Name: Nowshin Sumaiya
ID: 21301276
Section: 08
Course: CS470 (quiz02)

Answer to the question no:01

a) Observer and Singleton design pattern(s) would be suitable to use in this scenario. Now will discuss about them below:

Observer design pattern : It is mentioned in the questions that multiple display devices (smartphones, tablets, desktop widgets) need to be notified in real-time whenever the weather data changes. That's why the observer design pattern is suitable. The observer pattern is used in situations when there is a one-to-many relationship between objects, such as when dependent objects need to be automatically notified when a subject (the object being observed) is modified. It gives a description of how the objects and the observer are connected. Broadcast-style communication is supported by it. Again, for the weather monitoring system, this design is perfect. As it will help us to send notifications whenever any changes in the weather. In this scenario ,all the display device is an observer, while the central weather data acts like a subject.

Singleton design pattern : It is mentioned in the questions that we also need a centralized manager that controls access to the weather data to ensure consistency across all displays.That's why the singleton design pattern is suitable . One of the most fundamental design patterns is the singleton pattern. It ensures that a class has a single instance and affords a universal point of access. Singleton design pattern saves memory because an object isn't created for every request. There is only one instance that is continually used. Moreover, the centralized manager needs to follow singleton pattern to guarantee that the weather data is updated and accessible through a single, reliable source. By doing this, inappropriate updates are avoided and consistent information is sent to every device.

b) Observer and Singleton design pattern(s) matches with the scenario provided in the question. Several methods need to be provided by the participants in the design pattern(s) in order for the mentioned functionality to be implemented.

Methods in observer Pattern:

- add_observer(Observer observer): This method adds a display device. Once it is added, the display device will be notified whenever the weather data changes.

- remove_observer(Observer observer) : This method removes a display device. Once it is removed the device would not get any notifications in future.

- notification(): When the weather data is updated. It sends the most up-to-date weather information to each and every registered observer. All this happens in a loop.

- display(): shows the current weather on the device screen.

- update(): whenever we get new information regarding weather ,the display devices update their user interfaces to present the updated weather information.

- current_state(): returns the current weather data. Observers can call this if they need the most recent weather information.

- extra_Weather_data(): This method can be called by a device to retrieve other information, such as humidity or wind speed, in addition to temperature. Fetch directly from the weather manager by method.

Methods in Singleton Pattern:

- obtain_Instance(): gives access to the weather data manager's single instance. The instance is created if it doesn't already exist. This function can be called by any part of the application to use the same weather manager.

- weather_data_update(data) : updates the weather data inside the singleton instance.

- obtain_weather_data(data): The current weather information is retrieved from the singleton using this method.

- Initialized(): This method verifies whether the singleton instance has already been generated or not . It is helpful in avoiding redundant re-initializations.

- reset(): Permits the singleton instance to be reset in case it's required for testing or unusual circumstances. This method is rarely used.

Answer to the question no:02

a) I would use adapter and observer design pattern(s) to solve this problem mentioned in the question.

Adapter design pattern : It is mentioned in the questions that, we have to integrate an old payment gateway with the modern e-commerce platform. That's why the adapter design pattern is suitable for solving the problem. On the other hand, a bridge between two mismatched interfaces is created by the adapter design pattern. It's frequently applied if one needs to integrate pre-existing classes with new ones without altering their source code. When an existing class's interface differs from the one we want, this pattern is especially helpful. Again, this design is ideal for creating a link between the current e-commerce platform and the old payment gateway. Through the use of an adaptor, the outdated payment gateway is enclosed and its API converted into a form that can be accessed by the latest system. By doing this, we can make sure that our platform can interact with the old gateway without modifying the platform's current code.

Observer design pattern : It is mentioned in the questions that, we need to notify multiple modules in the system (such as inventory, shipping, and order processing) whenever a payment is successfully completed. That's why the observer design pattern is suitable for solving the problem. Moreover, when there is a one-to-many relationship between objects, like when dependent objects must be automatically informed when a subject (the object being observed) is changed, the observer pattern is applied. It shows the relationship between the objects and the observer. It provides communication in the form of broadcasting . Again, Order processing, shipping, inventory, and other modules act as observers, while the payment system acts as the subject in the scenario . The payment system notifies these modules whenever a payment is finished, making sure they are updated appropriately.

b) Adapter and observer design pattern(s) matches with the scenario provided in the question. Now I will describe the key characteristics of this pattern(s) in terms of their intent, motivation and participants.

Adapter design pattern :

- Intent: The key goal of the adapter pattern is to make a possible combination of incompatible interfaces. It operates as a translator for two different systems. Assume that you have two friends, one of whom speaks only Bangla and the other just English. Despite the language barrier, you want them to talk to one other. You translate conversations between them, acting as an adaptor.

- Motivation: An outdated payment gateway needs to be integrated with a modern e-commerce platform. This is made possible by the adapter pattern that creates a connecting layer.

- Participants:
  - ★ Client: The modern e-commerce platform, which is in need of an API format for payments.
  - ★ Target: The adapter's interface, which the modern platform requires.
  - ★ Adaptee: Old payment gateway with outdated API.
  - ★ Adapter: The portion that converts the old API into the format that the platform requires.

Observer design pattern :

- Intent: The observer pattern makes sure that all observers are automatically informed and updated when an object (subject) modifies its state.

- Motivation: When a payment is successfully processed, we have to notify several system modules (including inventory, shipping, and order processing). By using the observer pattern, these modules are able to 'observe' the payment procedure and get updates.

- Participants:

  - ★ Subject: The payment method that manages and gives notifications to the observers.
  - ★ Concrete Subject: the authentic payment mechanism that sends out notifications.
  - ★ Observers: The order processing, shipping , inventory etc modules that must be informed when a payment has been received.
  - ★ Concrete Observers: Each module's particular implementations

Answer to the question no:03

a) I would use singleton design pattern for this logging system mentioned in the question.

Singleton design patterns ensure that a class has just a single instance. Also provide a global access point to that instance. Since an object isn't created for

every request, the singleton design method saves memory. There is only one instance that is continually used. On the other hand, Singleton pattern is frequently used in multi-threaded and database applications. It is used in thread pools, caching, logging, configuration settings, and other areas.

So, for the logging system, the Singleton Pattern works effectively. A single instance of the logger is generated and shared across the entire application. Means it makes sure that a single instance of the logger gets created, this will help us to avoid duplicate log entries. Also it provides the same logger instance to be accessed by all application parts, maintaining consistent behavior and reducing unnecessary overhead.

Adapter Pattern, observer Pattern is not suitable for the task mentioned in the question. A bridge between two mismatched interfaces is created by the adapter design pattern. It's frequently applied if one needs to integrate pre-existing classes with new ones without altering their source code. So, it does not solve the need for a single instance across the application. On the other hand,the observer pattern is used in situations when there is a one-to-many relationship between objects, such as when dependent objects need to be automatically notified when a subject (the object being observed) is modified. which isn't the main drawback with a logging system.

Lastly, based on the detailed discussion above, it can be said that the Singleton pattern is an ideal choice for implementing a logging system mentioned in the question.

b) We will use the Singleton design pattern for implementing a logging system mentioned in the question. Now I will describe the key characteristics of the pattern in terms of their intent, motivation and participants.

- Intent: To ensure a class has only one instance, and to provide a global point of access to that instance.

- Motivation: The singleton pattern ensures that only one instance is used, preventing conflicts, redundant data, or inefficient resource utilization in scenarios where resource sharing, consistency, or centralized control are necessary

- Participants:
  - ★ Singleton Class (Logger): The class that ensures that just one instance is created and allows access to it by implementing the singleton pattern.
    Responsible for managing the single instance.

  - ★ Client: Any part of the application that makes use of the singleton instance (in this scenario, the various application modules that require logging features).