PART 1:

**Executive Summary:**

The Gaming Room aims to develop a web-based distributed game application that allows multiple users to participate in various gaming activities concurrently. The software design problem revolves around creating an efficient and scalable architecture that supports the dynamic nature of online gaming while ensuring a seamless user experience. To address this challenge, our solution focuses on implementing design patterns and adhering to best practices in software engineering.

**Design Constraints:**

Developing the game application in a web-based distributed environment presents several design constraints that must be considered:

1. **Scalability:** The application must be capable of handling a large number of concurrent users without sacrificing performance. This requires designing a scalable architecture that can dynamically allocate resources based on demand.
2. **Reliability:** In a distributed environment, network failures and server crashes are inevitable. The application must be designed to handle these failures gracefully, ensuring minimal disruption to the user experience.
3. **Security:** Online gaming platforms are prime targets for cyberattacks. The application must implement robust security measures to protect user data and prevent unauthorized access.
4. **Cross-platform Compatibility:** The application should be accessible from various devices and operating systems, including desktops, laptops, and mobile devices. This requires developing a responsive user interface that adapts to different screen sizes and resolutions.
5. **Data Consistency:** In a distributed environment, maintaining data consistency across multiple servers can be challenging. The application must implement mechanisms to synchronize data between servers and ensure consistency across all instances.

**Domain Model and Object-Oriented Principles:**

The provided UML diagram represents the Domain Model of the game application, illustrating the relationships between different entities. The key classes include **Game**, **GameService**, **Player**, **Team**, and **Entity**.

* **Game:** Represents a gaming session with associated players and teams. Demonstrates the principle of encapsulation by hiding the internal state (ID and name) and providing access through getter methods.
* **GameService:** Acts as a singleton service for managing game instances. Demonstrates the singleton design pattern, ensuring that only one instance of the **GameService** class exists in memory at any time.
* **Player:** Represents a player participating in a game session. Demonstrates the principle of abstraction by encapsulating player-specific data (ID and name) and providing methods to access this information.
* **Team:** Represents a group of players united for a common goal. Demonstrates the principle of composition by aggregating player objects within a team instance.
* **Entity:** Represents a generic entity with an ID and name. Serves as a base class for other domain entities, facilitating code reuse and maintaining a consistent interface across different entity types.

Overall, the UML diagram demonstrates the application of object-oriented principles such as encapsulation, abstraction, composition, and inheritance, which are essential for designing scalable and maintainable software systems.