Draw It or Lose It

CS 230 Project Software Design Template

Version 1.0

Table of Contents

CS 230 Project Software Design Template 1
- **Table of Contents** 2
- **Document Revision History** 2
- **Executive Summary** 3
- **Design Constraints** 3
- **System Architecture View** 3
- **Domain Model** 3
- **Evaluation** 5
- **Recommendations** 7

Document Revision History

Version	Date	Author	Comments
1.0	1/22/22	Zach Meisner	Filled out rubric

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

The Gaming Room wants to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It. The staff at The Gaming Room does not know how to set up the environment, and will need help streamlining development of a web-based version of the gaming app. This game needs to have the ability to have one or more teams involved, in addition to multiple players on each team. The game and team names must be unique to allow users to check whether a name is in use when choosing a team name. And only one instance of the game can exist in memory at any given time. The solution in this case would be to implement the singleton design pattern. This will allow the unique identifiers to be allocated properly and make it so only one instance of the game can run at any time, in addition to being able to assure that there will not be duplicates of teams, in addition to making it so each team will have unique identifiers based on their names. To allow multiple players on each team, we can implement the singleton pattern into a data structure such as a linked list or an array, which could allow turn-based play dependent on who's turn it is.

Design Constraints

Some design constraints due to the development of the game application in a web-based distributed environment involve the environment itself and how it operates. For example, given that the application currently operates as an Android app only, apart from seeing potential configuration issues during run time due to the potential differences in architecture and code, we must consider scalability and deployment. We must balance the number of users with the amount of backend application servers to avoid run time crashes of the game while deployed. Though multi-user platform is a great implementation within an application, this means we will have to configure every application to the same backend server, which may end up being messy, and difficult to implement in the case we want so many unique identifiers. In addition to this, in the case each instance of the game is not privatized to the specified teams, the run time may increase as the game grows, so the architecture of the players and teams must be cohesive enough to not overload the system in the case we have many different algorithms running at the same time against our servers, potentially overloading it with requests again, causing a potential crash.

System Architecture View

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

Encapsulation
Within the UML class diagram, one of the more important object-oriented features exhibited by this program is the singleton pattern. Encapsulation within the program is important due to the protection of data, in addition to the use of that data, and the state of each instance during runtime. In this case, we are protecting the instance itself and treating it as an individual so we can efficiently run a program, without any other instances interacting or interfering with one another.

Abstraction

When we have encapsulation, we also will have abstraction. The required functionality of this program can be considered complex on a base level of programming. The principles of OOP though foundational, need not be shared with the end user, and therefore we can justify the abstraction of certain aspects of the program, for example the singleton tester to assure that the functionality of the program is correct, in addition to the usage of the encapsulated object within the method of using the singleton pattern. This abstraction is what allows the program to function so efficiently, and fluidly given that in the case there is a problem with the test case, we can address and solve this problem without affecting the user.

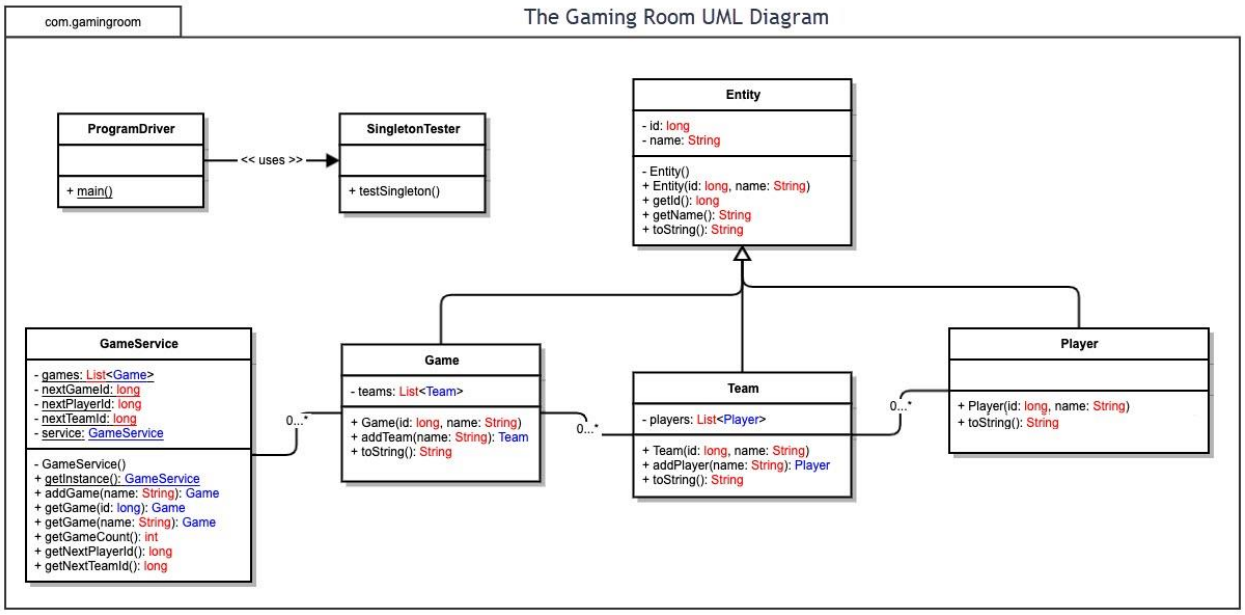
Inheritance

Inheritance is also utilized within the scope of our project due to the perceptive reality of the object we are creating within the specified instance of the game. The usage of inheritance is more of a baseline of logic that is foundational for understanding the actuality of presence within the game when the user connects, and then is "pushed" into a specific team given the allocation of the identifier from the singleton class. This final entity that is created uses all of the prior logic in order to manipulate the ideological state of the program.

Polymorphism

Lastly, we also use polymorphism as the combination of different itemized instances within the spectrum of our scope. In one way, we can say that the different teams, albeit they are considerably the same as other teams show a specific type of polymorphism. The differences between the objective teams based on the individuality of the identifiers is important to consider due to the difference of the makeup player wise. We need to be able to distinguish that although we have these teams, they have different identifiers, and therefore they are this object, stored in this place, which we will assign these players to. The transformation of this object in runtime is pertinent to the program as it is what allows the program to run efficiently.

On a finishing note, within the UML class diagram, we can see that Cardinality is expressed in terms of 0…* which is zero to many, again inheritance, which is represented between GameService, Game, Team, Player, and Entity.



Evaluation

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	+Good use of CLI +Easy Administration -Not great for large scale distribution -Not open source	+Open Source +Easy to use +Can customize distro specifically for development -Linux API can be difficult for non-Linux users - May take time to learn how to use it	+Most stable choice +Easy server development and administration -Windows is not open source -Can be annoying to setup environments if configuration on computer is not perfect.	+Mobile Web Server +Can develop for both websites and phones interchangeably -Not always able to connect to the internet -Not always the best for handling large amounts of traffic.
Client Side	+Great for Mac users +Easy to develop on Mac -Mac's can be expensive and are not known for gaming -Knowledge of Mac is necessary	+Linux is free +Linux is easy for people who know how to use it +Linux is used in some form in everything -Not generally the choice to play games on -Least used OS	+Dominates the gaming market +Most used Operating System +Great Graphics -User issues and configuration problems for a game are annoying and difficult to fix -Not as secure as Mac or Linux	+Different web browser +Don't have the amount of configuration you do on a computer -Dependent on brand of device -Different devices react differently and have to be translated for specific application stacks.

Development Tools	+Great for Java, website design, and Python +Swift and Xcode are specifically made for Mac -C++ is difficult and is not easy to setup/Use due to compiler issues -May have to translate languages into Mac language for app development.	+Large variety of development tools +Can pretty much run anything SeaMonkey Quanta Sublime Text	+Great for app development +Microsoft offers a great number of tools to use and tutorials to follow -Not specifically made for usability when developing Visual Studio Eclipse C++, Java, Python, Website development	+Apple has new M1 chips +Specific Library for applications that has been proven to be easy and secure +Offers in-house tools for growing your application business wise. +Android has Xamarin and Kotlin

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 the best development platform hands down. The amount of people using windows, in addition to the availability of multiple tools in addition to differing virtual environments to test applications on is unbeatable.

2. **Operating Systems Architectures**: Windows allows users to develop and set up the environment specifically the way they want it with an interactive GUI that is easy to use and is well known. In addition to having great user technical support, there is a massive amount of documentation online to help solve any issues that occur. Again, we also can use the OS in order to split up the computer resources easily to run dual operating systems if needed for test cases of applications on a different OS like Mac or Linux.

3. **Storage Management**: There are many different options to use, some of the more popular in the present day tend to be Dropbox and OneDrive. We also can use company coded databases, in addition to SQL.

4. **Memory Management**: Windows allows you to specify the exact amount of memory that you use for applications within a designated virtual instance during runtime. The ability to allocate different types of memory with partitions of your computer is exponential within the development of applications.

5. **Distributed Systems and Networks**: Cloud technology allows us to run servers with company databases to allow users to call the application on whatever platform they maybe using. The ability to configure the type of requests and respond with different instances of the game with multiple types of virtual instances dependent on the operating platform is unbeatable, as they communicate efficiently and easily, especially with company infrastructure and local networks.

6. **Security**: Security will always be an ongoing issue for whatever type of platform we use. Using windows, we can create a personalized database, and infrastructure to implement firewalls, and encryption. The ability for the company to be able to maneuver and respond as any type of hacker is incredibly important, and windows offers this versatility, in addition to teams at Microsoft that can offer support as well.