

<Valve’s Offhand Game>

# **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 | <11/18/2024 | <Derek Cun> | <  - added feature to check team names being used  - code updates  - added player and team 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](#_sbfa50wo7nsh)

The purpose of this project is to design and implement a web-based game application that meets the client’s software requirements. The app should use object oriented programming and design patters to make it easy to understand. The primary goal is to create a scalable, efficient, and maintainable application capable of handling entities such as games, teams, and players within a distributed environment.

This game application will let users:

* Create and manage games, teams, and players with unique names.
* Use connected interaction between these entities using efficient data management and search functionalities.
* To achieve this, the software design will incorporate key principles such as inheritance, encapsulation, and polymorphism to organize the system efficiently. Additionally, design patterns will make use of the singleton pattern and ensure the GameService class is instantiated only once, avoiding resource duplication. The iterator pattern will facilitate efficient searching for unique names and ensure smooth data management for adding games, teams, and players.

This solution will address key challenges such as enforcing unique names, ensuring proper entity relationships, and maintaining optimal system performance. The accompanying UML class diagram provides a clear overview of the domain model, detailing how each class interacts to fulfill the client’s requirements.

## 

## Requirements

***Business Requirements****:*

1. Support creation and management of games, teams, and players.
2. Enforce unique names for all entities.
3. Enable efficient name search and validation.

***Technical Requirements****:*

1. Use a base Entity class for shared attributes and behaviors.
2. Apply the **singleton pattern** to the GameService class for single-instance management.
3. Implement the **iterator pattern** for name validation in addGame(), addTeam(), and addPlayer().

## [Design Constraints](#_2et92p0)

**Business Constraints:**

* the system must be scalable to support the creation of multiple games, teams and players
* games, teams, and players should be unique to be able to clearly identify a specific one and avoid confusion
* the app should be easy to use to make it more friendly for its users

**Technical Constraints:**

* data access design should be efficient to ensure the app is performant
* the app must use the singleton pattern to ensure only copy of the class runs at a time
* ensure names for teams, games, and players are unique
* keep data consistent across all servers

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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** | * easy to use and good for smaller scale * less common and can be expensive | * free and reliable * harder to use and manage | * user friendly and compatible with other Microsoft tools * can be expensive | * portable and more accessible |
| **Client Side** | * high quality * less accessible * expensive | * Flexible and cost-effective * has fewer users and support challenges | * widely used * compatible with most softwares | * most portable * multiple platforms so difficult to maintain |
| **Development Tools** | * codes are limited to apple’s systems * xcode | * many free tools are available * requires more technical knowledge | * easy to use tools * more expensive * visual studio | * android studio or xcode compatible * require specific technical skills |

## 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 would recommend using a cloud based platform such as Amazon Web Service or Microsoft Azure, they support the game on different operating platforms and allow for easy scaling.
2. **Operating Systems Architectures**: A cloud platform is compatible with various operating systems which makes it more accessible from a larger audience.
3. **Storage Management**: AWS S3 would keep game data safe, synchronize data across devices, and keep backups.
4. **Memory Management**: Cloud systems automatically handle memory efficiently. They help prioritize resources and clean up unused processes.
5. **Distributed Systems and Networks**: Use APIs to let the game communicate across devices in real time. A CDN would help distribute game data to servers closer to players.
6. **Security**: Use data encryption with HTTPS. Can also use secure logins with authentication to authenticate users and keep user information safe.