Draw It or Lose It

# CS 230 Project Software Design Template

Version 1.0

## Table of Contents

**Table of Contents 1**

[**Document Revision History 1**](#_Toc115077319)

[**Executive Summary 1**](#_Toc115077320)

**Evaluation 2**

[**Requirements 3**](#_Toc115077321)  
 [**Design Constraints 3**](#_Toc115077322)

[**Domain Model 3**](#_Toc115077324)

[**Recommendations 4**](#_Toc115077326)

## Document Revision History

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/16/25 | Christopher Meglen | Initial release with complete Executive Summary, Design Constraints, and Domain Model sections. |

## Executive Summary

The Gaming Room wants to take their mobile game, Draw It or Lose It, and make it available on more platforms, basically turning it into a cross-platform web app. To get things rolling, this project lays the groundwork for a scalable Java-based version of the game. The code uses some key design patterns, like Singleton and Iterator, to keep the game logic organized and enforce rules like 'only one game instance in memory' and 'no duplicate team or player names. This document covers the structure, design decisions, and technical constraints involved in building the initial version. Once this is running smoothly, expanding it into a fully web based system should be straightforward.

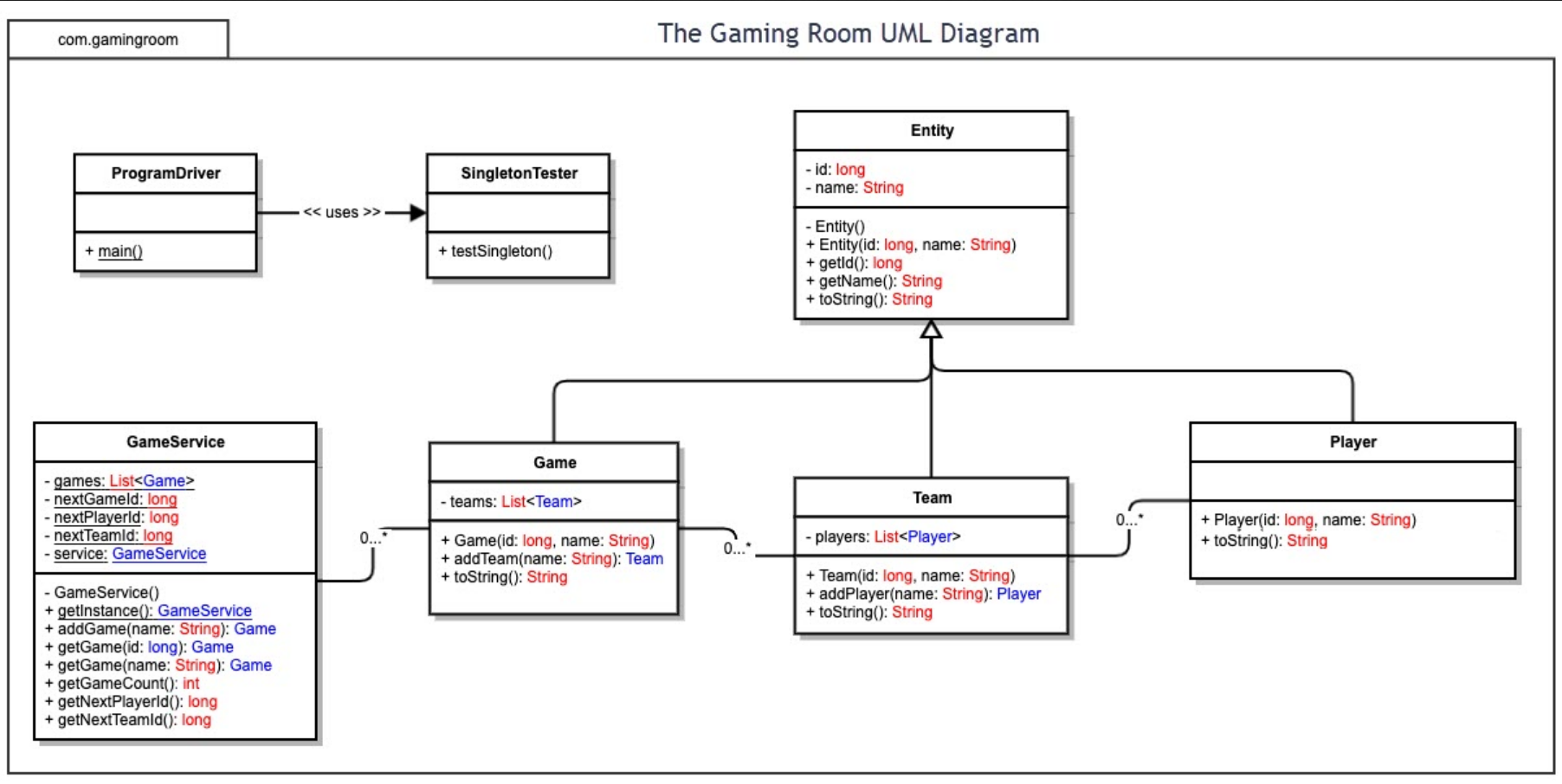
## Requirements

The game needs to be built with a web-based distributed system in mind. That brings a few design rules to the table:  
• Only one GameService should exist at a time,hence the Singleton pattern.  
• All teams and players need to have unique names,checked with iteration over existing data.  
• It has to scale for multiple teams, players, and rounds without breaking.  
• The architecture should play nice with web services down the line, like REST APIs.  
These constraints shaped the current class structure and logic. Everything was built to be clean, lightweight, and ready for web integration later.

# Design Constraints

The design of the 'Draw It or Lose It' application must adhere to several key constraints. These include using Java version 1.8 to ensure compatibility with course tooling, limiting the use of external libraries to those specified in the project requirements, and maintaining a modular code structure to support future maintenance and scaling. Performance constraints also apply, requiring the application to respond in real-time to player inputs. Additionally, platform-specific constraints, such as UI responsiveness on mobile versus desktop, must be addressed while preserving a consistent user experience.

## Domain Model



The UML diagram lays out how the different parts of the game system are connected. At the top is an abstract class, Entity, which holds shared values like ID and name. Game, Team, and Player all inherit from it, which cuts down on duplicated code.

Game contains a list of Teams, and each Team contains a list of Players. These relationships are built into the class structure. GameService acts as the main control center, making sure only one instance exists while managing the creation of games, teams, and players. It uses the Singleton pattern to limit instantiation and loops through existing data to prevent duplicate names. The design follows solid object-oriented principles and makes it easy to scale or move into a web-based system later.

## Evaluation

| Development Requirements | Mac | Linux | Windows | Mobile Devices |
| --- | --- | --- | --- | --- |
| Server Side | Linux is ideal for hosting web applications. It's fast, secure, customizable, and widely supported in production environments. Its command line tools and package management are well-suited for back-end deployment. | Windows is commonly used in corporate environments. It supports IIS for hosting, .NET, and other tools. However, it's more resource-intensive and less flexible than Linux. | Mobile devices are not suitable for server hosting. They lack the memory, persistent connectivity, and processing power needed to run or serve web applications effectively. |  |
| Client Side | Linux is free and flexible but has a steep learning curve. Software must be tested for multiple distributions, and support for commercial tools may be limited. Ideal for open source development. | Windows is widely used and has strong IDE support like Visual Studio. It's developer friendly, but platform updates and dependency issues can cause delays. | Mobile development requires testing across Android and iOS. Tools like Android Studio and Xcode are required. Development is time-consuming and expertise-intensive due to fragmentation. |  |
| Development Tools | Linux supports tools like Eclipse, IntelliJ, NetBeans, and terminal-based compilers. Languages include Java, Python, JavaScript, and shell scripting. Docker is commonly used for deployment. | Windows developers use Visual Studio, Eclipse, IntelliJ, and tools like WAMP. Languages include Java, C#, Python, and JavaScript. It supports Git, CLI tools, and remote debugging. | Mobile tools include Android Studio for Android and Xcode for iOS. Cross-platform tools like Flutter and React Native are also popular. Java, Kotlin, Swift, and Dart are common languages. |  |

## 

## 

## Recommendations

1. Operating Platform: Linux is the best choice for expanding Draw It or Lose It. It’s widely used in production environments, open-source, lightweight, and works well with cloud-based infrastructure. It supports popular frameworks, web servers, and containerized deployments, making it scalable and cost-effective.
2. Operating Systems Architectures: The proposed architecture follows a standard multi-tier model: client devices (browsers or apps) communicate with a web server running the game logic, which interacts with a database layer. Linux supports this layered architecture cleanly, using tools like Apache/Nginx for web services, Java for the logic layer, and MySQL or PostgreSQL for persistent storage.
3. Storage Management: For storage, a cloud-based database like Amazon RDS or Google Cloud SQL is recommended. These provide scalability, automatic backups, redundancy, and ease of integration. If deployed on-premises, a PostgreSQL database on a Linux VM would offer robust transaction support and schema management for teams, players, and game states.
4. Memory Management: Linux handles memory using paging and virtual memory, along with solid support for garbage-collected languages like Java. For this application, the server environment will benefit from Linux’s efficient memory allocation, cache management, and ability to handle multiple processes without major performance hits.
5. Distributed Systems and Networks: Draw It or Lose It should use a RESTful API to communicate between the web client and the game server. These services can be hosted across multiple Linux VMs or containers. Load balancers and redundancy ensure uptime, while tools like Docker and Kubernetes help manage distributed workloads across servers. This allows the game to scale and survive outages.
6. Security: Security should focus on HTTPS communication, user authentication, encrypted passwords, and role-based access. Linux offers firewall tools like iptables and integration with SSL/TLS certs. Using OAuth or token-based sessions will protect users during gameplay, and cloud providers can add DDoS protection and automated monitoring.

Mac systems, built on the Unix-based Darwin architecture, provide strong built-in security and stability. On the client side, macOS includes Gatekeeper to prevent unauthorized apps from running, and XProtect to provide malware protection. The system architecture uses sandboxing, System Integrity Protection (SIP), and strict app notarization, which enhances software safety. On the server side, while macOS Server is less common today, Mac-based servers can still be used for web hosting and development, particularly for iOS or macOS-related software testing. However, server-side deployment on macOS is not as scalable or cost-effective as Linux, especially in enterprise environments. This makes macOS better suited as a client platform rather than a server-hosting platform for large-scale applications like Draw It or Lose It.