

<Name-of-Software-Application>

# **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 | <11/15/2021> | Joseph Triche | Created entity superclass and refactored existing classes  Added create team and player options |

**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)

In order to make sure only one instance of the game service client is running at one time, a singleton type design pattern will be implemented. This means that whenever something attempts to create a new instance of the game service, it will automatically check to see whether or not an instance of the service exists. If it does, a new instance cannot be created. To create unique identifiers for the games, players, and teams, an iterator pattern will be used. This allows us to correspond objects with alphanumeric names or numeric IDs. These are then stored inside a list. To check if a name or ID is already in use, we iterate through the list. If the identifier is not found, a new object is created and conversely if it is found, then the object is not created.

## [Design Constraints](#_2et92p0)

* Web based distributed environment things
* Making sure multiple instances of the client cannot be run
* Making sure each name is unique

## [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 GameService class holds all the current games. The Game, Team, and Player classes inherit from the Entity class. The ProgramDriver class uses the SingletonTester. The GameService, Game, Team and Player classes are connected with a multiplicity of one to many. The OOP principle’s that are employed here are Inheritance, Composition, and Polymorphism. Inheritance is demonstrated by Player, Team, and Game which are child classes of Entity. Polymorphism is used because the Entity class has a method which is implemented differently in all 3 child classes. Finally, Composition is used by the GameService class as part of the singleton design pattern.

**"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** | Mac has a command line which can be used to configure and access the server. | Linux also uses a command line which can be very helpful in manipulating the server. | Since Windows is so popular, there are a lot of tools to help. |  |
| **Client Side** | Mac is more difficult to work with in general, with less support than Windows. It probably requires a bit more expertise to use. | Linux probably requires the most skilled developers because it is the least supported operating system of our 3. It does have a passionate following though. | Windows is the easiest to work with and provides more tools to it’s users. | You have to redo the UI to fit mobile devices as well as adjust the I/O to a touch based interface. This leads to more work and more costs. |
| **Development Tools** | TextMate  Apache Netbeans  Pycharm  Xcode | IDLE  PyDev  ItelliJ IDEA  Visual Studio | Visual Studio for C++ or python  .NET  Eclipse for Java  Pycharm for python | Android Studio  App Inventer for Android |

## 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 the Windows operating system. It is very widely used and has a lot of tools and APIs that can be used to simplify interacting with and requesting services from the operating system.
2. **Operating Systems Architectures**: Windows uses a one-to-one threading system, where for every thread the application generates, the kernal generates one. Processes can be created and managed with the win32 API, allowing for maximum portability. Windows allows processes to send messages to each other using a local procedure call facility. It schedules threads using a priority-based system where each thread is assigned a value between 1-31. The priority of applications running in the foreground is increased relative to applications that aren't on screen.
3. **Storage Management**: Windows organizes the disk into directories and files. It uses new technology file system to store and retrieve information on the disk. It recognizes a large list of file extensions, but any or no extension may be used. Support for long file names is included along with support for the old 8.3 length file names. Windows can now support very large file sizes and volumes, up to 8 PB. Special considerations must be given if a file this size is desired, as you have to format the disk to change the default cluster size. Clusters are a way of improving performance. Instead of recording each individual file block, a pointer is kept to clusters of blocks. Windows uses a log-based system to ensure the integrity of its file system. Changes are written to a log before being enacted. Corruption of the file system can occur for a variety of reasons. Changing a file involves a lot more than just changing the file’s data, because each file has associated metadata. Because windows uses caches to improve performance, if the system crashes while data is in cache waiting to be written to disk, that data will be lost. The log-system mitigates some of this risk. Windows has an API called Windows Storage Management Provider. We can use this tailor our storage needs to the situation.
4. **Memory Management**: Depending on the underlying hardware, different ways of managing memory may be employed. Most designs focus on problems that arise when you look at how the user views memory versus how the hardware uses memory. The user assumes that their data is stored in a contiguous location, but in reality, their data is stored in units called pages and is not contiguous. The processor generates logical addresses that must be translated to physical addresses. This takes special hardware to remain fast and efficient. Window’s employs slightly different architectures between 64-bit and 32-bit systems but for both memory is split 50-50 between the user and the kernal.
5. **Distributed Systems and Networks**: Distributed systems are when files or services are split up amoung multiple computers or servers. Windows has functions to help implement network functionaility into an app. They’re callled Windows Networking functions, or WNet. In networks resources are spread across the network. If they hold other resources they’re called containers, if they don’t, they’re objects. An object that can be accessed is called a sharepoint.
6. **Security**: Different security measures exist throughout the windows system. In its common internet file system is uses login authentication. To protect hard disk files, different access controls may be used. If there is a need to further protect sensitive information, some file systems support encryption. Calling the getVolumeInformation command and checking the bit flag FS\_FILE\_ENCRYPTION will verify if files can be encrypted. At the memory level, processes are restricted in the memory they can access. They usually can only access the memory that is assigned to them. Each process has an access token and unique ID.