

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 |
| --- | --- | --- | --- |
| 1.2 | 06/07/2025 | Thomas Sweet | "Went over recommendations 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.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room, a client of Creative Technology Solutions, is seeking to expand its Android-based drawing game, *Draw It or Lose It*, into a web-based game that functions across multiple platforms. The goal is to create a flexible, scalable system that supports multiple teams and players per game instance, while maintaining unique identifiers and names for each entity. This software design proposes a Java-based application using object-oriented principles and common design patterns (Singleton and Iterator) to enforce structure, reusability, and maintainability. This approach allows The Gaming Room to effectively manage gameplay across distributed environments and prepare for future cross-platform deployment.

## Requirements

The Gaming Room has requested the development of a web-based game application that builds on the existing Android app, Draw It or Lose It. The client requires that the game be accessible across multiple platforms in the future, starting with a web deployment. Functionally, the application must support the creation of games that include one or more teams, with each team consisting of multiple players. Each Game, Team, and Player must have a unique name to avoid duplication and ensure proper identification during gameplay. The system must also enforce a constraint that only one instance of the GameService class exists in memory at a time to centralize control of entity creation and ID assignment. The application should be built using a modular, object-oriented structure that promotes maintainability and expansion into distributed environments.

## [Design Constraints](#_2et92p0)

The application must be built in a way that supports a web-based, distributed environment, meaning multiple users will interact with the system concurrently across different devices. A key constraint is ensuring that only one instance of the GameService class exists in memory at any time to manage state and unique identifiers (Singleton pattern). Additionally, to maintain data integrity and usability, names for games, teams, and players must be unique; requiring logic that checks for duplicates before new entities are added (Iterator pattern). The design must also account for future expansion to multiple platforms, so all code must follow standard best practices and modular design for portability and scalability.

## [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)

**The UML class diagram defines a base Entity class, which holds shared attributes id and name. Three classes, Game, Team, and Player inherit from this base class, demonstrating the object-oriented principle of inheritance. Each of these subclasses adds specific behavior relevant to their role in the application. For example, Game can hold a list of teams, and Team can hold a list of players. The encapsulation principle is used to protect internal data, with access to attributes controlled through methods. The GameService class applies the Singleton pattern to ensure a single point of game management, and the Iterator pattern is used within addGame, addTeam, and addPlayer methods to prevent duplicate names. These design choices make the system efficient, easy to maintain, and scalable across platforms.**

**"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** | While macOS can support Java development and web hosting with tools like Apache Tomcat, it is rarely used in production for web-based applications due to licensing, scalability limitations, and lower server market share. It’s more suitable for local development than deployment. | Linux is the most common server-side platform for hosting Java-based web applications. It’s stable, lightweight, highly configurable, and well-supported by cloud infrastructure (AWS, GCP, Azure). It is the recommended OS for this project. | Windows Server can host Java applications but typically requires more overhead and configuration than Linux. It is less commonly used for large-scale Java web deployments but may be acceptable in environments already committed to Microsoft tools. Windows Server incurs licensing costs, which can increase total cost of ownership for The Gaming Room. However, it may be an acceptable choice in Microsoft-focused environments. | Mobile devices are not suitable for server-side hosting due to hardware limitations, inconsistent connectivity, and lack of server-grade resource management. They are intended to act as clients in this application architecture. |
| **Client Side** | Mac support requires ensuring the web app functions properly in Safari and possibly offering macOS-native clients in the future. Development costs and timelines may increase slightly due to platform-specific testing, but web compatibility is straightforward. | Linux desktop users typically access applications through web browsers like Firefox or Chrome. Supporting Linux clients requires minimal extra effort beyond standard browser testing, as long as the web app follows responsive design principles. Developers must ensure responsive design and thorough testing across browsers and Linux distributions, although the effort required is lower compared to native mobile development. | As the most widely used desktop OS, ensuring compatibility with Windows browsers like Chrome, Edge, and Firefox is essential. Windows users may also expect the option of a downloadable desktop version later, depending on feature expansion. | Mobile users on iOS and Android will need optimized web interfaces or native apps. Web-based UI must be responsive, and care should be taken to minimize bandwidth and battery usage. Supporting mobile adds complexity but is critical for user reach. |
| **Development Tools** | Java development is supported through IntelliJ IDEA, Eclipse, or NetBeans on macOS. Tools like Xcode are also available for future native macOS/iOS development. Terminal, Git, and Homebrew streamline tool installation. | Linux supports a full range of Java development tools, including Eclipse, IntelliJ, VS Code, and build tools like Maven or Gradle. It offers powerful command-line tools and excellent package management via APT or YUM. | Windows supports all major Java IDEs and tools. Developers may encounter path and environment configuration issues more frequently, but tools like WSL (Windows Subsystem for Linux) can bridge the gap. Common Java development happens on Windows. | Mobile platforms require Android Studio for Android and Xcode for iOS. Java/Kotlin is used for Android native development, while Swift is used for iOS. Cross-platform tools like Flutter or React Native can streamline development for both. Developing native apps for Android and iOS requires two separate toolchains (Android Studio and Xcode), often necessitating either cross-platform tools (Flutter, React Native) or multiple development teams. Licensing costs for tools are low, but iOS requires a $99/year Apple Developer account. |

## 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**: The recommended operating platform for expanding *Draw It or Lose It* is a Linux-based web server environment. Linux offers strong support for Java applications, is widely used in enterprise web deployments, and provides excellent performance, scalability, and cost-efficiency. It is also the most flexible for setting up distributed systems and supports a wide variety of frameworks and cloud deployment tools.
2. **Operating Systems Architectures**: The chosen Linux platform supports both monolithic and microservices architectures. For this project, modular service-based architecture is ideal, as it allows each part of the application (e.g., game management, player tracking) to be maintained and scaled independently. This aligns well with the object-oriented structure of the Java codebase and prepares the application for future cloud-based scaling.
3. **Storage Management**: A relational database like MySQL or PostgreSQL is recommended for storing game, team, and player data. These systems allow enforcement of unique constraints and support transactional operations, which is essential for preventing data duplication and preserving consistency. Java integrates easily with these systems through JDBC or ORM frameworks like Hibernate.
4. **Memory Management**: The Java Virtual Machine (JVM) handles memory management on the server through garbage collection and runtime optimization. The Singleton pattern used for GameService ensures that only one instance of the game manager exists in memory, which helps reduce memory overhead and guarantees consistent ID tracking. Proper memory handling is further ensured by encapsulating entity data and minimizing unnecessary object creation.
5. **Distributed Systems and Networks**: To enable communication between different devices and platforms, the system will use a **RESTful API** layered on top of the Java backend. This allows mobile and web clients to interact with the server consistently. The architecture should also account for **fault tolerance**, such as retry mechanisms, data caching, and proper session management, to handle network issues or outages. Load balance and cloud deployment (e.g., AWS, Azure) can further enhance distributed performance.
6. **Security**: Security will be enforced at both the data and transport layers. Secure communication protocols like HTTPS will protect data in transit, while authentication mechanisms (such as OAuth or token-based login) can safeguard access to game sessions and user data. The system should also validate all client input to prevent injection attacks or malformed requests. Linux servers offer built-in user privilege controls and firewalls that further support these security goals.