

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc94723243)

[Table of Contents 2](#_Toc94723244)

[Document Revision History 2](#_Toc94723245)

[Executive Summary 3](#_Toc94723246)

[Design Constraints 3](#_Toc94723247)

[System Architecture View 3](#_Toc94723248)

[Domain Model 3](#_Toc94723249)

[Evaluation 4](#_Toc94723250)

[Recommendations](#_Toc94723251) 5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 8/17/2025 | Chris Jabbour | There is no way this is a sophomore level class. |

## [Executive Summary](#_sbfa50wo7nsh)

CTS was asked by The Gaming Room to design a web-based version their Android-based game Draw It or Lose It. The game should allow multiple teams with multiple players per team. Each game-instance, team, or player should only occur once.

A singleton creation pattern has been adopted for object creation to prevent multiple game instances and an iterator pattern will prevent conflicting teams and team members.

## [Design Constraints](#_2et92p0)

The Gaming Room has an existing Android-based deployment of the Draw It or Lose It. CTS has been asked to extend this to the web and thus the tech stack needs to be compatible with web-deployment - Java has been selected for this purpose. Java being the native Android SDK language should ease this new deployment effort.

Any existing APIs serving the Android platform need to be reviewed or extended for mobile usage.

## [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 for the proposed design is shown below.

**"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.**

The application consists of a main driver class that will be used to initiate the creation of the games, teams, and players. The actual creation is through the GameService class and follows a singleton design pattern so that only a single GameService class may exist at any time in memory.

GameService blocks its creation of any instances by setting its constructor to private. The only way to instantiate a GameService is through the getInstance() method. getInstance() checks whether GameService has been started and only starts if it is not present in memory.

Once GameService is running, the driver class is able to call the method addGame(). addGame() uses the iterator pattern to prevent similarly named Game objects from being created. This new Game object is then added to the List *games*.

After the game is created a team may be added to the game with the addTeam() method. addTeam() uses the iterator pattern to prevent similarly named Team object from being added to the game. This new Team object is then added to the List *teams*.

After the team is created a player may be added to the team with the addPlayer() method. addPlayer() uses the iterator pattern to prevent similarly named Player objects from being added to the team. This new Player object is then added to the List *players*.

Game, Team, and Player classes are all subclasses of Entity. Entity has 2 protected attributes: *id* and *name*. The default constructor is also protected so null objects are blocked at creation and only the overloaded constructors may be used.

The designed UML shows multiple Object-oriented program techniques. Polymorphism and inheritance are used in the extension of the Entity class and the overloading of constructors. Encapsulation and Abstraction are also apparent in the techniques used to add teams. A Team object may not be created directly since accessing the constructor is blocked but the user is still able to with the addTeam() method – without the user knowing how that team was added.

## [Evaluation](#_2o15spng8stw)

There are a few potential targets for development, e.g., Macintosh, Windows, Linux, mobile platforms, for either hosting/serving or acting as the client. The following table discusses the strengths and weakness of each platform.

It should b pointed out that the server and client choices are not linked. That is, if Linux was chosen as the server OS windows may still be the preferred option for the client.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Games should not be on Mac. | Linux is the most popular webhosting OS. Because Linux is opensource, maintenance and license costs tend to be cheaper than closed OSes like Windows. | Windows servers are nice because they are GUI based and many applications used in the office will also run on the server – so familiarity is abundant. |  |
| **Client Side** | Games should not be on Mac. | Development in Linux should be straightforward as Java or C/C++ or Python could be the language of choice – which are all commonly used. | Windows is typically developed using C# or .NET. | Mobile devices are not designed to be multi-user. However, design a client application is straightforward. |
| **Development Tools** | Games should not be on Mac. | Linux development may take the form of C/C++, Java, or Python. | Windows is primarily developed using C# and primarily .NET. |  |

## 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**: Windows or Mobile as those are the most widely used OS’s.
2. **Operating Systems Architectures**: Que?
3. **Storage Management**: SSD or M.2’s reign supreme here.
4. **Memory Management**: Don’t melt the ddr5?
5. **Distributed Systems and Networks**:
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.>