

<Name-of-Software-Application>

# **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 |
| --- | --- | --- | --- |
| 1.0 | 07/14/25 | Jalen Horn | <Brief description of changes in this revision> |

**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 is developing a cross-platform version of their popular game, *Draw It or Lose It*. This game, inspired by classic television game shows, allows teams of players to guess phrases based on drawings. The new version of the game will be built in Java and hosted as a web-based application. Currently available only as an Android app, the goal is to create a functional prototype that supports web deployment and can eventually expand to other platforms. To support this effort, the application must allow players to create and manage teams, assign multiple players to those teams, and ensure that all data is stored in memory only. The goal of this software design is to outline the approach, structure, and components that will help make this vision possible.

## Requirements

The application must allow users to create new teams, each of which can have multiple players assigned. It is important that every team name is unique, so the system must include logic to prevent duplicate team names. Additionally, all data for teams and players should be stored only in memory, without the use of any external storage like databases or files. The application should provide users the ability to add or remove players from any existing team at any time, and there should be a way to view all teams along with their associated players. To prevent errors, the system must validate user input, such as rejecting blank team names or handling attempts to remove players who do not exist. The solution must be developed using Java and should be designed with the future goal of web-based deployment in mind. Although the initial version runs locally, it must be structured in a way that supports future expansion to multiple platforms, such as mobile or web-based environments.

## [Design Constraints](#_2et92p0)

While designing the solution, there are some design constraints. First, the system must be developed using Java programming language. Second, because this is a prototype, no persistent storage solutions like SQL or file handling can be used. All information must live in memory during runtime. Third, the application must support team and player management in a lightweight environment that simulates how a larger-scale web application would behave. Finally, the design should make it easy to expand functionality later, especially to support web-based access and interactions between multiple users.

## [System Architecture View](#_ilbxbyevv6b6)

The UML diagram shows how the classes in the application relate to each other. The GameService class manages a list of games, and each Game holds a list of teams. Each Team holds a list of players. Both the Team and Player classes inherit from a shared Entity class, which helps reduce code duplication by keeping common data like id and name in one place. This shows the use of object-oriented principles like inheritance and abstraction. The Singleton pattern is used in GameService to make sure only one version of it runs at a time. These design choices make the system easier to manage and help it meet the software requirements, like keeping team names unique and storing all data in memory.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram shows how the main classes in the system are connected. GameService controls the overall system and stores a list of Game objects. Each Game contains a list of Team objects, and each Team has a list of Player objects. Both Team and Player inherit from the Entity class, which holds shared properties like id and name. This use of inheritance demonstrates object-oriented principles such as reuse and abstraction. The Singleton pattern is used in GameService to ensure only one instance is used throughout the program. These relationships and design choices help organizethe code clearlyand support the software’s goals, like storing all game data in memory and ensuring each team has a unique name.

**"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** | Mac is stable and user-friendly, but it's less common for hosting web-based applications in large-scale server environments. It’s more often used for development and local testing. | Linux is one of the most widely used platforms for hosting web-based applications because it's fast, open-source, and highly customizable. It performs well in server environments and is often used in production. | Windows is common in many business environments. It supports a wide range of enterprise applications and has good tools for hosting web-based systems, though it’s not as lightweight or efficient as Linux. | Mobile devices aren’t used to host applications but must connect reliably to the servers hosting the game. The main challenges are ensuring compatibility across different OS versions and handling screen size limitations. |
| **Client Side** | Mac systems are generally reliable but are more expensive and not as common in enterprise environments. | Linux is cost-effective and ideal for customization, but it may require more technical knowledge. | Windows is widely used and very compatible, but software licenses can be expensive. | Development can be more time-consuming because of the need to support both Android and iOS. |
| **Development Tools** | Developers often use tools like Xcode or IntelliJ IDEA on macOS, which work well for Java development, though some tools may have compatibility limitations compared to Linux or Windows. | Tools like Eclipse, IntelliJ IDEA, and other open-source IDEs are very popular on Linux, and Java development is fully supported. | Developers can use Visual Studio, Eclipse, or IntelliJ IDEA to build applications on Windows easily, making it a flexible platform for testing and deployment. | Developers typically use Android Studio for Android and Xcode for iOS, and sometimes cross-platform tools like Flutter or React Native to reduce duplication. |

## 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**: After evaluating the client’s needs for expanding *Draw It or Lose It*, I recommend Linux as the operating platform because it is widely regarded as the best option for scalable web-based applications. It provides excellent support for Java, which is the core language of the application, while also being cost-effective compared to other platforms. Linux is also known for its reliability in both development and deployment environments, making it a strong choice as the foundation for the system.
2. **Operating Systems Architectures:** The system architecture should be based on a 64-bit Linux distribution. This allows for efficient use of memory, better multitasking, and overall faster performance. A 64-bit architecture also supports modern Java-based web servers, which ensures that the application can scale well across different environments and handle the higher demands of a growing user base.
3. **Storage Management**: The current prototype does not require external storage since all data is being stored in memory. However, as the game moves into full production, Linux provides strong support for reliable file systems such as ext4 or ZFS, which are useful for both backups and scaling. This ensures that as more players join, the system can still safeguard data while remaining flexible enough to expand.
4. **Memory Management**: Linux also offers powerful memory management techniques that support the game’s performance needs. Features like virtual memory, swap space, and garbage collection help prevent crashes or slowdowns when the system is under heavy load. Additionally, resource monitoring tools available in Linux environments work seamlessly with Java applications, ensuring that the system can remain stable even during spikes in usage.
5. **Distributed Systems and Networks**: Because the game is designed to run in a distributed environment, it is important that the platform supports distributed systems and networks. Linux can be configured with web servers, load balancers, and container tools such as Docker to maintain efficient communication between clients, whether they are running Windows, Mac, or mobile devices. This setup allows for cross-platform access and ensures that different parts of the system stay connected through API gateways and networking tools.
6. **Security**: Linux is well-known for its robust security features, including user permission controls, firewalls, and frequent security patches. For the application itself, Java programs can be secured with HTTPS protocols, authentication layers, and deployment practices that safeguard information across platforms. Together, these measures provide the necessary protection to ensure the integrity and security of the game environment as it expands.