

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_a5po5ct8gfd6)

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

[**Document Revision History 2**](#_13o7m5cdt2s5)

[**Executive Summary 3**](#_bbnbj0axd7z9)

[**Requirements 3**](#_dfl04655wgmm)

[**Design Constraints 3**](#_17r0q7kfacyx)

[**System Architecture View 3**](#_6zp9v0nfn6ki)

[**Domain Model 3**](#_jke4n72n5izm)

[**Evaluation 4**](#_ddpl37cvfzzh)

[**Recommendations 5**](#_q14odubuz86n)

## [Document Revision History](#_gt6ivl6x0n9g)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/15/2025 | Malaki Guenther | Initial draft of software design document for Draw It or Lose It, including Executive Summary, Requirements, Design Constraints, and Domain Model. |

**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](#_ckk1oc7oaodi)

The Gaming Room seeks to develop "Draw It or Lose It," a web-based game where teams compete to guess puzzles based on stock drawings rendered over 30 seconds across four 1-minute rounds, with a 15-second steal phase for other teams if the active team fails. The client requires a single game instance in memory, unique game and team names, and support for multiple teams with multiple players. We propose a Java-based application using the Singleton pattern to ensure a single game instance and the Iterator pattern to validate unique names. The system leverages object-oriented principles, including inheritance via an Entity base class and composition for team-player relationships, to create a modular, scalable solution. Designed for a web-based distributed environment, initial development focuses on client-side logic, with future integration into a server-side framework like Spring Boot for cross-platform compatibility, ensuring robust functionality and a foundation for enhancements like stock image rendering.

## Requirements

* *Support multiple teams, each with multiple players.*
* *Ensure unique game and team names with validation.*
* *Maintain a single game instance in memory using unique identifiers.*
* *Implement four 1-minute rounds with 30-second drawing rendering and a 15-second steal phase.*

## [Design Constraints](#_xwpv6g63bwnf)

* **Cross-Platform Compatibility**: Requires consistent performance across browsers, necessitating WebAssembly or server-side framework integration.
* **Single Instance**: Singleton pattern ensures one game instance, requiring thread-safe implementation in a distributed environment.
* **Name Uniqueness**: Iterative checks for unique names may impact performance for large datasets.
* **Real-Time Rendering**: 30-second drawing rendering demands precise client-server synchronization.
* **Scalability**: Efficient memory and data handling needed to support multiple teams and players.

## [System Architecture View](#_vftrl990w0mi)

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](#_4lha5pknrtcg)

The UML class diagram for "Draw It or Lose It" defines a modular structure using OOP principles: the **Entity** base class (with id and name) enables **inheritance** for **Game**, **Team**, and **Player**, which use **composition** to manage teams within games and players within teams, respectively, while **encapsulation** ensures data integrity via private attributes and public getters; the **GameService** class employs the **Singleton** pattern to maintain a single instance managing a list of games and the **Iterator** pattern to enforce unique names in addGame(), addTeam(), and addPlayer(), efficiently meeting requirements for a single game instance, unique identifiers, and multiple teams/players, with **ProgramDriver** and **SingletonTester** facilitating testing.

**"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](#_6984w6umt84g)

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’s Unix-based system supports Java and Spring Boot well, but higher costs limit scalability. | Linux’s open-source stability excels for Java-based web hosting, though configuration needs expertise. | Windows supports Java and Spring Boot but is less cost-effective due to licensing fees. | Mobile devices lack the power for server-side hosting of Java-based web applications. |
| **Client Side** | Mac requires browser testing for Safari, increasing development time with moderate expertise needs. | Linux supports browsers but limited adoption reduces testing priority, needing moderate expertise. | Windows’ large user base demands extensive browser testing, with moderate costs and expertise. | Mobile devices need responsive design, increasing development time and expertise for diverse devices. |
| **Development Tools** | Mac supports Java with IntelliJ IDEA and Maven, but some tools may have licensing costs. | Linux offers open-source IDEs like Eclipse and Maven for Java, requiring command-line skills. | Windows uses IntelliJ IDEA and Maven for Java, with potential costs for premium IDEs. | Mobile web apps use React and Visual Studio Code, requiring complex Java integration tools. |

## 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**: Windows is recommended for hosting "Draw It or Lose It" due to its widespread popularity, as most computers come pre-installed with Windows, reducing setup costs despite higher licensing fees compared to Linux, which offers similar development capabilities but less market share.
2. **Operating Systems Architectures**: Windows’ NT kernel architecture provides robust support for Java-based applications using Spring Boot, with efficient multitasking and networking to handle multiple client connections for real-time game interactions.
3. **Storage Management**: Use a relational database like Microsoft SQL Server on Windows, integrated with Spring Boot, to store game, team, and player data, ensuring scalable storage and efficient querying for unique name validation.
4. **Memory Management**: Windows’ virtual memory system, paired with Java’s garbage collection, optimizes memory for the Singleton-based GameService, ensuring a single game instance with minimal overhead for multiple teams and players.
5. **Distributed Systems and Networks**: Implement a client-server model with Spring Boot on Windows, using WebSocket for real-time communication between clients (browsers on Mac, Windows, mobile devices) and the server, with load balancing to handle connectivity issues and ensure reliable game updates.
6. **Security**: Protect user information with HTTPS encryption, OAuth 2.0 for authentication, and role-based access control in Spring Boot on Windows, securing data across platforms while leveraging Windows’ built-in security features like Windows Defender.