

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 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

**Recommendations 6**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/21/2025 | Alisha Brayboy | Prepare a software design report for new game |

**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)

We will assist The Gaming Room by developing their successful ‘Draw It or Lose It’ gaming app into a web-based version. To accomplish this goal, we will create a Java application using object-oriented programming. By using a Singleton design framework, we will create an Entity class that provides a base for all other classes that include identifying attributes like ID and Name. This plan would let us fulfill the software requirements and create a functional app.

## Requirements

* The ability to have one or more teams involved in the game
* Each team will have multiple players assigned to it
* Game and team names must be unique
  + Allow users to check whether a name is in use when choosing a team name
* Only one instance of the game can exist in memory at any given time
  + Create unique identifiers for each instance of game, team, and player

## [Design Constraints](#_2et92p0)

* Must be a web-based distributed environment. Meaning this app will be available to be played across the internet and needs to be able to support multiple users.
* Must be created in Java Programming language. Being limited to Java also limits the frameworks, tools, and libraries we can use in development.
* It’s our plan to use object-oriented programming so the code will be easier to maintain in the long run and promote scalability. This presents a constraint in the initial design because we will have to be very careful when coding classes to ensure correct inheritance and encapsulation.

## [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 model shown below provides a blueprint of how we plan to set up this program. The ‘ProgramDriver’ class includes the main method which is what will be called on when users start the program. The main method will then call on the other classes when needed, utilizing the SingletonTester class to ensure there is only one object in use at a time so the program can maintain its singleton behavior.

The ‘Entity’ class is the parent class to most of the other classes used in the program. It showcases the inheritance aspect of Object Oriented programming by holding common attributes and methods that its children classes will use. The children’s classes of the ‘Entity’ can be described as follows:

‘GameService’ class manages all the games hosted by this app. It supports the ongoing list of games and allows users to add games to that list. This class also contains methods for counting how many games are currently in the list and methods to retrieve games by unique ID or name.

‘Game’ class is used to represent a single game in this application. Maintaining a list of teams that participate in that game. It includes methods to add Teams to the list of teams participating in this game and retrieve game data and display it as a String.

‘Team’ class represents a single team in the list and retains a list of players on each team. It includes methods for adding players to a team as well as retrieving game data and displaying it as a String.

‘Player’ class represents each individual player on a team. It provides method to retrieve a string representation of each player.

‘Game’, ‘Team’, and ‘Player’ classes, in particular, exemplify the principle of aggregation in Object Oriented Programming. Each class depends on the others to properly maintain their list.

**"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** | Mac has a Unix based system that helps for hosting web-based apps. Mac OS server includes Apache Web Server and PHP support. But it is more expensive and has limited hardware choices. | Linux is free, highly customizable, with a wide range of server software options, and known for its security. However, it will need a higher level of technical expertise to manage than Mac or Windows. | Most used platform for web-based apps. Microsoft tools are widely available and easy to use for a seamless integration. Offers wide range of web server options and supports .NET apps. But the cost is very high due to licensing requirements with Windows. | Mobile devices usually don’t host web-based apps. A mobile device’s app data is usually hosted on a server. Any of the previous OS’s can be used for mobile app backends. |
| **Client Side** | Mac clients must use Apple’s development tools to create and maintain the software. This would have an average cost and time. But developers will have to be skilled in Swift and Objective-C languages. | Cost is low for development. The time and expertise needed can vary widely depending on the distribution used. | Development cost and time can vary. But familiarity in .NET and C# languages are often required | Development on mobile devices requires expertise in different platform specific languages and cross platform tools like Flutter. Cost and time can vary widely depending on how many platforms a developer is targeting for the app. |
| **Development Tools** | Xcode is the primary IDE for Mac applications. It supports Swift, Objective-C, and C++. | Linux can host pretty much any programming language. Most popular and versatile IDEs include Visual Studio Code and Eclipse | Visual Studio Code is the primary IDE for Windows and it supports most programming languages in existence. | For iOS, Xcode and Swift/Objective-C are used. For Android, Android Studio and Kotlin/Java are common. |

## 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**: Based on the information laid out in the Evaluation section I would recommend Linux as our primary operating platform. Its cost effectiveness is unsurpassed and there are a variety of ways to keep other project constraints limited. By choosing the right distribution we could also utilize a plethora of tools and frameworks to make the technical development a bit easier.
2. **Operating Systems Architectures**: I would recommend a multi-tiered architecture system that separates the presentation, application, and data of the app. Where the presentation tier displays the user interface, the application tier handles the coding logic, and the data tier manages the data storage. This method helps with maintaining the app long term and efficiently updating it in the future
3. **Storage Management**: I would recommend an SSD (solid-state drive) for the servers storage. SSDs are faster than hard drives and can hold more data in a physically smaller device.
4. **Memory Management**: Linux uses a paging system for memory management. It works by dividing virtual memory (i.e. RAM) into fixed blocks called pages and the Linux memory manager tracks the used and available pages. The less used pages are swapped to the disk to free up virtual memory space for the user. Linux also users demand paging, meaning only the pages a process needs are loaded into physical memory initially.
5. **Distributed Systems and Networks**: I would recommend the use of REST APIs for communication between various platforms. REST APIs are already widely used for web apps to enable communication over HTTP. And Linux already has REST APIs built into the platform.
6. **Security**: Linux provides great support for security, such as firewalls access control. Linux is known for being less vulnerable to security breaches due to how niche the platform is, among other reasons. To further protect user information on various platforms we should also use SSL/TLS encryption for communication. To prevent interception of data transmitted between the server and clients.