

CS 230 Project One Game App

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.gjdgxs)1

[**Table of Contents**](#_heading=h.30j0zll)2

[**Document Revision History**](#_heading=h.3znysh7)2

[**Executive Summary**](#_heading=h.2et92p0)3

[**Design Constraints**](#_heading=h.tyjcwt)3

[**System Architecture View**](#_heading=h.3dy6vkm)3

[**Domain Model**](#_heading=h.1t3h5sf)3

[**Evaluation**](#_heading=h.2s8eyo1)3

[**Recommendations**](#_heading=h.3rdcrjn)5

## [Document Revision History](#_heading=h.3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/24/2022 | Christopher Hardy | Add Entity class as the parent to game, team, and player classes |

**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](#_heading=h.2et92p0)

Create a program that allows for players to have multiple teams that do not overlap. This means that there can only be one instance of each game and team in memory at one time. Each team will be able to have multiple players assigned to it.

## [Design Constraints](#_heading=h.tyjcwt)

This application needs to be web based. This means that the application should be available on many platforms. This is why we will use java in this application as many platforms support java based applications.

## [System Architecture View](#_heading=h.3dy6vkm)

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](#_heading=h.1t3h5sf)

The singleton model will allow us to create instances of the game that restrict the amount of instances there are. Specifically only one instance of the game can exist in memory at any given time. The Entity class creates a base class that the Game, Team, And player classes can build upon. Each of these classes will have attributes for adding their respective players and teams while also checking to see if that item already exists.

**"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](#_heading=h.2s8eyo1)

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** | Easy command access that allows for easy access and modification | Great terminal support and is cost efficient. | Large software availability and support | Not a good platform to host a serverside application |
| **Client Side** | Expensive and needs a good deal of knowledge on the system prior to using. | Cost efficient and wide support. Some knowledge needed beforehand | Expensive, but cheaper than Mac? Wide support available | Wide variety of applications and support for mobile client applications |
| **Development Tools** | Most programming languages supported on this OS. Less than others. | Most programming languages supported on this OS. | Most programming languages supported on this OS. | Most programming languages supported on this OS. Less than others |

## 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 a linux or windows based server for their wide support and availability of development tools and applications. If we were to develop for as many platforms as possible we would want to include systems such as gaming consoles, mobile devices and personal computers which each have different operating systems such as Windows, Unix, Mac, and Linux. We would need to cater the game to each platform to enable use of their specific peripherals such as touch screens, keyboard and mouse, or controllers.
2. **Operating Systems Architectures**: There are various options for this but I recommend an x64 base architecture. x64 will allow for much more headroom for things like memory channels. ARM architectures and x86 are not quite as powerful as an x64 based system.
3. **Storage Management**: SSDs and slower long time storage may be needed for this application. depending on the apps usage requirements we may need fast access and ability to store scores for a long period of time. We could even use things like PCI express storage systems or NVME or M.2 storage ssds but they may be more expensive than what this application would even require.
4. **Memory Management**: Memory management tools are widely available and using best practices in java will allow us to help reduce memory issues such as buffer overflows and data breach issues. As previously mentioned an x64 base architecture will allow for better memory options. An x32 system can only allocate half as much memory to a thread compared to an x64 based system.
5. **Distributed Systems and Networks**: With dedicated servers and a common language such as java, we will be able to perform this application on many different devices. Redundancy will be required for continued connectivity. The distributed systems will also have to have communications between systems so that it has shared information that all users have access to.
6. **Security**: Programming best practices and security software and teams should be utilized to ensure data is secure. This is a growing concern in today's environment and needs to be addressed even in the beginning of software development. User information needs to be kept secure and the system needs to be secure itself to prevent people from cheating on the games.