

Draw It or Lose It

# **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 | 03/23/2024 | Daniel Leon | Summarizing requirements, constraints, and description of UML. |

**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 wants our company (CTS) to take their Android app “Draw It or Lose It” and create a web-based version. Each game should be able to have multiple teams with multiple players for each team. There should be only one game in memory at a time.

## Requirements

*We will design a singleton pattern to control instances of games. We will also use an iterator to ensure that game, team, and/or player names can not be used more than once at a time.*

## [Design Constraints](#_2et92p0)

The Gaming Room has asked us to create a web-based version of the Android app. Some features may be transferrable, others will have to be re-written. Screen resolutions will need to be updated also. The mobile app being written in Java should make for an easy conversion to the web-based app.

## [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)

The GameService class contains a private constructor and can only be called by the getInstance() method. This method contains the Singleton pattern that allows for only one game with a specific name to exist at a time in memory. This class is associated with the Game class to addGame() to the games list or getGame() from the games list.

The Game class inherits from the Entity class to create a unique id and name for each game. It also calls an addTeam() method to add a new team to the teams list.

The team class inherits from the Entity class to create a unique id and name for each team. It also calls an addPlayer() method to add a new player to the players list.

The Player class inherits from the Entity class to create a unique id and name for each player.

The Entity class contains the getId() and getName() methods that allow its subclasses to create an id and name.

The Singleton Tester class contains the testSingleton() method. This method is used to call the getInstance() method from another class to test if the Singleton is working properly. It then iterates through the games list to ensure that one instance of each game is created.

The ProgramDriver class contains the main() method which is used to call the GameService class with the getInstance() method. Once it does this, it can use the addGame(), addTeam() and addPlayer() methods. It also calls the testSingleton() method to test if the Singleton pattern is working properly.

Multiple principles of OOP are implemented in our program. We have polymorphism demonstrated in the Entity class by making the constructor protected and then creating an overloaded public version of it to enable the subclasses to call it.

We also have an example of inheritance here, where the Game, Team, and Player classes all inherit from the Entity class. The reason for this is because we want each of those classes to be able to set unique id and names, and by inheriting this option from one parent class we save space and have a more organized code.

Abstraction is obviously used throughout the code by using different classes and implementing method calls. The Entity is also an abstract class that only exists as a blueprint to be used by other classes.

And, of course, we have encapsulation where we have private, protected, and public members in our classes. For different reasons, we may not want other classes to have access to our variables or methods. One example is setting our GameService constructor to private. We didn’t want this being called outside its class so to avoid more than one instance being created at a time. We used the Singleton pattern and created a public method called getInstance() as the only access to the GameService class without creating multiple instances.

**"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** | macOS server was discontinued. Most of the available services are bundled in with every regular macOS download. Other services have to be out-sourced. | Linux is an opensource platform with no licensing costs, except if you’re an enterprise-level company. Bigger companies would use SUSE for Linux, costing around $1000 and up.  Uses command line for operations so one would need to be skilled in this area. | Windows Server is more user friendly with its GUI setup, although this is more resource intensive. Costs for the standard edition start around $1000, while the Datacenter edition starts around $6100. | Mobile devices do not possess the required resources or functionality to act as servers for jobs such as what is presented by our client, The Gaming Room. |
| **Client Side** | Cross-browser compatibility testing and basic web standards for HTML, CSS, and JavaScript.  Depending on expertise, cost and time would vary from OS to OS. | Cross-browser compatibility testing and basic web standards for HTML, CSS, and JavaScript.  Depending on expertise, cost and time would vary from OS to OS. | Cross-browser compatibility testing and basic web standards for HTML, CSS, and JavaScript.  Depending on expertise, cost and time would vary from OS to OS. | Cross-browser compatibility testing and basic web standards for HTML, CSS, and JavaScript.  Depending on expertise, cost and time would vary from OS to OS. |
| **Development Tools** | Xcode, SWIFT, Objective-C, HTML, CSS, JavaScript | Visual Studio Code, Eclipse, Pycharm, C, C++, Python, Java, HTML, CSS, JavaScript | Visual Studio Code, Eclipse, Pycharm, C#, Python, Java, HTML, CSS, JavaScript | Xcode for IOS, SWIFT, Objective-C  Android Studio for Android, Java, Kotlin  HTML, CSS, JavaScript |

## 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**: I would suggest Ubuntu as an OS. Linux servers are open source which allows for more flexibility and security, all coming in at a low cost due to no licensing fees.
2. **Operating Systems Architectures**: Linux kernel focusses less on UI and more on utilizing resources for processes which in turn will give the game better overall performance. Being open source allows our client The Gaming Room to have more control over the system itself when it comes to any changes to be applied, instead of having to wait on a vendor such as Windows to come out with an update.
3. **Storage Management**: Storage files for this game amount to a little under 2gb of space which in my opinion can be handled by the Linux OS internal storage itself. Preferably on an SSD for better performance and speed.
4. **Memory Management**: RAM will be the primary focus here as this game will be ran by multiple users at any given time. Docker would work great here as multiple instances of the program will be ran through containers even though it launches once on the host server. Each container can be set with a memory limit so as to control the amount of RAM being used by each user.
5. **Distributed Systems and Networks**: Once again I think Docker would work well here. Being consistent and easy to use, Docker containers as a network for distributed systems can handle running multiple server instances with different internet setups. Simplicity is the winner here.
6. **Security**: Linux allows for ease of access to customization of role-based security permissions. Authentication and authorization is built into the system and can be customized to your liking. This would protect user information and data from other users as well as browsers and extensions trying to access it without proper permissions.