

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 | 01/29/24 | Jordan Clark | This initial submission contains the Executive Summary, Design Constraints, and Domain Model for adapting “Draw It or Lose It” from an Android app to web-based 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)

The Gaming Room selected CTS to adapt their Android-only application titled Draw It or Lose It into a web-based game that serves multiple platforms. The web-version should only have one game instance, team, or player in active memory at any given time. The game should also have a unique name, one or more teams with unique names, multiple players within each team, and give users immediate validation when creating a team name.

We will use a singleton pattern in the GameService class to ensure that only one game instance can occur at any time. To implement the pattern, we store the service operation that creates the sole instancein a private static field. Then we will grant public access through instance operators, which will allow the creation of multiple games on a single service. We will use an iterator pattern, within the singleton, to access and search through the list of games. The iterator returns the current instance of a game, or creates a new one, further preventing any duplicate games.

The second implentation of the singleton pattern will allow multiple teams to contain multiple players. We create an Entity class that contains the encapsulating properties that embody the sole instance of an object. The Game, Team, and Player classes inherit this property to grant individuality to each object created from those classes. The Entity also contains an Iterator pattern to travserse its lists. The Game and Team classes inherit these behaviors to iterate through their persepective lists of teams and players, providing users immediate validation when they create their team name.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

The design constraints for this application include web-based distribution, tech stack compatibility, and scalability.

Unlike the Android app, the web-based version needs to be available across multiple platforms to serve a broader audience. This is a business constraint. If the application only works on a particular browser, it negates the efforts to diversify the market reach. The adaptation also poses a technical constraint because it limits the tech stacks compatible with both Android devices and web applications.

Lastly, scalability is a technical constraint that impacts the user's experience, a business constraint. The game service must be able to scale up as more users add games, teams, and players to it. Similarly, the game service shouldn’t hold on to memory for previously held games. Without scalability, users could experience lagged gameplay for easily avoidable reasons.

To address these constraints, we have chosen to use the Java language. Java is the native Android SDK and has popular web frameworks, like Spring, that are easily adaptable. The common code base minimizes the time it will take to refactor existing functionalities. Java is also great for handling multiple users and simultaneous operations, ensuring a responsive user experience while under heavy loads.

## [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 UML model depicts the classes involved in building this application and their relationships.

The ProgramDriver class uses data input to run the program.

The SingletonTester class checks the service and returns the active games to validate the singleton pattern functionality.

The GameService class contains a singleton pattern design to ensure a sole instance of the game service exists at any given time and an iterator pattern to traverse a private list. The class’s private attributes contain a list of games, the next game/player/team id’s, and a the sole instance of the service operation. The private constructor *GameService()* prevents instance duplication; the public behaviors access and return the private attributes of the class.

The Entity class is a parent class containing the private attributes (id, name), a private constructor, iterator behaviors to traverse a private list, and public accessors to the private attrivbutes

The Game class is a child of the Entity class, meaning it conatins a sole instance of itself with the ability to search and return information from a private list. The Game class invokes the service from GameService, holds a private list of teams, and adds teams to that list.

The Team class is another child of the Entity class which contains a sole instance of itself, holds a private list of players, and adds players to that list.

The Player class is another child of the Entity class which holds a sole instance of itself, containing the players id and name.

The object-oriented programming principles utilized within this model include:

Encapsulation: The class attributes control access to their data to ensure that only one instance of the game service, a team, or a player exists. This allows multiple players on multiple teams in multiple games on one service.

Composition: The GameService class has a list of games, the Game class invokes the service of the GameService class. Their relationship is a zero to many because it can host zero to many games. Similarly, the Game class holds a list of zero to many teams, and the Team class holds a list of zero to many players. This utilization of composition relationships emphasizes the existence of the game relies on the existence of the service; the team relies on a game; and a player relies on a team.

Inheritance: The Game, Team, and Player classes inherit their private attributes (id, name) and behaviors (private constructor) from the Entity class. The inheritance principle promotes code reusability, reduces redundancy, and allows similar qualities to be shared among common classes. In our case, each entity can interact with one another in their perspective hierachies.

Polymorphism: We overload the getGame function because it is used to recover different pieces of information from list. The modifications allow us to use the same functionality of getGame without having to write a separate iterator for each piece of information we are looking for. We simiply alter the parameters to simultaneously recover the id, name, or index within a 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)

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** | <Evaluate Mac for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Linux for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Windows for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Mobile Devices for their characteristics, advantages, and weaknesses for hosting a web-based software application.> |
| **Client Side** | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mac.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Linux.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Windows.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mobile Devices.> |
| **Development Tools** | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mac.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Linux.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Windows.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mobile Devices.> |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>