

<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 | 09/20/2024 | Logan Samples | Creating the document |
| 1.1 | 10/10/2024 | Logan Samples | Updating for project 2 |
| 1.2 | 10/20/2024 | Logan Samples | Updating for project 3 |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has a game named "Draw It or Lose I" that is only currently available on Android. The Gaming Room does not have the experience to set up the environment for the web-based version of the gaming app. This web-based version must have the ability to have one or more teams with multiple players, unique game and team names, and single-game instances.

## Requirements

The Gaming Room needs the application to be web-based and the ability to support multiple unique teams in a single game instance.

## [Design Constraints](#_2et92p0)

1. Web-based compatibility
2. The ability to have one or more teams in a game instance
3. Teams with the ability to have multiple players
4. Unique Game and Team names
5. Only one instance of a game can exists in memory at a given time

## [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)

This program demonstrates encapsulation by maintaining variables and methods in a single class that controls the access to the contents of the class. For example, the Entity class contains a private variable “id” that other classes cannot directly access. The class does have getter methods that allow other classes to indirectly access variables. Encapsulation makes code easier to understand, secure, and easier to manipulate. Inheritance will enable classes to use contents from another class. The Game, Team, and Player classes are inherited from the Entity class in this program. This allows these classes to use methods like “getID()” and “getName()” from the Entity class. Polymorphism requires methods that have multiple uses and enable other classes. Polymorphism is represented by the way classes inherit methods from each other in this program. For example, the “toString()” method is presented in all 4 classes, but the subclasses perform method overriding. Even though these overriding methods have the same name and parameters, they perform an action based on the object referred to in their class. The use of Abstraction includes hiding unnecessary details from other classes and only sharing essential content. The Entity class achieves this by using getter methods to share information with its subclasses while keeping variables and the “Entity” object itself private.

The Entity class is a base class from which Game, Team, and Player classes inherit. The Entity object is like a hub that connects the objects created in the subclasses. This creates a framework in which the subclasses can share common variables and methods, which will help keep consistency and reduce redundancy. The GameService class manages instances of games using a singleton pattern. This class ensures that only one instance of a game is held in the memory at a time. Each game instance can have multiple team objects created in the Team class in a one-to-many relationship. Each team can have multiple player objects created in the Player class with a one-to-many relationship. The ProgramDriver class uses the SingletonTester class to ensure that only one instance is being held in the memory at a time.

**"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: MacOS is a Unix-based operating system that is similar to Linux.    -Advantages: Apple provides a specialized app that makes servers easy to manage. Because it is Unix-based, MacOS can run many Unix and Linux apps natively. MacOS also provides robust security features.  -Disadvantages:  Hardware that supports macOS is much more expensive than alternatives. Many enterprise-grade server management tools are not supported, making macOS an obsolete choice for large servers. | -Characteristics: Linux is known for being a flexible and open-sourced operating system.    -Advantages: Linux is a free operating system that is stable and secure. Linux is very customizable and modular, which makes Linux great for small to large-scale servers.    -Disadvantages: Potential compatibility issues between versions of Linux can occur. Some third-party apps do not support applications that you may need. | -Characteristics: Windows is the most well-known operating system that Microsoft heavily supports.    -Advantages: Automatic updates make maintenance less frequent for Windows servers. Windows supports most third-party applications. Microsoft offers many enterprise services and applications for Windows.    -Disadvantages: Windows can be very resource-heavy compared to other operating systems. Windows is the most common operating system for non-server use, making it the most exposed to malware and other security issues. | -Characteristics: Mobile devices are small, portable, and can access mobile data.    -Advantages: Mobile devices can be easily portable and can access the provided service from a variety of places due to mobile data.    -Disadvantages: Being wireless will greatly slow the service provided by a mobile device. There is not much support for servers on mobile devices. Hardware is typically slower on mobile devices. |
| **Client Side** | MacOS requires expensive hardware only from Apple. MacOS uses many specialized tools and is not a standard operating system for web services. This can increase development time when getting developers comfortable with these systems or finding developers who are. | Linux is very cheap because the operating system and many tools it supports are open-source. Linux is the most common server operating system that supports most applications for operating a server. Linux is very customizable, and developers are usually comfortable with the applications Linux uses. Developers might not be comfortable with the operating system itself and the command line interface. | Licensing for Windows can be very expensive, especially with enterprise versions of Microsoft's provided tools. Most developers use Windows daily and are comfortable using it. Microsoft runs an open-sourced library, ".NET," that can offer solutions, thus speeding up development time. | Using mobile devices as servers can be very costly when using iOS because Apple requires an Apple developer subscription. Many developers are experienced in mobile app development, but mobile device servers are very rare and not supported by many tools. |
| **Development Tools** | Swift is the language that most applications on macOS use, with Objective-C being used for legacy macOS applications. MacOS still supports most programming languages like C, C++, Python, and Java. Xcode is the main IDE developers use for macOS applications and was developed to improve the speed of an application on Apple operating systems. MacOS still supports other popular IDEs, like PyCharm and Visual Studio. | Linux is compatible with almost every programming language. Linux has a variety of popular IDEs like PyCharm, Eclipse, and Visual Studio. Linux has access to Apache HTTP Server, Nginx, and LiteSpeed as tools to create web servers. | Most Windows applications are written in either C, C#, C++, and Java. Microsoft developed Visual Studio, which integrates well with Azure, .NET, and SQL Server services. | Swift is the leading language for creating iOS applications, and Xcode is the best choice for an IDE. Kotlin is the leading language for Android applications, with Android Studio as a popular IDE. Firebase and AWS Mobile provide many services for mobile development, such as app testing and hosting servers. |

## Recommendations

1. **Operating Platform**: Linux would be a great choice for an operating platform. Linux is free with many different versions and great compatibility with many web service tools. Linux is open-sourced and will allow for unique customizations. The flexibility of Linux also allows greater scalability.
2. **Operating Systems Architectures**: Linux uses a command line that offers a very customizable experience with the operating system. There are also many different versions of Linux with different advantages and disadvantages like speed or compatibility with certain applications.
3. **Storage Management**: Since we are using single instances of games and there is no need to record previous games, storage should be handled locally to be simple and secure. This is a cheap solution that only requires a hard drive, which is already needed to run the operating system.
4. **Memory Management**: The operating platform can support a singleton pattern that will meet the requirement of single instances of games using virtual memory. Each game instance has its own location in the memory that will contain data related to the game such as team names and player names.
5. **Distributed Systems and Networks**: A simple and effective way to distribute the game would be to run a server that users can connect to through the web. Creating web-based applications for multiple platforms will be simpler to create and much easier to update. The application takes up less storage space for the user's device and would not need direct updates to the application to update the game.
6. **Security**: Linux is known for being a secure platform, especially for web services. Linux contains many built-in features that keep you secure along with automatically assigning low user permissions. Using a REST API is a secure and flexible way to run the web-based game. REST API ensures security by ensuring that only authorized users can perform specific actions.