

# Draw It or Lose It

LaTisha Burns

# **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 5**

**Recommendations 6**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/21/2025 | LaTisha Burns | Summary of essentials needed to execute project such as requirements, constraints, and recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions (CTS) has been commissioned by The Gaming Room, to develop a web-based game that is compatible across multiple platforms, evolving from their existing mobile application, *Draw It or Lose It*, which is currently available exclusively on Android. The game’s primary objective is to support multiple teams and players in a drawing-based guessing challenge. Development will focus on clean, well-structured Java code that follows object-oriented design principles. The final product must operate seamlessly, scale easily, and support multiple users across various platforms.

## Requirements

* A game can have one or more teams participating.
* Each team will consist of multiple players.
* Game and team names must be unique for users to verify their availability.
* Only one game instance can exist in memory at a time, achieved by using unique identifiers for each game, team, or player.

## [Design Constraints](#_2et92p0)

The key constraints for this project include the need to develop a game that functions across multiple platforms and within a web-based, distributed environment. It must be scalable and responsive. To meet these needs, specific design patterns are required: the **singleton pattern** (to ensure only one instance runs in memory) and the **iterator pattern** (to safely manage object lists and avoid duplicates). Unique names and IDs must be enforced, and the solution must satisfy all client requirements for each platform.

## [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 diagram below shows how the ‘Draw It or Lose It’ game uses **object-oriented programming principles** to organize its classes. The Entity class is the base for Game, Team, and Player, and includes shared fields like id and name. This demonstrates **inheritance**, reducing duplication and promoting consistent access to common attributes.

Each Game contains a list of Team objects, and each Team contains a list of Player objects. This demonstrates **composition**, where complex structures are built from simpler components.

The GameService class uses the **singleton pattern** ensuring only one instance exists in memory by using a private constructor and the getInstance() method. This fulfills the requirement for a single game controller instance at a time.

Methods such as addGame(), addTeam(), and addPlayer() use the **iterator pattern** to prevent duplicate names by checking existing lists before adding new entries. These lists (games, teams, and players) are privately managed, showing **encapsulation** by restricting direct access and exposing only public methods.

The Entity class also includes an overloaded constructor alongside a default one, demonstrating **polymorphism**, by allowing objects to be created in different ways. **Abstraction** s used to hide internal implementation details, such as how teams and players are created. Users simply call addTeam() or addPlayer() without needing to know how the objects are constructed.

Overall, the UML design supports a scalable and secure application by using **inheritance, composition, encapsulation, abstraction**, and well-known design patterns.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Characteristics:  Easy to use, consistent, can run multiple OSs.  Advantages:  Great for testing and small-scale hosting.  Weaknesses:  Limited server tools and hosting flexibility. | Characteristics:  Open-source, stable, and secure.  Advantages:  Ideal for web hosting, strong community support.  Weaknesses:  Can be harder to configure for beginners. | Characteristics: Widely used and well-supported.  Advantages:  Easy setup and broad compatibility.  Weaknesses: Higher exposure to security risks. | Characteristics:  Platform-independent and portable.  Advantages:  Accessible from anywhere with an internet connection.  Weaknesses:  Not suitable for hosting full applications; limited to light client use. |
| **Client Side** | Excellent for testing on Apple hardware. More expensive and less common for deployment. | Requires more technical skill. More common in enterprise settings. | Most common for users. Easy to support and highly compatible. | Must support a wide range of devices, OS versions, and screen sizes. Requires responsive design and optimization. |
| **Development Tools** | Uses Swift as the primary language. Xcode and Xcode Cloud are the main IDEs. | Offers extensive tools like Docker, IntelliJ, Eclipse, Java, Git, and powerful command-line utilities. | Compatible with popular tools like Visual Studio, IntelliJ, Eclipse, Java, and Git. Best suited for Windows environments. | Java is commonly used, along with Kotlin and Swift. Android Studio and Xcode are the main IDEs depending on platform (Android/iOS). |

## 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 system for this project is Windows. It is the optimal choice due to its capacity to seamlessly integrate with the existing Android application. Windows accounts for approximately 90% of the global OS user base, providing access to a broad range of developers, skill sets, and tools essential for completing a cross-platform application.
2. **Operating Systems Architectures**: Windows employs a layered architecture that distinguishes the user interface, kernel, and hardware layers. This design enhances system stability and facilitates efficient multitasking. Additionally, it supports both 32-bit and 64-bit applications, rendering it suitable for both development and hosting of games. The architectural framework of Windows also simplifies the management of updates and the integration of services such as web hosting and cloud storage.
3. **Storage Management**: For a Windows-based environment, it is advisable to utilize a cloud-based storage solution such as Microsoft Azure Blob Storage or SQL Server. These options offer high reliability, automatic backups, and easy scalability. Furthermore, they facilitate remote access and can be easily integrated with the Java application through APIs, rendering them particularly well-suited for managing user data and game state information.
4. **Memory Management**: Windows uses virtual memory and paging. When RAM is full, the system allocates part of the hard drive to keep programs running smoothly. It also tracks memory usage for each process, thereby preventing crashes. This method is especially beneficial for applications like Draw It or Lose It, where many users may be active simultaneously, as it helps ensure application stability and responsiveness.
5. **Distributed Systems and Networks**: "Draw It or Lose It" is designed to support multiple platforms through the utilization of RESTful APIs, enabling communication between server and client devices. Data transmission occurs over secure connections (HTTPS) to ensure data integrity and confidentiality. The implementation of load balancers and backup servers is essential to mitigate downtime. In the event of a server failure or network disruption, these backup systems help maintain operations, improving reliability for users.
6. **Security**: To safeguard user data, it is essential for the application to implement HTTPS, user authentication mechanisms, such as logins and passwords, and data encryption protocols. Windows operating systems provide features such as firewalls, antivirus protection, and role-based access control, which collectively restrict access to system files. Moreover, it is critical to employ secure tokens and encrypted connections, such as SSL/TLS, across different platforms to mitigate the risks of hacking and data breaches.