

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 9/21/2025 | Athena Trunkhill | Wrote Requirements, Design Constraints, and Domain Model |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

*To facilitate The Gaming Room's development of a web-based version of "Draw It or Lose It" game from mobile only application to a multi-platform, web-based service, they have requested Creative Technology Solutions design and develop a foundational Java-based application. The application design focuses on maintainability and ensures consistent experience for all users, regardless of the platform they use to connect. The solution achieves this through software design patterns, including the Singleton pattern for centralized game management, the Iterator pattern to ensure unique names, and Inheritance to promote a reusable and maintainable code structure.*

## Requirements

*The solution directly follows the client's key technical requirements. It utilizes design patterns and inheritance to ensure only one instance the central Game Service can exist, and to ensure that the code is clean, reusable, and easily extendable.*

## [Design Constraints](#_2et92p0)

Developing "Draw It or Lose It" as a web-based, distributed application introduces several design constraints. These constraints and their implications on development are:

**Multi-Platform Support**: The primary goal is to support various client platforms from a single backend.

**Implication**: This requires a clear separation between the server-side application logic and the client-side user interface. The backend must use a versatile Application Programming Interface that any front-end client can use to communicate with the game server to ensure consistent experiences across all devices.

**Distributed Systems & Networks**: Players will be connecting from different locations through the internet with varying levels of connection stability.

**Implication**: The application must be resilient to network issues and temporary disconnects. The system architecture needs to manage game state centrally on the server to ensure all players have a synchronized game state so that way they have a fair experience.

**Scalability**: The game system may need to support a rapidly growing number of concurrent players.

**Implication**: The server-side infrastructure must be designed to scale based on volume of users. This influences choices in relation to the hosting environment and database performance to handle increasing loads if they change.

**Uniqueness of Game and Team Names**: The requirement for unique names is critical for data integrity.

**Implication**: The software must contain logic to validate new names against all existing names before they are created and have a database able to store the information of used names. There also needs to be clear parameters for what is a valid name for Games or Teams.

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

The Domain Model that is represented by the UML class diagram, illustrates the core components of the "Draw It or Lose It" application and the relationships between them. It provides a blueprint for the software's structure. The model is built upon a foundational base class called Entity. This class contains the two attributes shared by all major components which are a unique id and a name. These attributes help ensure that each instance of an entity is unique.’

The classes that inherit from the base Entity class are Game, Team, and Player. They are a part of the inheritance principle as they all are built upon the base entity class. By defining the common features in Entity once, we avoid duplicating code across the other three classes.

The diagram shows relationships between these classes with a Game being composed of one or more Team ‘s and a Team composed of one or more Player ‘s. This follows the principle of encapsulation as the objects have their internal details store separately and only accessible through methods like addTeam and addPlayer to ensure they are only modified in a controlled manner.

The last principle represented is regarding the GameService class which centralizes everything to a single instance. It is a Singleton class so there is only one instance at any time which ensures that rules, such as the uniqueness of game names, are enforced.

Additionally, the UML diagram utilizes two utility class of ProgramDriver and SingletonTester. The ProgramDriver class serves as the entry point for the application, containing the main() method that starts the program. The SingletonTester verifies that the GameService singleton pattern is working correctly.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Characteristics**: macOS Server is an add-on to the desktop OS. It's built on UNIX foundation but is primarily designed for small-scale deployments.  **Advantages**: It offers an easy to use interface and is well-integrated with other Apple products.  **Weaknesses**: It is not a common choice for hosting web applications because it only runs on Apple hardware, as well as higher costs. | **Characteristics**: Linux is the primary OS for web servers. Distributions like Ubuntu Server and CentOS are specifically designed for hosting applications.  **Advantages**: It is open-source and generally free of licensing costs. It is highly stable, secure, and has an ecosystem of support and tools.  **Weaknesses**: It can have a steeper learning curve for administrators who are not familiar with the command-line interface. | **Characteristics**: Windows Server is a widely used commercial server operating system from Microsoft.  **Advantages**: It has strong commercial support, is well integrated with other Microsoft products, and offers a familiar graphical user interface.  **Weaknesses**: There are significant licensing costs for the OS and for client access licenses. | **Characteristics**: Mobile devices are not suitable for use as servers. They lack the processing power, stable network connections, and OS features required.  **Advantages**: None  **Weaknesses**: Mobile platforms are designed as clients, not servers. Attempting to use one for hosting would be unstable, insecure, and unable to handle user traffic. |
| **Client Side** | **Considerations**: Supporting Mac clients is straightforward. Development must focus on ensuring the web application's HTML, CSS, and JavaScript work correctly on the Safari browser, as well as on other No extra costs are associated with supporting Mac clients beyond standard web development and testing time. | **Considerations**: Supporting Linux clients is also straightforward. The primary consideration is ensuring compatibility with popular browsers on Linux, mainly Firefox and Chrome. Like with Mac, there are no special costs, as the application is web-based. | **Considerations**: Supporting Windows clients is a standard requirement. Development must ensure the application works on Microsoft Edge, as well as Chrome and Firefox. There are no extra costs associated with supporting Windows as a client | **Considerations**: This is a key requirement. The application must have a "responsive" design, meaning the user interface automatically adapts to the smaller screen sizes of both iOS and Android devices. This requires significant development and testing effort to ensure a good user experience on touch screens. |
| **Development Tools** | The backend can be developed in Java using IDEs, like Eclipse, which are cross-platform. Front-end development will use HTML, CSS, and JavaScript. Xcode is the primary tool for native iOS development but it is not needed for a web-based client. | The development tools are the same as on Mac. Linux is an excellent development environment. There are typically no costs for the primary development tools. | The tools are same utilizing Java and web technologies. Windows is a very common development environment with excellent support for all the necessary tools. | The server-side code remains Java. For the client-side, developers will use web development tools and responsive design frameworks to create the mobile web interface. Browser-based developer tools on desktop and mobile simulators are needed for testing the user experience. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: **Linux**- I recommend using Linux as the operating system for the server-side hosting. The reason being the cost factor since it is open source and is an industry standard for web hosting. The choice offers the best price for performance of each option.
2. **Operating Systems Architectures**: I recommend that the application uses a client-server architecture. The Linux server will host the backend Java application and a web. All game logic and data will be processed on this central server. Players' devices will run a web browser that acts as a client and connect to the server over the internet to play the game.
3. **Storage Management**: I recommend using a relational database system MySQL for storage management. It is open-source, powerful, and works well with Linux and Java. This database will store user account information, game data, and other essential application data.
4. **Memory Management:** I recommend memory management be handled at two levels. First, the Linux operating system will manage the system's overall memory using virtual memory techniques. Second, the Java Virtual Machine (JVM) will manage the memory for the application itself. JVM’s automatic garbage collection will handle allocating and freeing up memory to prevent memory leak issues.
5. **Distributed Systems and Networks**: I recommend that the game functions as a distributed system, where clients on different devices communicate with the central server. This will be accomplished using a REST API over the secure HTTPS protocol. The clients will send requests to the server, and the server will send responses back. This architecture allows any client platform to communicate with the server as long as it has an internet connection.
6. **Security**: To protect user information, several security measures are essential. All communication between clients and the server must be encrypted using HTTPS (SSL/TLS). User passwords must be securely hashed before being stored in the database. Finally the application code must include input validation to protect against vulnerabilities like SQL injection.