Draw It or Lose It

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	11/15/25	Kenya Craw	Initial submission of project document with UML, design patterns, and code evaluation.
1.0	11/29/25	Kenya Craw	Edits and/or changes to evaluation table, additions in references for clarity, and edits, and recommendations for further clarity.
1.0	11/30/25	Kenya Craw	References alphabetized for clear communication and clarity. Spelling and grammar revisions.
1.0	12/14/2025	Kenya Craw	Additions and edits applied to recommendations section for further clarity and better explanations.
1.0	12/14/2025	Kenya Craw	Table of contents updated for easier section viewing and/or finding.

Executive Summary

This document describes the design of Draw It or Lose It, a multi-user web-based game application. The software allows players to create games, join teams, and play collaboratively. To ensure data consistency and proper management of active games, the GameService class implements the Singleton Pattern, providing a single global instance to manage all games. The application also uses the Iterator Pattern to safely traverse game and team collections. These design choices enhance maintainability, prevent duplicate instances, and provide clear communication for future developers working on the project.

Requirements

The system must allow multiple games to be created and managed simultaneously.
Each game must have a unique name and ID.
Teams and players can be added to games with unique IDs.
The application should prevent duplicate game or team names.
The system should support multiple platforms including Windows, Mac, Linux, and mobile devices.
Security and data integrity must be maintained across all game operations.

Design Constraints

The application must be implemented in Java and follow object-oriented programming principles.
Only a single instance of GameService can exist at runtime (Singleton Pattern).
Collections of games and teams must be traversed using iterators to avoid exposing internal structures.
The solution must be compatible with multiple operating platforms and web-based deployment.
Limited memory and storage constraints must be considered when managing large numbers of games, teams, or players.

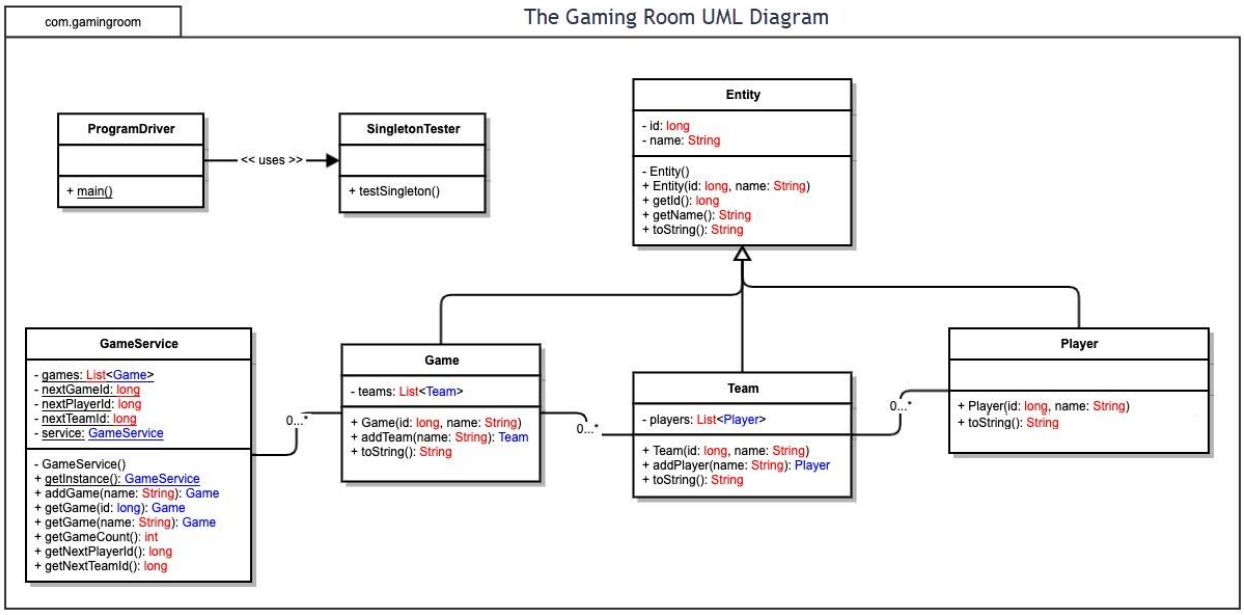
System Architecture View

The system architecture consists of the following components:
Client Tier: Users interact with the game through a web interface or mobile application.
Application Tier: The Java-based game engine (GameService, Game, Team, Player) handles business logic.
Data Tier: In-memory storage is used for games, teams, and players, with potential future expansion to a database.
Logical communication follows a client-server model where all requests pass through GameService, ensuring a single point of control.

Domain Model

The UML diagram represents the following relationships:
GameService implements the Singleton Pattern.
GameService manages multiple Game objects (0..*).
Each Game contains multiple Team objects (0..*).
Each Team contains multiple Player objects (0..*).
Game, Team, and Player inherit from a conceptual Entity class (common id and name).
Object-Oriented Principles Applied:

Encapsulation: Internal lists of games, teams, and players are private.
Inheritance: Game, Team, and Player inherit shared characteristics of Entity.
Design Patterns: Singleton ensures a single GameService instance, and Iterator provides safe traversal.



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices

Server Side	provides a stable JVM environment, making development and local testing reliable. Tools like Apache, NGINX, Tomcat, and lightweight Java servers run effectively for prototyping distributed systems. Although not widely used for production deployment due to licensing and lower enterprise adoption, macOS remains excellent for development environments already using Apple hardware. *(Apple, 2023; Baeldung, 2022; Oracle, 2023)*	Linux remains the industry leader for Java-based production servers due to its scalability, performance, and strong compatibility with tools like Docker, Kubernetes, Tomcat, NGINX, and cloud platforms such as AWS and Azure. Its open-source nature keeps operating costs low, making Linux the most cost-effective and scalable option. *(Red Hat, 2023; NGINX, 2024; AWS Docs, 2023)*	Windows Server supports Java but is more commonly used in enterprises already invested in Microsoft ecosystems. It offers straightforward GUI-based configuration and strong security options like Active Directory integration. However, licensing costs may be higher, and performance tuning requires more overhead than Linux. *(Microsoft Docs, 2023; Oracle Java Platform Guide, 2023)*	Mobile devices cannot host server-side Java applications because they lack support for full JVM back-end processing. Instead, they rely on remote servers hosted on Linux, Windows, or cloud-based services. Mobile applications communicate through REST APIs or WebSockets. *(Google Firebase Docs, 2024; Android Developers, 2024)*

Client Side	Supports all major browsers (Safari, Chrome, Firefox), making it ideal for testing responsive and cross-platform user interfaces. Java GUI applications also run smoothly for testing desktop clients. Mac is widely used by developers and creatives, making testing here valuable for real-world coverage. *(MDN Web Docs, 2022; Apple Developer Docs, 2023)*	Linux supports strong browser-based testing and Java for desktop applications. Developers benefit from powerful debugging tools, customizable environments, and lower-level system access, making Linux ideal for performance testing under varied system conditions. *(Ubuntu Docs, 2023; Red Hat, 2023)*	Windows, the most widely used desktop OS globally, is essential for client-side testing. It supports all major browsers and provides a realistic testing environment for large user populations. Java applications run well through JVM implementations on Windows. *(StatCounter, 2024; Microsoft Docs, 2023)*	Mobile testing requires assessing performance under tighter constraints (battery, memory, CPU). Development uses frameworks like Xcode or Android Studio, and testing must cover adaptive UI behavior, touch interactions, and device-specific constraints. *(Android Developers, 2024; Apple Developer Docs, 2023)*
Development Tools	MacOS supports IntelliJ IDEA, Eclipse, VS Code, Maven, Gradle, Git, and Docker. Its Unix-based terminal provides strong scripting and automation capabilities. macOS is popular among developers due to its stability and software ecosystem. *(JetBrains, 2024; Docker Docs, 2023)*	Linux excels with development automation thanks to bash scripting, package managers, and containerization tools. It is ideal for DevOps pipelines and scalable distributed development. *(JetBrains, 2024; Docker Docs, 2023; CNCF Report, 2024)*	Windows supports Java development through IntelliJ, Eclipse, VS Code, and Windows Subsystem for Linux (WSL), enabling hybrid workflows. Its tool variety makes it highly flexible for enterprise environments. *(Microsoft Docs, 2023; JetBrains, 2024)*	Mobile development tools include Xcode, Android Studio, and cross-platform frameworks like Flutter and React Native. These tools provide advanced profiling, simulators, and debugging environments. Secure API communication is essential for mobile-to-server interactions. *(Flutter Docs, 2024; Android Developers, 2024)*

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**: A Java-based platform is recommended due to JVM portability and dedicated support across macOS, Linux, Windows, and cloud environments (Oracle, 2023; Baeldung, 2022). The Java Virtual Machine allows the Draw It or Lose It application to maintain a single server-side codebase while supporting multiple client platforms. Linux is recommended for production deployment because of its high reliability, scalability, and cost efficiency in enterprise server environments (Red Hat, 2023; AWS Documentation, 2023). Linux is widely used for hosting Java-based applications and integrates well with cloud infrastructure and container-based deployments.

2. **Operating Systems Architectures**: Using 64-bit operating systems is advised because they allow larger memory allocation and improved performance for Java applications. The JVM abstracts platform-specific differences, simplifying cross-platform development and long-term maintainability (Oracle JVM Guide, 2023; JetBrains, 2024). Linux uses a layered operating system architecture consisting of the kernel, system libraries, and user space. The kernel manages hardware resources such as CPU scheduling, memory allocation, and device access, while user-space applications operate independently for improved security and stability. This architecture supports multitasking and multi-user environments, allowing multiple game sessions and users to run concurrently without performance degradation.

3. **Storage Management**: In-memory storage is sufficient for the current small-scale implementation due to its speed and simplicity, enabling fast access to game, team, and player data during gameplay (Oracle Java Docs, 2023). As the system scales, persistent storage will be required. Relational databases such as MySQL, PostgreSQL, or SQLite provide stronger data persistence, indexing, and transactional reliability, making them suitable for storing user accounts, game history, and scores (MySQL Developer Reference, 2023; PostgreSQL Documentation, 2023; SQLite Docs, 2024).

4. **Memory Management**: Java's automatic garbage collection efficiently manages memory for Game, Team, and Player objects, reducing the likelihood of memory leaks in continuously running server applications (Oracle Java SE Documentation, 2023; Baeldung, 2023). This allows developers to focus on application logic rather than manual memory allocation. At the operating system level, Linux supports virtual memory and paging, enabling efficient use of physical memory and maintaining system stability during high user activity and peak load conditions.

5. **Distributed Systems and Networks**: A centralized GameService instance maintains a consistent game state across all users and simplifies synchronization logic in a multi-user environment (Google Cloud Architecture Guide, 2024). Communication between clients and the server should occur through REST APIs or WebSockets to support responsive, real-time gameplay across different platforms (MDN Web Docs, 2023; Android Developers, 2024). This distributed architecture allows the system to scale horizontally and handle network latency, connectivity interruptions, and increasing user demand through load balancing and server redundancy.

6. **Security**: Input validation at every endpoint mitigates injection attacks and malformed requests, protecting the application from common vulnerabilities (OWASP, 2023). HTTPS should be required to ensure encrypted communication between clients and servers, safeguarding sensitive user data during transmission (Mozilla Security Docs, 2023). Role-based access control (RBAC) provides structured authorization and supports scalable, multi-tier expansion as the

system grows, ensuring that users and administrators only have access to appropriate resources (NIST Access Control Guide, 2022).

Role-based access control (RBAC) provides structured authorization and supports scalable multi-tier expansion as the system grows (NIST Access Control Guide, 2022).

References

Apple. (2023). *macOS developer documentation*. https://developer.apple.com

Android Developers. (2024). *Android developer guides*. https://developer.android.com/docs

AWS Documentation. (2023). *Modern Java application hosting on AWS*. https://docs.aws.amazon.com

Baeldung. (2022). *Java and JVM basics*. https://www.baeldung.com

Baeldung. (2023). *Java memory management and garbage collection*. https://www.baeldung.com/java-memory-management

CNCF (Cloud Native Computing Foundation). (2024). *State of Cloud Native Development Report*. https://www.cncf.io/reports

Docker Docs. (2023). *Docker documentation*. https://docs.docker.com

Flutter Documentation. (2024). *Flutter docs*. https://docs.flutter.dev

GeeksforGeeks. (2025). *Iterator Design Pattern*. https://www.geeksforgeeks.org/iterator-pattern

Google Cloud. (2024). *Distributed application architecture*. https://cloud.google.com/architecture

Google Firebase. (2024). *Firebase backend services*. https://firebase.google.com/docs

JetBrains. (2024). *IntelliJ IDEA features and JVM tooling*. https://www.jetbrains.com/idea

MDN Web Docs. (2022). *Web standards and browser compatibility*. https://developer.mozilla.org

MDN Web Docs. (2023). *Using REST and WebSockets*. https://developer.mozilla.org/en-US/docs/Web/API

Microsoft Docs. (2023). *Windows and Windows Server documentation*. https://learn.microsoft.com

Microsoft Docs. (2023). *HTTPS configuration and Windows networking*. https://learn.microsoft.com

Mozilla Security Docs. (2023). *Transport layer security overview*. https://infosec.mozilla.org

MySQL Documentation. (2023). *MySQL Developer Reference Manual*. https://dev.mysql.com/doc

NGINX. (2024). *NGINX web server guide*. https://nginx.org/en/docs

NGINX Documentation. (2024). *Production server deployment*. https://docs.nginx.com

NIST. (2022). *Role-Based Access Control (RBAC) Guide*. https://csrc.nist.gov/publications

Oracle. (2023). *Java Platform, Standard Edition Documentation*. https://docs.oracle.com/javase

PostgreSQL Global Development Group. (2023). *PostgreSQL documentation*. https://www.postgresql.org/docs

Red Hat. (2023). *Enterprise Linux platform overview*. https://www.redhat.com/en/technologies/linux-platforms

SQLite Documentation. (2024). *SQLite documentation*. https://www.sqlite.org/docs

StatCounter. (2024). *Global operating system market share*. https://gs.statcounter.com

Ubuntu Documentation. (2023). *Ubuntu tutorials and documentation*. https://ubuntu.com/tutorials

Refactoring.Guru. (n.d.). *Singleton design pattern*. https://refactoring.guru/design-patterns/singleton