

Project One

# **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 | <08/12/2003> | <Yoel Marquez> | <Start of project> |

**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 goal is developing a scalable, multi-platform game that can handle multiple teams and players. Making sure that only one instance of the game is able to exist in memory at once. Creating and making use of design patterns would allow us to create the program quickly and make is scalable as these design patterns are tested.

## Requirements

*Technical:*

* *A game will have the ability to have one or more teams involved.*
* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
* *Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## [Design Constraints](#_2et92p0)

The application must be efficient with handling multiple teams and players making scalability a potential program. Concurrency, we need to make sure that the game can only be in memory once at any time. We also need to make sure that games, teams, and players are all unique to aid in differentiating them. Making sure that the application can run seamlessly across different web browsers as creating we application has to take in the variety of web browsers that can potentially host this game.

## [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)

**We have an Entity class that will be inherited from by Game, Team, and Player. It holds some basic variables to make the objects unique.**

**The classes that inherit starting with Game, makes use of polymorphism adding on the Entity attributes and methods to create a list of teams of which could be 0 or many and allows the adding of teams,**

**Team similar to Game has a 0 to many relationships with Player as it has a list of players as its attributes with the ability to add players through its method.**

**The player class has the least added attributed only needing the toString() and Player() constructor which is required. Lastly the Game Service which has a List of Game(s) from 0 to many games. This game service class is also where the singleton pater and the iterator pattern is managed.**

**"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** | macOS offers robust security features and development tools but comes at a higher cost. | Linux is open-source, fostering community contributions, but lacks user-friendly experience. | Windows has the largest user base and extensive development tools, though it can be resource-heavy. | Mobile devices face resource constraints but have a vast user base. |
| **Client Side** | macOS tools are comprehensive but expensive; development can be lengthy due to application licensing complexities | Linux tools are open-source, cost-effective, but require high expertise for effective use. | Windows licensing costs are high; ensuring cross-browser support is crucial for application deployment. | Supporting diverse devices and ensuring compatibility are time-intensive processes |
| **Development Tools** | Xcode on macOS offers a comprehensive integrated development environment (IDE) but may be restrictive for non-Apple platforms. | Linux provides a variety of customizable tools like Visual Studio Code, though it has a steep learning curve. | Windows tools, such as Visual Studio, are resource-intensive but offer extensive support for various development needs. | Supporting both Android and iOS platforms requires integration of tools like Android Studio and Xcode. |

## 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 is recommended due to its vast user base and robust development ecosystem, which supports extensive game deployment and scalability.

1. **Operating Systems Architectures**:

Windows dominates the market with widespread compatibility and developer support, making it ideal for reaching a broad audience.

1. **Storage Management**: Utilizing SQL databases like MySQL ensures efficient data handling and scalability for managing user information, game progress, and scores.
2. **Memory Management**: Java’s garbage collection mechanism automates memory management, optimizing performance and scalability for the game.
3. **Distributed Systems and Networks**: Implementing microservices architecture with RESTful APIs will enhance connectivity and facilitate seamless gaming experiences across platforms.
4. **Security**: Enhance security through encrypted data transmission and robust authentication methods, mitigating risks associated with data breaches and unauthorized access.

L