

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_3znysh7)2

[**Executive Summary**](#_2et92p0)3

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

[**System Architecture View**](#_3dy6vkm)3

[**Domain Model**](#_1t3h5sf)3

[**Evaluation**](#_2s8eyo1)3

[**Recommendations**](#_3rdcrjn)5

## [Document Revision History](#_3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/18/2022 | Dustin Runkel | Constraints, Domain Model, Evaluation, Recommendation. |

**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](#_2et92p0)

Our company has been tasked with creating game service management software. The software will manage the instance of the games, teams, and players, in such a manner that games, teams, and players cannot be duplicated. The solution provided is a singleton pattern, which only allows creation of games through its methods. The main service is static, meaning one instance will serve as a container for all games, and no other instance can be created.

## [Design Constraints](#_tyjcwt)

Business Constraints:

* Time
  + We cannot spend years in development. An appropriate schedule must be set.
* Budget
  + We have to keep erroneous costs low and focus on our core market.

Technical Constraints:

* Enough memory to hold game variables and images.
* Network communication speed between server and client
  + pictures cannot be too big (4k)
* Player count in a game cannot be excessive
  + reduced load on servers results in better performance

## [System Architecture View](#_3dy6vkm)

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](#_1t3h5sf)

The provided UML offers many examples of OOP principles. Entity and its subclasses express inheritance, as Entity passes methods and variables to game, team, and player. Game, Team, and Player also exhibit polymorphism, as the toString() function is overridden in all cases. The program must choose the correct method to call at runtime, which is a good example of polymorphism. Encapsulation is expressed in Entity and GameService. Gameservice holds a private counter for the object Id’s and passes a unique one for every issuance through its methods. Entity, and its subclasses keep names and Id’s private to prevent unintended manipulation of the two.

**"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](#_2s8eyo1)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Most servers do not support Mac OS. | Fast and reliable. Git is useful. | Windows servers are reliable and serve many other web based developments well. | Not useful. Hosting servers requires more power and processing power. |
| **Client Side** | Application should work fine on Mac. Adjustments to file structure in development shouldn’t be an issue. | Linux is very similar to MacOs. If it works on mac, it should ship on linux too. | Windows is the most popular OS. If the game ships on windows, it should boost sales. | Portability and easy download makes this a viable option. Competing in the mobile market is a good option. Extra development might not be worth the profit. |
| **Development Tools** | Mac has Git built in. Has support for all IDE’s and has xCode. Java would work as most people have some version of java installed. Most other languages are supported. | Same as Mac. Git built in. Most IDE’s supported, no xCode though. Most languages are supported. | Git is not built in. Has support for Most IDE’s. Most languages are supported. | Swift (IOS) or Java for android. Most IDE’s support java. Swift can be supported by some IDE’s |

## 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**: Android/Windows. The ability to use one language for multiple platforms significantly reduces the budget and time required to make the game.
2. **Operating Systems Architectures**: Android/Windows with client-server method to store stock images, host games and store user data.
3. **Storage Management**: Direct Attached Storage for servers is ideal. This allows speed and reliability in retrieving user data and image renderings.
4. **Memory Management**: Java has garbage collection, which allows the recycling of unused memory.
5. **Distributed Systems and Networks**:

With the server-client method, the user will have to download a client to interface with the servers. The client will queue users to find games, and simultaneously test ping and latency. Mobile users using a wireless network might experience disconnections if moving into areas of low coverage. A timer can be placed, where if a user doesn’t return to a match, they are kicked from the game.

1. **Security**:

User data can be stored on an encrypted server, only accessed upon login to the client. HTTPS is a good option for a secure connection over a network.