

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.1 | Feb 9, 2024 | John munguia | First Draft |
| 1.2 | Feb 16, 2024 | John Munguia | Second Draft |
| 1.3 | Feb 24, 2024 | John Munguia | Added Recommendations |

**Executive Summary**

CTS undertook the task assigned by The Gaming Room to transform their Android-based game, Draw It or Lose It, into a web-based version. The goal was to enable the game to support multiple teams, each consisting of multiple players, with the restriction that each game instance, team, or player should occur only once.

To achieve this, a singleton creation pattern was implemented to ensure the creation of a single game instance, preventing the occurrence of multiple instances. Additionally, an iterator pattern was employed to manage teams and team members, ensuring that conflicts were avoided during the gaming experience.

**Design Constraints**

Considering The Gaming Room's existing Android-based deployment, CTS decided to extend the game to the web. The chosen tech stack for this extension is Java, selected for its compatibility with web deployment. Leveraging Java, the native Android SDK language, facilitates a seamless transition to the web environment.

Review and potential extension of existing APIs catering to the Android platform were identified as crucial steps. These actions were necessary to adapt the APIs for mobile usage and ensure a smooth integration into the web-based version.

**System Architecture View**

Although not an immediate requirement for the current project, acknowledging the importance of describing system and subsystem architecture becomes essential for future endeavors. This section serves as a reminder that detailing the application's architecture, including physical components or tiers, is crucial. Providing a logical topology of communication and storage aspects aids in understanding the overall architecture.

**Domain Model**

The proposed design's UML diagram, portraying the relationships and structures of the domain model, is presented below. This visual representation serves as a blueprint for the design and implementation of the web-based Draw It or Lose It game.

**"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.**

The application is centered around a primary driver class responsible for initializing the creation of games, teams, and players. The creation process is orchestrated by the GameService class, which adheres to a singleton design pattern, ensuring the existence of only one instance of the GameService class in the memory at any given time.

To enforce this restriction, the GameService constructor is set to private, preventing external instantiation. The exclusive means of creating a GameService instance is through the getInstance() method. This method checks whether GameService has been initiated and starts it only if it is not currently present in memory.

Once the GameService is operational, the driver class can invoke the addGame() method. Employing the iterator pattern, addGame() prevents the creation of Game objects with similar names. The newly created Game object is then added to the List of games.

Subsequently, teams can be added to the game using the addTeam() method. Similar to addGame(), addTeam() utilizes the iterator pattern to avoid the addition of teams with identical names. The freshly created Team object is added to the List of teams.

Following the team creation, players can be added to the team with the addPlayer() method. Similar to the previous methods, addPlayer() uses the iterator pattern to prevent the addition of players with duplicate names. The newly created Player object is then added to the List of players.

The Game, Team, and Player classes are all subclasses of Entity, which contains two protected attributes: id and name. The default constructor is also protected, ensuring that null objects are blocked during creation, and only the overloaded constructors can be utilized.

The UML diagram reflects the application's utilization of various object-oriented programming techniques. Polymorphism and inheritance are evident in the extension of the Entity class and the overloading of constructors. Encapsulation and Abstraction are showcased in the techniques used to add teams, where a Team object cannot be directly created due to blocked constructor access, but users can still achieve this through the addTeam() method without knowing the internal workings.

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac systems, known for security and developer support, face challenges in cost, hardware options, and server market share for hosting web-based applications. | Mac systems, recognized for security and developer support, encounter challenges in terms of cost, hardware choices, and server market share when hosting web-based applications. | Windows systems, known for user-friendly interfaces and software compatibility, provide ease of use for hosting web-based software; however, drawbacks include licensing costs, security vulnerabilities, and higher resource usage. | Windows systems, valued for user-friendly interfaces and software compatibility, offer ease of use but face challenges in licensing costs, security vulnerabilities, and resource usage when hosting web-based software. |
| **Client Side** | For supporting multiple client types on Mac, consider the associated costs, time commitments, and expertise required in software development. | For Linux support across multiple client types, assess software development considerations, including costs, time, and expertise. | To support various client types on Windows, essential software development considerations involve assessing costs, time commitments, and the expertise required for successful implementation across diverse client environments. | For supporting multiple client types on Mobile Devices, crucial software development considerations include assessing costs, time commitments, and the expertise required to ensure effective implementation across diverse mobile platforms. |
| **Development Tools** | For Mac deployment, use Swift or Objective-C with Xcode as the primary IDE, and consider tools like Homebrew and CocoaPods for efficient development. | To deploy software on Linux, use languages like C, C++, Python, with IDEs such as Visual Studio Code or Eclipse, and tools like GCC and package managers like APT or YUM. | For building software to deploy on Windows, relevant programming languages include C#, C++, and .NET languages, with popular IDEs like Visual Studio. Tools such as MSBuild aid in compilation, and NuGet serves as a package manager for dependency management. | For Mobile Devices, use Swift for iOS, Kotlin/Java for Android, with tools like Xcode and Android Studio; consider cross-platform frameworks such as React Native or Flutter, and use Gradle or CocoaPods for dependency management. |

## Recommendations

1. **Operating Platform**: To expand Draw It or Lose It to other computing environments, consider adopting a cross-platform development approach using technologies like Unity or HTML5, allowing seamless integration across various operating systems such as Windows, macOS, Linux, and mobile platforms.
2. **Operating Systems Architectures**: The chosen operating platform architectures involve adopting a cross-platform development strategy, leveraging technologies such as Unity or HTML5. These frameworks facilitate the creation of software that can seamlessly run across diverse operating systems, including Windows, macOS, Linux, and mobile platforms like iOS and Android. This approach ensures a unified architecture that enhances scalability and accessibility across a wide range of computing environments.
3. **Storage Management**: For the recommended cross-platform operating strategy, employing a versatile and scalable storage management system like Firebase, AWS DynamoDB, or SQLite can offer seamless integration and compatibility across different computing environments. These systems provide efficient data storage, retrieval, and synchronization capabilities, catering to the diverse needs of Draw It or Lose It across multiple platforms.
4. **Memory Management**: The recommended cross-platform operating strategy, through platforms like Unity or HTML5, employs automated memory management techniques such as garbage collection and resource pooling for Draw It or Lose It software, optimizing memory utilization and stability across diverse computing environments.
5. **Distributed Systems and Networks**: For cross-platform communication in Draw It or Lose It, implement a distributed software architecture using technologies like RESTful APIs or WebSockets. Consider dependencies on network connectivity and potential outages, incorporating robust error handling and monitoring for reliable communication between devices.
6. **Security**: Secure user information across platforms by implementing encryption, secure data transmission, and leveraging the recommended operating platform's built-in security features like authentication and access controls. Regularly update security measures for comprehensive protection in Draw It or Lose It.