

<Name-of-Software-Application>

# **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 | <07/22/23> | <Francis> | <Draft> |
| 1.1 | <08/11/23 | <Francis> | Checked for final, fixed errors |

## [Executive Summary](#_sbfa50wo7nsh)

The ProgramDriver class serves as the main entry point for the program and contains a single method named "main()". It utilizes the SingletonTester class by invoking the "testSingleton()" method through a <<uses>> relationship.

The Entity class represents a generic entity and features attributes such as "id" of type long and "name" of type String. It provides constructors, accessor methods, and a "toString()" method for converting the object to a string representation.

The GameService class manages a collection of games and serves as a singleton, ensuring only one instance exists. It maintains attributes such as "games," "nextGameId," "nextPlayerId," "nextTeamId," and "service." The class offers methods for adding games, retrieving games by their ID or name, obtaining the game count, and managing player and team IDs.

The Game class represents a specific game and maintains a list of teams. It includes a constructor for initializing the game with an ID and name, as well as a method for adding teams and generating a string representation.

The Team class represents a team within a game and maintains a list of players. It provides a constructor for creating teams with an ID and name, along with a method for adding players and generating a string representation.

The Player class represents an individual player and includes a constructor for initializing players with an ID and name. It also provides a method for converting the player object to a string.

The diagram illustrates the relationships between classes. The GameService class is associated with the Game class, denoted by "0...\*" cardinality, indicating that a game can be linked to multiple instances of GameService. Similarly, the Game class is connected to the Team class, and the Team class is linked to the Player class.

## Requirements

1. Game Management: The program should provide functionality to manage games, including adding games, retrieving games by ID or name, and counting the total number of games.

2. Team Management: The program should allow the management of teams within each game. This includes adding teams to a game and associating players with each team.

3. Player Management: The program should support the addition and management of individual players. Players should be associated with teams within a game.

4. Singleton Design Pattern: The GameService class needs to be implemented as a singleton to ensure only one instance of the class exists throughout the program.

5. Entity Management: The Entity class represents a generic entity and serves as a base class for other classes. It should support the creation of entities with an ID and name, as well as provide accessor and string representation methods.

6. Class Interactions: The program should enable the interactions between classes as depicted in the diagram. Specifically, the ProgramDriver class should utilize the SingletonTester class through the "<<uses>>" relationship. The GameService class should be connected to the Game class, the Game class to the Team class, and the Team class to the Player class, as indicated by the solid lines and "0...\*" multiplicity notation.

7. Data Structures: The program needs to incorporate appropriate data structures to manage lists of games, teams, and players. These data structures should allow efficient retrieval and manipulation of game-related information.

8. User Interface: Although not explicitly mentioned in the UML diagram, a user interface may be required to interact with the program, providing options for adding games, teams, and players, as well as displaying relevant information.

## [Design Constraints](#_2et92p0)

1. Programming Language: The program must be implemented using a specific programming language or languages, as dictated by project requirements or organizational guidelines.

2. Platform Compatibility: The program should be designed to run on specific platforms or operating systems, considering any limitations or dependencies associated with the chosen platforms.

3. Performance: The program should be designed to perform efficiently and meet any performance requirements or constraints specified by the project. This includes optimizing data structures, algorithms, and resource utilization.

4. Scalability: The program should be designed to handle a growing amount of data, users, and system load. It should be scalable and capable of accommodating increased demands without significant degradation in performance or functionality.

5. Security: The program should adhere to security best practices and address potential vulnerabilities. This may include implementing secure authentication, encryption of sensitive data, and protection against common security threats.

6. Data Persistence: If required, the program should support data persistence by integrating with a database or other storage mechanisms. The choice of database technology and implementation details should align with project requirements and constraints.

7. Compliance: The program should comply with relevant legal, regulatory, or industry-specific standards and guidelines, such as data protection regulations or industry-specific security protocols.

8. Usability: The program should be designed with a user-centric approach, considering the needs and abilities of the intended users. The user interface should be intuitive, accessible, and user-friendly.

9. Maintainability: The program should be designed and structured in a way that facilitates ease of maintenance and future enhancements. This may involve adhering to coding standards, utilizing modular design principles, and providing appropriate documentation.

10. Integration: If the program needs to interact with other systems or components, it should be designed to accommodate integration requirements. This may involve defining interfaces, protocols, or data formats for seamless communication with external systems.

11. Time and Budget Constraints: The design and development of the program should consider any time or budget constraints specified by the project. It should be feasible to complete the project within the given time frame and budget limitations.

## [System Architecture View](#_ilbxbyevv6b6)

1. Layered Architecture: The program can be structured using a layered architecture approach, where different layers handle specific responsibilities and communicate with each other through well-defined interfaces. The following layers can be identified:

a. Presentation Layer: This layer handles the user interface components and user interaction. It can include forms, screens, and input/output handling.

b. Application Layer: The application layer contains the business logic of the program. It coordinates the interactions between the presentation layer and the domain layer. It handles tasks such as game management, team management, and player management.

c. Domain Layer: The domain layer represents the core domain of the program. It includes the classes identified in the UML Class Diagram, such as Game, Team, Player, and Entity. The domain layer encapsulates the behavior and data associated with the game management system.

d. Persistence Layer: If data persistence is required, a persistence layer can be introduced. It handles the storage and retrieval of data from a database or other storage mechanisms. It includes components responsible for data access and persistence operations.

2. Singleton Pattern: The GameService class should be implemented as a singleton, ensuring that only one instance of the class exists throughout the program. This can be achieved by using a static method or initializing the instance at program startup.

3. Class Interactions: The interactions between classes, as depicted by the solid lines in the UML Class Diagram, should be realized through appropriate method calls and object references. For example, the GameService class can maintain a list of Game objects, and the Game class can manage a list of Team objects. These relationships should be established and managed within the appropriate layers.

4. Data Structures: The program should utilize suitable data structures for efficient management of game-related information. For example, lists or collections can be used to store instances of Game, Team, and Player classes.

5. Separation of Concerns: The architecture should aim to separate different concerns, ensuring that each component or layer is responsible for a specific set of functionalities. This separation allows for better maintainability, testability, and reusability of the system.

6. Modular Design: The system should be designed with modularity in mind, allowing for individual components or layers to be developed, tested, and maintained independently. This modular approach facilitates easier collaboration among development teams and promotes code reusability.

7. Integration Points: If the program needs to integrate with external systems or services, appropriate integration points should be identified and implemented. This may involve defining APIs, message formats, or communication protocols to enable seamless data exchange.

## [Domain Model](#_8h2ehzxfam4o)

1. The ProgramDriver class serves as the main entry point for the program.
2. The SingletonTester class is utilized by the ProgramDriver class and performs singleton testing.
3. The Entity class represents a generic entity with an ID and name.
4. The GameService class manages the games, teams, and players within the game management system. It maintains a list of Game objects and provides methods for adding games, retrieving games, and managing IDs for players and teams.
5. The Game class represents a specific game, which includes a list of Team objects. It provides a method for adding teams to the game.
6. The Team class represents a team within a game and contains a list of Player objects. It provides a method for adding players to the team.
7. The Player class represents an individual player in a team.

"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.

There is not enough room for a paragraph in the cell will be written below. Sorry if misunderstood the instructions

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | <Evaluate Mac for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Linux for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Windows for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Mobile Devices for their characteristics, advantages, and weaknesses for hosting a web-based software application.> |
| **Client Side** | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mac.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Linux.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Windows.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mobile Devices.> |
| **Development Tools** | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mac.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Linux.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Windows.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mobile Devices.> |

**Server Side:**

Mac:

1. Characteristics: macOS provides a Unix-based environment with robust security, stability, and seamless integration with other Apple products.
2. Advantages: It offers excellent development tools (such as Xcode) and is well-suited for server-side web application hosting due to its reliability and scalability.
3. Weaknesses: Limited market share compared to Windows may impact availability of certain software and support resources.

Linux:

1. Characteristics: Linux is an open-source operating system known for its stability, flexibility, and customization options.
2. Advantages: It is highly favored for server-side web application hosting due to its robust security, extensive software ecosystem, and cost-effectiveness.
3. Weaknesses: Advanced technical skills may be required for setup and maintenance, and hardware compatibility can be a potential challenge.

Windows:

1. Characteristics: Windows is a widely-used commercial operating system known for its user-friendly interface and extensive software support.
2. Advantages: It offers a familiar development environment, compatibility with various programming languages and frameworks, and easy integration with other Windows-based technologies.
3. Weaknesses: Historically, Windows may have higher licensing costs and is perceived to be less stable than Unix-based systems.

Mobile Devices:

1. Characteristics: Mobile devices (iOS and Android) provide portable and accessible platforms with extensive app ecosystems.
2. Advantages: Mobile devices can serve as servers for lightweight web applications, leveraging their mobility, ease of deployment, and ability to target specific user segments.
3. Weaknesses: Limited resources, such as processing power and memory, may limit the scalability and performance of server-side web applications on mobile devices.

**Client Side:**

Mac:

1. Characteristics: macOS provides a user-friendly and visually appealing interface with seamless integration across Apple devices.
2. Advantages: It offers excellent development tools (such as Xcode) and a smooth user experience for client-side web applications with access to native macOS features and technologies.
3. Weaknesses: Limited market share compared to Windows may result in compatibility issues with certain software and potential user reach.

Linux:

1. Characteristics: Linux is an open-source operating system known for its stability, security, and customizable nature.
2. Advantages: It offers flexibility, a wide range of compatible software, and excellent performance for client-side web applications, particularly for developers who prefer open-source technologies.
3. Weaknesses: User interface inconsistencies across different Linux distributions and potential challenges with proprietary software compatibility.

Windows:

1. Characteristics: Windows is a widely-used commercial operating system known for its user-friendly interface and extensive software support.
2. Advantages: It offers a familiar environment for client-side web application development, extensive compatibility with popular software and browsers, and a vast user base.
3. Weaknesses: Possible security vulnerabilities and occasional inconsistencies across different versions of Windows.

Mobile Devices:

1. Characteristics: Mobile devices (iOS and Android) provide portable and touchscreen-based platforms with extensive app ecosystems.
2. Advantages: Mobile devices offer a personalized and immersive user experience for client-side web applications, leveraging features like touch interactions, push notifications, and access to device hardware.
3. Weaknesses: Limited screen size and potential performance constraints may require careful design considerations for responsive and optimized client-side web applications on mobile devices.

**Development Tools:**

Mac:

1. Xcode: Xcode is the integrated development environment (IDE) for macOS and iOS app development. It provides a comprehensive set of tools, including code editing, debugging, interface design, and performance analysis.

Linux:

1. GNU Compiler Collection (GCC): GCC is a widely used compiler collection for various programming languages on Linux. It includes compilers for languages like C, C++, and Fortran, along with related tools for building and debugging applications.
2. Visual Studio Code: Visual Studio Code is a popular open-source code editor that runs on Linux, offering a lightweight and extensible development environment for multiple programming languages. It supports features like syntax highlighting, debugging, and source control integration.

Windows:

1. Visual Studio: Visual Studio is a powerful IDE developed by Microsoft for Windows application development. It supports multiple programming languages, including C#, C++, and JavaScript, and offers a wide range of tools for building, debugging, testing, and deploying software.
2. IntelliJ IDEA: IntelliJ IDEA is a popular Java IDE that also supports other programming languages like Kotlin and Python. It provides a rich set of features for code editing, debugging, and version control, making it suitable for Windows development.

Mobile Devices:

1. Android Studio: Android Studio is the official IDE for Android app development. It offers tools for designing user interfaces, writing code in Java, Kotlin, or C++, testing, and deploying Android applications on mobile devices.
2. Xcode: Xcode, as mentioned earlier, is the primary development tool for iOS app development. It includes a suite of tools for designing interfaces, coding in Swift or Objective-C, testing, and deploying iOS applications.

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

**Web Platform hosted on Debian Server. Using Phaser framework or Unity if the Liceance fee is not a problem:**

1. Compatibility: Web technologies are platform-agnostic and can run on various operating systems, including Windows, macOS, Linux, and mobile platforms like Android and iOS. This allows for seamless access and compatibility across different devices and operating environments.
2. Reach: Web applications can be accessed through web browsers, which are widely available and supported across different operating systems and devices. This ensures a broader user reach and eliminates the need for platform-specific installations.
3. Easy Deployment: With a web-based solution, you can deploy the game as a web application hosted on a server, making it accessible to users via a URL. This eliminates the need for users to install the game on their devices, simplifying the deployment process and reducing compatibility issues.
4. Cross-Platform Development: Web technologies like HTML, CSS, and JavaScript offer a consistent development experience across platforms. Frameworks like React, Angular, or Vue.js can be used to build responsive and interactive web applications that adapt to different screen sizes and device capabilities.
5. Flexibility: Web technologies allow for continuous updates and enhancements without requiring users to download and install updates manually. This facilitates iterative development and the ability to roll out new features and improvements seamlessly.

**Debian Server security:**

Security is never guaranteed but Linux is superior.

1. Secure Communication: Ensure that all communication between the client (web browser) and the server is encrypted using HTTPS (HTTP over SSL/TLS) to protect sensitive data in transit. This prevents unauthorized parties from intercepting or tampering with the data exchanged.
2. User Authentication: Implement a robust user authentication system to verify the identity of players. Use secure authentication mechanisms, such as password hashing, salting, and multi-factor authentication, to protect user accounts from unauthorized access.
3. Authorization and Access Control: Implement access controls to restrict user actions and ensure that players can only perform authorized actions within the game. Enforce proper authorization checks to prevent unauthorized access to sensitive game features or administration functions.
4. Input Validation: Validate and sanitize all user input to prevent common security vulnerabilities such as cross-site scripting (XSS) and SQL injection attacks. Apply input validation and output encoding techniques to ensure that user-supplied data does not pose a security risk.
5. Secure Game Logic: Design the game logic with security in mind. Implement measures to prevent cheating, tampering, or manipulation of game results. Encrypt sensitive game data, such as scores or game progress, to prevent unauthorized modifications.
6. Secure Session Management: Implement secure session management techniques to handle user sessions effectively. Use secure session tokens, enforce session timeouts, and protect against session fixation attacks to ensure that player sessions remain secure.
7. Secure Server Configuration: Ensure that the server hosting the game is properly configured and hardened. Regularly update and patch software components, use secure configurations, and follow industry best practices to mitigate security vulnerabilities.
8. Security Auditing and Testing: Regularly conduct security audits and vulnerability assessments to identify and address potential security weaknesses. Perform penetration testing to simulate real-world attacks and evaluate the resilience of the application against various security threats.
9. Data Protection and Privacy: Handle player data in accordance with data protection regulations and best practices. Implement measures to protect sensitive player information, such as encryption at rest, secure data storage, and privacy controls.
10. Regular Security Updates: Stay up-to-date with security updates and patches for all software components used in the game's infrastructure, including web servers, databases, frameworks, and third-party libraries.