Draw it or Lose it!
CS 230 Project Software Design Template
Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	07/20/25	Zack Crostreet	Added an Entity class and updated the other classes to inherit from the Entity class.

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 has requested a game service architecture for their game "Draw it or Lose it" that not only supports unique game and team names but also prevents duplicate instances. This was solved using java while utilizing key software design patterns. This ensures that there is only one instance of the game in memory, there are unique games, teams and players, utilizing inheritance from the Entity class, and scalable architecture for future expansion. This is done by using a model where the GameService class manages Game instances which have teams that contain players, all of them inheriting common attributes through the Entity class.

Requirements

Business requirements:

- Unique game and team names that avoids duplicates
- Only one instance of the game in memory
- Support for multiple concurrent game instances
- Scalable architecture
- User security to protect information

Technical Requirements:

- Implement a singleton pattern for GameService
- Iterator for name validation
- Entity class for other classes to inherit common attributes
- Refactor Game, Team, and Player to inherit from Entity
- Online distribution environments
- Ensure unique names

Design Constraints

Some design constraints for this:

- Stateless operations to ensure requests are handled independently
- Unique game, team and players for all instances
- Scalability to accommodate growth in data
- Handle network conditions
- Support multiple OS platforms
- Support multiple sessions without data corruption

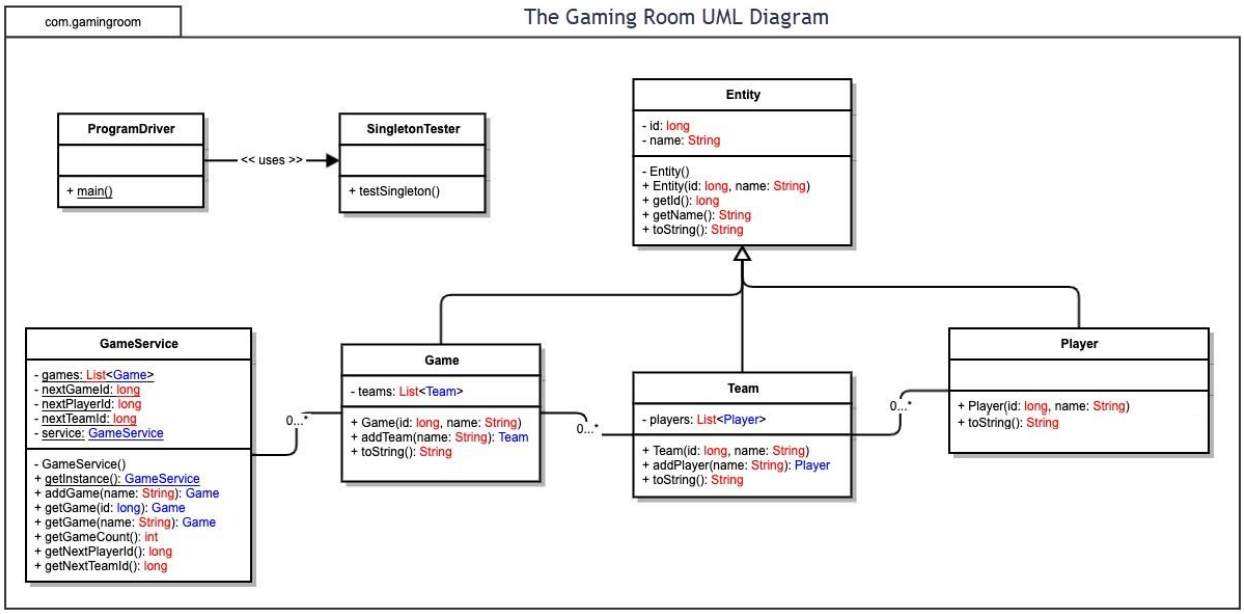
These constraints will ensure that the ID is centralized, thread-safe implementation, database-backed persistence for scalability, API communication, and cache distribution.

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

The domain model used elements of OOP principles which include Encapsulation, Inheritance and Polymorphism on a singleton model to ensure that every instance is unique. All the classes inherit from the Entity class such as Game, Team and Player inheriting key attributes from the Entity class. Demonstrated OOP principles to ensure private fields in public accessors, details hidden in the class, common attributes shared between classes and a consistent interface for all entities which each class having a clear focused purpose. Implementing these principles ensures that code is re-used, lifecycle management, single instance in GameService and the Driver uses a SingletonTester.



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	Exceeds in support for development but is limited due to the high cost. It is based on a Unix architecture to provide stability.	Cost-effective, open source, stable, great performance and stable. Excels in Java deployment.	Wide customer base which promotes tooling support but, it is much more expensive and resource heavy than Linux.	Due to the security risks and the limited resources, the mobile platforms are not suitable for server-side deployment.
Client Side	Requires Xcode/Swift which would cost more. Java is ideal as it excels in cross-platform support.	Java is supportive of cross-platform clients and web clients work the same across all platforms.	Large amount of installs means more user support across networks. Supports Java .NET and native languages like C#.	Specific development tools are needed such as IOS Swift and Android Java. Cost-effectiveness is very important on mobile platforms.
Development Tools	Java, Swift and VS code, excellent support for java.	Full Java support along with the ability to deploy Kubernetes.	Visual Studio, and Intellij java support through several supported IDEs.	Android Studio, Xcode with cross platform support from React Native and Flutter. Javascript and HTML5 frameworks.

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**: The clear top choice for this type of situation would be to adapt to the Linux operating system as it is more scalable, secure and cost effective.

2. **Operating Systems Architectures**: Linux for stability to keep contained in a Kubernetes or Docker framework for scalability, along with Java support.

3. **Storage Management**: Redis for caching and PostgreSQL for uniqueness in validation and ACID compliance for critical data.

4. **Memory Management**: Optimize object allocation for unique instances to cut down on garbage collection times while automatically managing memory through implementation of generational garbage collection with JVM.

5. **Distributed Systems and Networks**: Handle network dependencies cutting down on redundant network paths, lowering times of outages, ability to have a local cache for offline use or outages and queues for messages to reduce overloading the network. This can be done by implementing the RESTful API because it offers WebSocket for real-time updates, circuit breakers, edge caching static objects, exponential backoff and routine self-health checks.

6. **Security**: Since Linux is being used, adapting AppAmor along with container isolation, regular patching, and firewalls will ensure better platform security. To protect the data, we would adapt HMAC for data integrity, field level encryption for Pll and following compliance with AES-256 standards. Finally, for network security it is recommended to adapt TLS 1.3 for communications and authentication with OAuth 2.0 with JWT tokens. This will ensure that a layered onion-like approach is taken to ensure a thicker wall of security.