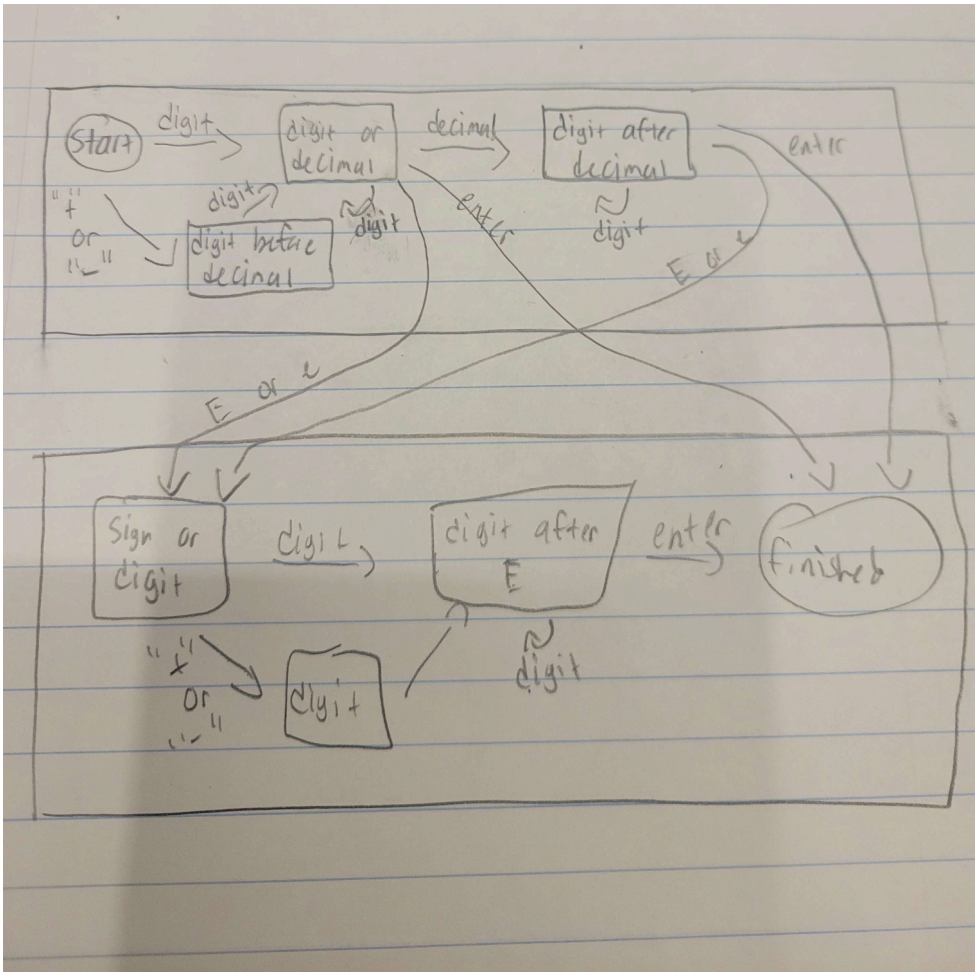
#5.4

While this scenario might appear more intricate compared to the previous one, it's manageable. The user interface remains largely unchanged. The only alterations involve: (1) The necessity for the program to share data with another instance across the Internet, and (2) The absence of a computer opponent. Eliminating the computer opponent obviates the need for distributed components. Employing web services facilitates communication between two instances over the Internet. This approach transforms the application into a monolithic, rule-based, and service-oriented system.

#5.6

The ClassyDraw application has the capability to store each drawing in an individual file, thereby minimizing the necessity for an extensive database. Operating system utilities enable users to effectively manage these files, facilitating tasks such as deleting outdated files and creating backup copies. To enhance user experience, the program could generate a temporary file during drawing editing sessions. In the event of a program crash or premature termination, the application could prompt the user about restoring the temporary file upon the next startup.

#5.8



#6.1

All these classes essentially represent drawable entities, sharing fundamental attributes needed to draw something. That includes common features like foreground color and background color. Additionally, each class can determine its position by storing details like an upper-left corner, a width, and a height.

While these classes share certain properties, they may require unique data to draw their specific shape. For instance, the Text class requires font information, whereas the Star class needs information about the number of points for the star. Certain properties, such as fill color, are applicable to some classes and not others. Classes like Rectangle, Ellipse, and Star can be filled, hence requiring a fill color.

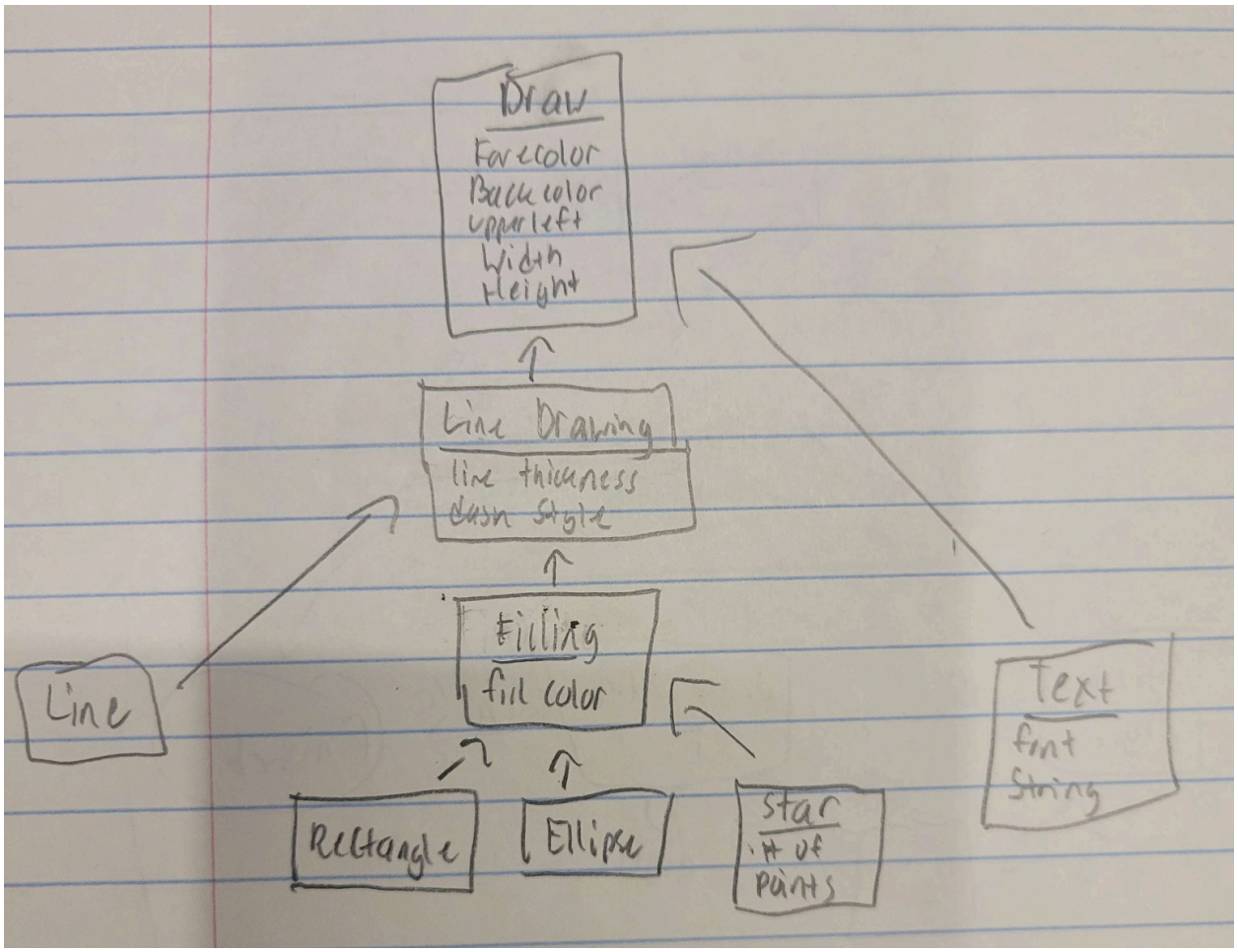
Moreover, the classes dealing with lines, namely Line, Rectangle, Ellipse, and Star, also need line-specific properties such as thickness and dash style.

Properties of classes:

- ForeColor, BackColor, UpperLeft, Width, Height → they use all/everything

- Font and String → also need Text
- Numpoints → needs Star
- Fill Color → Rectangle, Ellipse, Star
- LineThickness and Dash Style → Rectangle, Ellipse, Star, Line

#6.2



.