

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

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

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/12/2022 | Chris Trimmer | Document creation. Complete the executive summary, design constraints, and domain model. |

**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)

Our client, The Gaming Room, is interested in developing a web-based game that can be played on multiple platforms. They want to base it on their current game, Draw it or Lose It, which is only available on the Android operating system. It works loosely based on the old television show ‘Win, Lose, or Draw’. The game will include multiple players and one or more teams playing the same game instance. For this reason, we will be designing the game using the singleton design pattern. To enhance performance of searching games, teams, and players, we will also be implementing the Iterator pattern to traverse our data structures.

Since the game will be web-based and available on multiple platforms, we will have design constraints in implementing the game. For example, we need to consider a portable language, and a language that has support for RESTful services. We have determined that using Java with Jersey (JAX-RS) API will meet our needs. Furthermore, we will be following the principles of Object-Oriented Programming (OOP) and will therefore design the software based on the four pillars: encapsulation, inheritance, polymorphism, and abstraction.

## [Design Constraints](#_2et92p0)

Specific design constraints are as follows:

* Only one instance of a game can exist at any given time.
* A game must be able to contain one or more teams.
* A team must be able to contain multiple players.
* Names must be unique for the Game instance, teams, and players.
* There must be a way to check whether a name is available or not.
* The game must be web-based and available on multiple platforms.

Since only one instance of a game can exist at a time, we will be implementing the singleton pattern. We will be using array lists to hold game objects: the primary game app will have an array list of game instances; each game instance will have an array list of teams; and each team will have an array list of players. Since game names, team names, and player names must be unique, will have validation in the code to test name creation for each object. This will be implemented using the iterator pattern to search through respective lists and testing for duplicate values.

The game is currently only available on Android. We may need to refactor code for any platform that is not Java-based. Since we are aiming for a web-based distributed environment, following a REST-based approach is appropriate. Java has a built-in API, JAX-RS, which simplifies development of REST-based applications. We will also be using the Jersey API, which extends JAX-RS and provides more features and utilities. Using a RESTful solution, we can ensure the game is widely distributed. Java is also a portable coding language, which supports the requirement for a multi-platform solution.

## [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 game will be developed following the principles of OOP. Below is the UML diagram that depicts the design of the main classes in the game.

**"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.**

First, we annotate each class with its name, public and private attributes and operations. This satisfies encapsulation in several ways. First, by using access modifiers we control how members and operations are accessible to other parts of the program. Second, the data members and operations are contained within the class, essentially “encapsulating” them to the class they are designed to be used for. In effect, everything required to consume the class is contained within it.

Next, we show inheritance hierarchy among Entity, Game, Team, and Player. The classes Game, Team, and Player share common characteristics so it makes sense to create a base class where they can be derived from. The base class is Entity and contains those common attributes and methods. This promotes code-reuse, enables extensibility, and is overall sound programming design.

We also see the association among the GameService, Game, Team, and Player classes. We use association because they are not subclasses of each other, rather they use the class for a purpose. In this scenario, GameService contains a list of Games, Game contains a list of Teams, and Team contains a list of Players. These associations show multiplicity of “zero to many” – meaning that a Game can have zero or more teams, for example.

We are creating the GameService class as a singleton to meet requirements that a game must be a unique instance. This is annotated in the diagram with the static members and static method underlined. The GameService constructor is also labelled as “private”, which means the only way to create a new game instance is via the static method. This prevents multiple instances of the game from being created.

Polymorphism is carried out using the toString() method of the Java base Object class. The toString() method is implemented in the base class Object. Our Entity, Game, Team, and Player classes each override and implement the toString() method in their own way. Namely, they use the toString() method to print details about their respective class objects such as their attributes. For example, a Player object will use toString() to print the Player attributes, while a Team object will use toString() to print team specific objects.

Finally, abstraction is seen at several levels. Firstly, we are modeling our classes based on real-world ideas. This helps organize and express the problem in a way that is easier to understand. Even the simplest of games can be complex to create in code. Abstraction is a way to model the game in a way that is easier to manage and understand. Secondly, we create the data members and operations in a way that hides complexity. For example, in the Game, Team, and Player classes we implement array lists to store their respective associated class objects (Team holds an array list of Players, for example). Consumers of the Team class simply access the players without care or concern about the underlying data structure of the player list.

## [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.>