Draw It or Lose It
CS 230 Project Software Design Template
Version 3.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	07/15/25	Hassan Lindsay	Initial Draft
2.0	08/03/25	Hassan Lindsay	Updated Evaluation and Recommendation Section
3.0	08/13/25	Hassan Lindsay	Updated Recommendation Section

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.

<u>Executive Summary</u>

The Gaming Room, a client of Creative Technology Solutions (CTS), has requested the development of a web-based version of their existing Android game, *Draw It or Lose It*. The game requires multiple teams, each with multiple players, and it must ensure name uniqueness for both teams and games. Additionally, only one instance of the game should exist in memory at any time. This software design document outlines the proposed object-oriented solution using the singleton and iterator patterns to meet the client's requirements. It also discusses how this design can scale across platforms while ensuring performance, security, and maintainability.

<u>Requirements</u>

- *A game must be able to support one or more teams.*
- *Each team can have multiple players.*
- *Game and team names must be unique.*
- *Only one instance of the GameService class should exist at any given time (singleton).*
- *Unique identifiers must be assigned to each game, team, and player.*
- *The system should allow name validation before creating a game or team.*

<u>Design Constraints</u>

The application must be developed for a web-based distributed environment, meaning it will be hosted on a server and accessed by clients on various platforms (e.g., desktops, mobile devices). Design constraints include:

- **Singleton Pattern**: Only one instance of GameService can exist. This avoids data duplication and inconsistency in game state management.
- **Unique Identification**: Games, teams, and players must have unique IDs and names.
- **Cross-Platform Compatibility**: The application must function across different platforms, requiring a clean separation of logic and UI.
- **Scalability**: The design must handle multiple games, teams, and players without performance degradation.

These constraints require careful memory management, robust iteration mechanisms, and strong validation logic.

<u>System Architecture View</u>

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.

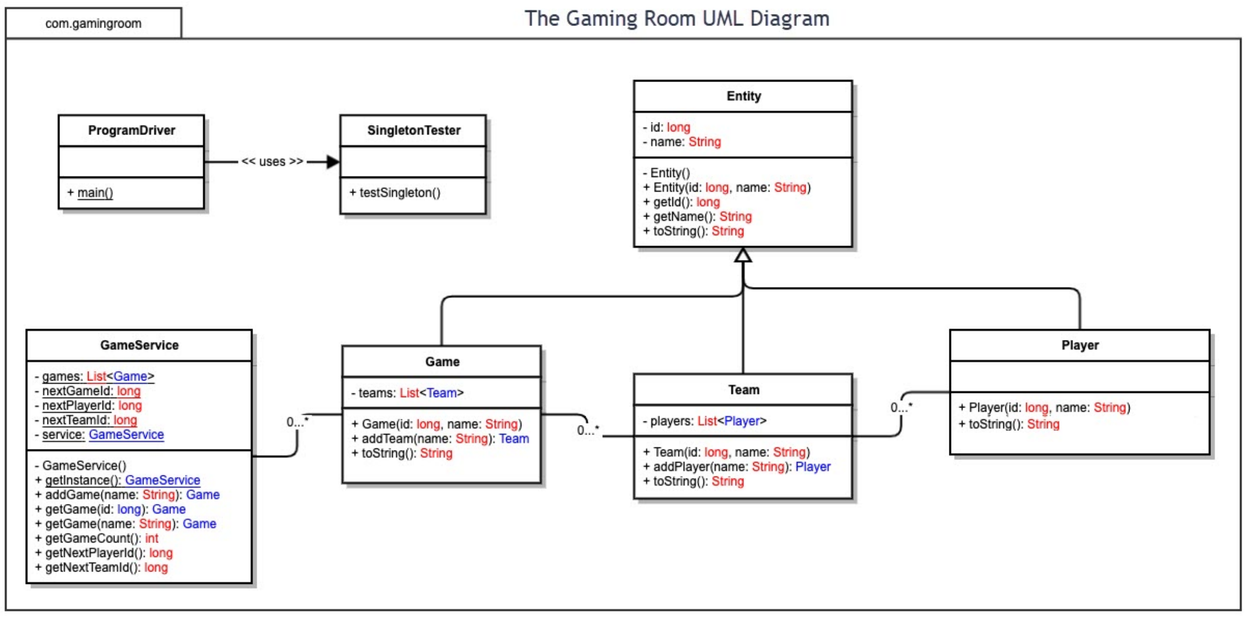
<u>Domain Model</u>

The UML diagram represents a hierarchical object-oriented structure for managing the game state:

- **Entity (Base Class)**: Holds common attributes (id, name) for reuse.
- **Player, Team, Game**: Inherit from Entity. This demonstrates **inheritance** and **encapsulation**.

- **GameService (Singleton)**: Manages a list of games and ensures only one instance exists. Demonstrates **singleton pattern**.
- **Game**: Contains multiple teams; ensures **composition**.
- **Team**: Contains multiple players; ensures **composition**.
- **Iterator Pattern**: Used in methods like addGame, addTeam, and addPlayer to ensure uniqueness.

These principles support maintainable and reusable code and align with best practices for object-oriented design.



Evaluation

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	Mac servers are Unix-based and share many features with Linux, making them reliable and secure for hosting. However, they are not commonly used in enterprise-scale server environments. Mac servers are best suited for development or internal hosting. Licensing is straightforward, but hardware is costly and limited.	Linux is a powerful, flexible, and scalable OS ideal for server-side deployment. It supports high-performance applications and is used widely in cloud environments. Licensing is open source, making it cost-effective. It provides robust community and professional support.	Windows Server is a commercial server OS used in many enterprise settings. It provides excellent support and compatibility with .NET applications. Licensing can be expensive, and hardware costs may vary depending on configuration.	Mobile platforms are not designed for traditional server deployment. They are unsuitable for hosting backend services due to hardware and OS limitations. Cloud-based mobile backend solutions may supplement mobile apps.
Client Side	macOS is widely used for client-side applications, especially in creative and professional environments. It supports rich GUI design and integrates well with native Apple frameworks.	While not traditionally popular for desktop clients, Linux client-side applications are gaining traction in open-source communities. GUI development is more complex and less standardized across distros.	Windows is a dominant client platform, compatible with a vast array of commercial software. It supports both native and web-based apps. UI development is highly customizable.	Mobile apps are essential for modern clients. They must be responsive, secure, and tailored to platform-specific UX expectations. Android has a larger market share; iOS has higher profitability per user.

Development Tools	Xcode is the primary IDE for macOS development. It requires a macOS system and is available for free through the Mac App Store. Development can be efficient for small teams but may increase workload due to limited hardware options. There is Java, Eclipse, IntelliJ.	Popular tools include Eclipse and Visual Studio Code. Linux offers great scripting support and flexibility. Teams may require more experience with shell environments and package managers. There is also Java and NetBeans.	Visual Studio and IntelliJ are commonly used. Development is well-documented, but licensing costs (Windows OS, Visual Studio Pro) must be considered. Maintenance requires regular patching and update management. Eclipse is also commonly used.	Android Studio, Flutter, and Xcode are commonly used. Cross-platform tools (e.g., React Native, Flutter) help reduce workload. Teams must handle platform-specific testing and comply with each app store’s guidelines.

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 is recommended due to its stability, scalability, open-source flexibility, and strong support for Java-based applications.

2. **Operating Systems Architectures**: Linux supports monolithic and modular kernel architectures, allowing optimized resource management and performance, ideal for hosting a scalable game like *Draw It or Lose It*.

3. **Storage Management**: Use a file-based or cloud-based storage system (e.g., Amazon S3, Firebase, or local file system) to store game state and user data. Linux has strong support for file system-level permissions and backups.

4. **Memory Management**: Linux offers advanced memory management techniques including paging, segmentation, and cache handling. These techniques ensure efficient memory use and reduce the risk of crashes.

5. **Distributed Systems and Networks**: The system will use RESTful APIs or WebSockets for client-server communication, enabling real-time updates across platforms. Redundancy and load balancing will mitigate potential outages.

6. **Security**: Implement HTTPS for secure communication, user authentication, and data encryption. Linux offers robust built-in security, and additional tools (like firewalls and SELinux) can enhance system safety.

For Server-Side: Linux is the strongest choice for server deployment due to its flexibility, scalability, cost-effectiveness, and extensive support for Java-based frameworks like Dropwizard. Licensing is open-source and hardware can be provisioned across on-premises or cloud environments. While Mac offers Unix compatibility, it lacks scalability and broad support. Windows Server is a viable alternative but comes with higher licensing costs.

For Client-Side: To reach the widest user base, supporting Android, iOS, macOS, and Windows is ideal. Android and iOS require tailored mobile apps, but cross-platform tools like Flutter or React Native can minimize redundant development. Windows and macOS clients should use native or hybrid web-based applications. Using RESTful APIs ensures that server-side logic remains centralized and reusable.

Team and Tooling Considerations: Development teams should be equipped with platform-specific IDEs (e.g., Android Studio, Xcode, Visual Studio). While this diversity may increase onboarding and QA workloads, it also allows flexibility in task assignments. Open-source tools like IntelliJ and VS Code help control cost. Cloud-based testing and CI/CD pipelines can reduce time spent on environment-specific debugging.

This strategy balances scalability, security, and usability while ensuring a robust client-server architecture that can adapt to future platforms and user needs.