

<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 | 03/18/23 | Morgan Getkin | Updated client’s problem, our solution, and design constraints, as well as business and technical requirements. |

## [Executive Summary](#_sbfa50wo7nsh)

*The Gaming Room wants to expand their title “Draw It or Lose It” to additional platforms by creating a web-based version of the game. Our solution is to develop the game using the Java programming language.*

## Requirements

* *Ability to Support One or Multiple Teams*
* *Multiple Players per Team*
* *Unique Identifiers for Game and Team Names*
* *Only One Instance in Memory at any Given Time*

## [Design Constraints](#_2et92p0)

1. *The company wants their app to support one or more teams. This can be accomplished by utilizing a static list of teams and allowing only an instance of a game to add a new team.*
2. *The company wants multiple players to be assigned to each team. This can be accomplished by utilizing a non-static list of players that is associated with each individual team. In other words, each team has its own list of players, and there are multiple instances of the player list. Additionally, only an existing team can add a new player.*
3. *The company wants each game and team name to be unique. This can be accomplished by utilizing an Iteration pattern within the application that checks a list of stored game names and team names. This ensures there are no duplicates, and therefore that each name is unique. This also requires static lists for both games and teams.*
4. *The company wants only one instance of their game to exist in memory at any given time. This can be accomplished by utilizing the Singleton pattern.*

## [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)

*ProgramDriver contains the main method and currently provides some game, team, and player initialization and testing data. It calls the SingletonTester to verify that the Singleton pattern is being effectively and accurately utilized by only having one instance of a game in existence at any given time. The GameService class drives the program by allowing the creation of games (by verifying unique identifiers via Iterator pattern), storing and providing the identifiers for games, teams, and players, implementing the Singleton pattern, and having multiple overridden getters for a given game. The Entity class is the parent class to the Game, Team, and Player classes. The Entity class contains the name and id attributes for its children’s classes, as well as getters and multiple constructors. It also contains a generic toString method that is overridden by each child class. Each child class (Game, Team, Player) has a constructor that accepts a long and a string parameter (as id and name), and an overridden unique toString method. The Game class includes a static arraylist that stores all the existing teams, and the Team has a unique arraylist that stores its existing players per instantiation of the class. Therefore, the Game class also has a method to add a team, which checks against existing teams to ensure a unique identifier, and the Team class has a method to add a player, which checks against existing players within a given team to ensure there are no duplicates. The GameService class allows zero to many instances of Game (although only one exists in memory at any given time), the Game class allows zero to many instances of Team, and the Team class allows zero to many instances of Player.*

**"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 must 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** | The macOS Server has an unlimited user license, is easy to upgrade due to its easy workload distribution across multiple machines and has great support and GUI. However, it only runs on Apple hardware, and to take full advantage of its capabilities, must only be used to develop for and deploy on other Apple products. There are not many third-party applications for help managing your server. | Linux is a free-to-use, open-source operating system that supports multi-user, multi-process, and multi-thread operations, and it boasts high security. However, servers running on Linux require more extensive knowledge to both install and maintain, and do not offer the same level of support as some of the other options. Most servers in existence today use Linux. | Windows Server OS prides itself on its ease-of-use and user-friendly UI, as well as symmetric multi-processor systems and a wide variety of versions available. However, it requires licensing and is more susceptible to security threats due to Windows’ widespread use on consumer systems. | Mobile devices would be the worst option for hosting a web server and would not work for our program. Not only do they have limited resources that would reach maximum utilization in a very short amount of time, but they cannot be upgraded or scaled, their GUI is much more complicated, and you would not be able to maintain a reliable internet connection or a fixed IP address due to your ISP. |
| **Client Side** | For our program to run on Mac operating systems, we may need to write code specific to this OS. That may range from a few lines of code to an entirely rewritten program in a separate language. This could have a large effect on the time, and therefore the cost, required to finish this project. In either case, it would require expanded expertise to incorporate. | To run our program on Linux, we would have a similar situation. Again, ranging from a few lines of code to an entirely rewritten program in a separate language, this could have a large effect on the time and cost required to finish this project. Again, it would require expanded expertise to incorporate. | Running our program on Windows would be as simple as any of the other operating systems. We could use just about any language, and most developers are more comfortable with this operating system. This would mean there would not be as much of a learning curve for development, and time/cost would be cut down. | To support multiple types of clients on mobile devices, we would need to rewrite code in different languages. Not all mobile devices can use the same languages as each other or other systems. Incorporating these systems into our development would have the greatest impact on time and cost for development and would require a big jump in expertise for these systems. |
| **Development Tools** | Since Mac supports Java, we would be able to use the Eclipse environment to code in Java to deploy on Mac. We could use HTTP protocol to communicate with the server, which could utilize a REST API. | Since Linux supports Java, our Java code in Eclipse would work. We could still use HTTP protocol to communicate with the server. | Since Windows supports Java, we can use the same code as Mac and Linux with little to no problems. | To deploy it on mobile devices, we need to write a new set of code. Specifically, iOS cannot run Java, so it would need to be written in Apple’s language, Swift, or Objective-C. This would include Apple’s development environment, Xcode. Android would be able to handle the Java code. |

## 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 recommend The Gaming Room uses a Windows operating system to develop their game “Draw It or Lose It”. Since the game is web-based, it can be accessed by any machine using a modern web browser. Therefore, the code for these different operating systems does not need to deviate from each other too much, if at all. Windows requires the purchase of a license to use their operating system, but it has the largest range of available software, which will benefit the development of this application greatly. It is also more familiar and widely used, meaning the team will not have to learn how to use the operating system, and includes many security features.
2. **Operating Systems Architectures**: The Windows operating system’s architecture is built using a hybrid kernel, meaning that it provides core services such as process and memory management, file system access, and device drivers, and system calls, while also providing process, thread, and memory management. Windows supports a range of user-mode subsystems, including a subsystem which provides a layer of abstraction between the kernel and user-mode applications. Its background services include things such as system management, networking, and security. It is designed to provide customization and flexibility with differing hardware and software without sacrificing strong security and stability.
3. **Storage Management**: An appropriate storage management system is Microsoft Azure, which is a cloud-based system that has many features and integrates very well with the Windows operating system. One efficient feature is file storage that can share files across multiple Windows workstations, which will help the development team tremendously. It is scalable, which ensures that The Gaming Room is not spending any more on storage solutions than they need to. They provide cloud-based backup and recovery services, and boast robust security and compliance features, including encryption, identity and access management, and compliance with industry standards and regulations.
4. **Memory Management**: The Windows operating system uses several techniques for memory management. Of these, “Draw It or Lose It” benefits from the use of virtual memory in conjunction with a paging system. When in use, the operating system will allow RAM to quickly interact with local storage by allocating storage space to memory and sends some less-frequently used memory data to this virtual memory space. When that data is needed, Windows will “page” the data from the storage drive back to memory. This ensures the memory is cleared up for other programs and more frequently used data and allows a seamless use of a web-based game. This game can load all the initial images into memory without worrying that they will take up too much space. Then once a game is started, the additional 30 images for that round can be loaded into the memory cache for quick access. Once the round ends, the program can take advantage of Windows’ garbage collection system to prevent memory leaks and makes sure the system allows memory to be used by other applications.
5. **Distributed Systems and Networks**: Microsoft Azure can take the place of a server by providing the ability to build and deploy our game, as well as a database. We can then create a RESTful API to access the database through standard HTTP requests. Using this type of API allows the different operating systems to communicate with the game’s backend servers in a uniform way. This helps with simplicity of development and maintenance. Azure offers server features such as load balancing (which distributes requests across different servers) and auto-scaling (which automatically adjusts the number of servers needed based on workload). These are beneficial because it will help our multiplayer game remain consistent with different users and create a more seamless experience. Microsoft Azure provides high availability and redundancy, boasting an impressive 99.99% uptime. This means that the only connectivity issues with our game will be on the user’s end.
6. **Security**: To help protect user information, Windows utilizes several features. It has built-in firewall protection and encryption, utilizes the principle of least privilege to protect its data from security breaches or accidental exposure, and offers an anti-virus and anti-malware solution through Windows Defender. Windows also allows biometric authentication using facial recognition or fingerprint scanners, which mitigates the use of insecure passwords. It also has a feature where it will create a virtual storage space for sensitive user credentials to be stored, called Credential Guard. Additionally, Azure provides a range of security features, including firewalls, encryption, and identity and access management. This keeps the system protected from any potential threats and protects data between the server and the client. Both solutions follow industry standards and regulations, and offer automatic updates, which keep the system up-to-date and as secure as they can always be.