



CS 230 Project One Milestone Guidelines and Rubric

Overview

Design patterns can speed up the development process. Many of the most common computer science problems have already been solved with well-established approaches. While code examples are often shown when describing a pattern, the code is not intended to simply be copied into your application. It is important to know what problem the pattern is designed to solve and if it fits your situation. When you find a pattern that does, you also get the added benefit of communication and familiarity. For example, simply stating that a class implements the singleton pattern lets other programmers instantly know how your class is built and what it is intended to do.

A common trap that beginning and intermediate programmers fall into is trying to use as many patterns as they can, thinking that these are the keys to making the best software possible. The best approach is to first understand the problem and then research it to determine if there is a pattern that fits well in the context of the application you are building.

In this assignment, you will demonstrate your ability to use design patterns to create efficient code. **It is important to understand these concepts and complete this assignment as it will directly inform future assignments.**

Prompt

UML Diagram

Review the [UML diagram](#) provided for a game software application. A text version is available: [Text Version for UML diagram Word Document](#). You may notice that it is incomplete - it is missing attributes and methods, and only includes a small portion of a complete game application. This is quite common! As a software developer, typically you will be given pieces of the puzzle or tasks to complete as part of a larger project and as a member of a larger team. It takes practice to focus on the information you are given to determine what you have and what steps you need to take to complete the task.

Specifically, the game application requires that only one instance of the game be able to exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of the game.

- **Complete the UML diagram to accurately represent a singleton pattern for the Game Service class.** You should replace the question marks in the UML diagram with appropriate static and instance attributes and/or methods to make the ordinary class into a singleton. You may create the diagram using Lucidchart or a tool of your choice, such as draw.io or Visio. A tutorial for creating UML diagrams is available: [Lucidchart Tutorial for Creating Class Diagrams PDF](#).

You may find pages 134–136 of the *Hands-On Design Patterns with Java* textbook and this [singleton pattern tutorial](#) helpful in using the singleton pattern.

Application

Revise the code to incorporate common design patterns to make your code more efficient. Given the starter code, complete the Game Service, Program Driver, and SingletonTester classes.

Begin by uploading [CS 230 Project One Game App.zip](#) as a new project in Eclipse. A tutorial is available for uploading files to Eclipse: [Uploading Files to Eclipse Desktop Version Tutorial PDF](#). Please note: You will not complete the Player and Team classes at this time.

To view the tasks you have been assigned, navigate in your Eclipse project to the **Window**, **Show View**, and **Tasks** view. You can double-click on any of the tasks (as seen in the [image](#)) to be directed to the line of code that needs to be completed where the tasks are located. A text version of this image is available: [FIXME Text Version Word Document](#).

Review the code for the Game Service class provided for you. Notice the static variables holding the next identifier to be assigned for game Id.

- **Use the singleton pattern to adapt an ordinary class definition**, so that only one instance of the GameService class exists in memory at any time.
- **Explain the purpose and characteristics of the singleton pattern** you are using in the context of this application. Include this brief explanation as comments in your code. The comments will be used to communicate your understanding of the pattern you implemented to your instructor.

This multi-user game application must have the ability to have multiple instances of the game running at once, each having players and teams. Often, individual instances of the game will prompt users to save the new game being played. To ensure there are no duplicate instances of the game at the point of saving, the application must check to ensure that the name chosen by the user is not already in use.

To meet this requirement, you will want to use an iterator pattern in the application. This will ensure that when a user starts a new game to play, the application checks every instance of the game already in play to determine if the new name chosen by the user is in use. Currently, most frameworks have iterator capabilities built in; it is rare to need to build an iterator from scratch. Therefore, we'll practice using one provided in Java.




- **In the Game Service class, use the iterator pattern to complete the coding for the addGame() and both getGame() methods.** Note: The getGameName() method is needed to allow a game instance to be retrieved using only the unique name.
 - **Explain the purpose and characteristics of the iterator pattern** you are using in the context of this application. Include this brief explanation as comments in your code. The comments will be used to communicate your understanding of the pattern you implemented to your instructor.
- Demonstrate industry standard best practices to enhance the readability of your code, including appropriate naming conventions and in-line comments that describe the functionality.
- Run and compile the code to ensure the application is functional and the requirements have been met.

What to Submit

Submit a zipped project folder containing the completed source code along with an image, or PDF, of the completed UML diagram.

Project One Milestone Rubric

Criteria	Proficient (100%)	Needs Improvement (55%)	Not Evident (0%)	Value
UML Diagram	Creates a UML diagram to accurately represent a singleton pattern	Shows progress toward proficiency, but with errors or omissions	Does not attempt criterion	10
Singleton Pattern	Uses the singleton pattern to adapt an ordinary class definition, so only one instance of the class exists in memory at any time	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include instances being directly instantiated, correctly using static modifier	Does not attempt criterion	17
Purpose of Singleton Pattern	Discusses the purpose and characteristics of the singleton pattern in the context of the application	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include aligning purpose of the design pattern to the context of the software application	Does not attempt criterion	17
Iterator Pattern	Uses the iterator pattern to efficiently navigate every instance in a list	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include using iterator methods	Does not attempt criterion	17
Purpose of Iterator Pattern	Discusses the purpose and characteristics of the iterator pattern in the context of the application	Shows progress toward proficiency, but with errors or omissions; areas for improvement may include aligning purpose of the design pattern to the context of the software application	Does not attempt criterion	17
Functionality	Develops functional code that meet software requirements	Shows progress toward proficiency, but with errors or omissions	Does not attempt criterion	17
Industry Standard Best Practices	Demonstrates industry standard best practices including in-line comments and appropriate naming conventions to enhance readability of code	Shows progress toward proficiency, but with errors or omissions	Does not attempt criterion	5
Total:				100%

 Listen	 Dictionary	 Translate
--	--	---