MAKERERE UNIVERSITY

College Of Computing & It

Software Design Patterns

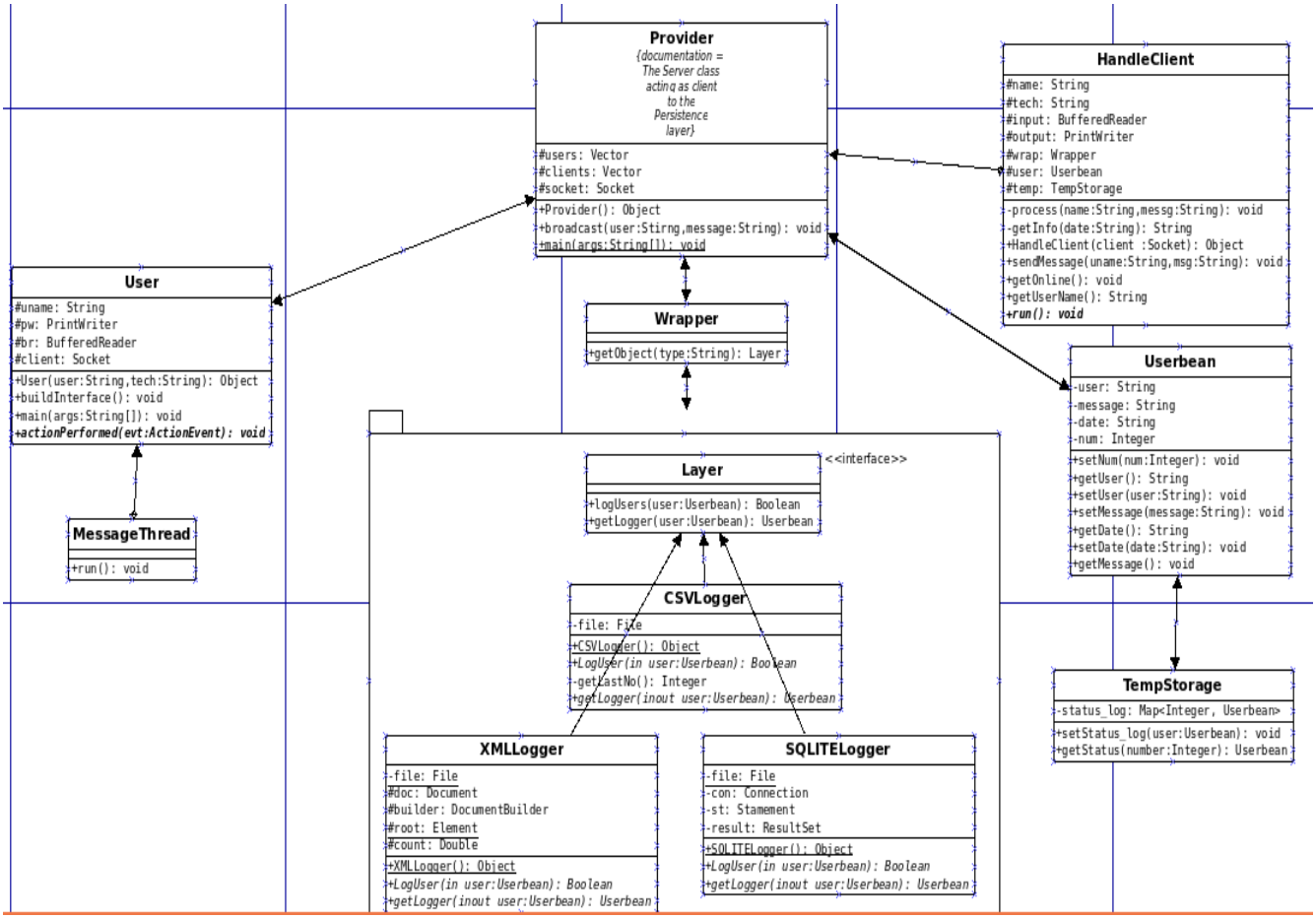
Course Work dated:

Names	Registration Number	Sign
OLUPOT DOUGLAS	10/U/9424/PS	
KEMIGISHA FLORENCE		
ACLEO KANSIIME		
SEBBUDE ABUBAKER		
HABUMUGISHA DAVID	10/U/9224/PS	

Table of Contents

Class diagram of the Application

The class diagram for the Universal Discussion application (UDA) representing the Actual application.



Patterns used

This application was implemented following the patterns listed below:

1. Memento
2. Mediator
3. Factory

These patterns are further discussed below based on how they were implemented in the application respectively.

Memento

This pattern is represented by the three classes namely TempStorage as the CareTaker, Userbean as the Memento class and Server as the Originator. The Provider (client to the persistent layer) instantiates objects from the Userbean and TempStorage classes respectively and sets values received from the User class (as client in the application) wrapping the user, message, date and num as a single object and instantiates the TempStorage class to temporarily store the object.

Mediator

This pattern is represented by the Userbean class which acts as an intermediary for the Provider and TempStorage classes. It is used by the Provider class to create different user-message threads (Userbeans) and pass their information to the TempStorage and persistence storage classes (Sqlite, Csv, XML) among others. The Provider class instantiates the client's chosen storage technology and creates an object of it. After this is achieved successfully, the object created is then used to write the message to the persistent layer chosen. However, all messages are stored in an Sqlite database to keep track of all records given the fact that the user has the capability to pick any technology and keep changing, we need a persistent storage to do referencing. Now all the users who connect are recorded into a List that stores their details from username, time of connection and a simple message referencing their connection.

Factory

This pattern is explained with the help of Three classes and one interface represented in the package where all the (XMLLogger, CSVLogger and SQLITELogger) classes implement that interface called Layer through it the Provider can create objects of the classes through the Wrapper class that returns different objects depending on what the Provider class chooses. It simply allows the client (Provider) to create objects of classes but allow the subclasses (XMLLogger, CSVLogger and SQLITELogger) to determine how the objects behave. In the class Diagram, the Layer is the interface and it's implemented by the XMLLogger, CSVLogger and SQLITELogger subclasses that do the actual saving of the information passed through the methods implemented.

Advantages of the patterns used

Advantages of the Factory pattern/Virtual Constructor.

- This pattern greatly assisted in creating loosely coupled classes since it allows any new class joining to just simply implement the interface in it's own way and not affect the existing classes.
- It is more flexible because it allows subclasses to have their own understanding of the objects instantiated through the interface.
- It also gives a client class the power to determine which class it wants to instantiate especially when the client class doesn't know the class objects it could use.

Advantages of Mediator pattern.

- This pattern helped to decouple classes and easily transfer known objects between classes like the Provider and Storage classes (XMLLogger, CSVLogger and SQLITELogger).
- It also allows centralized control of data being sent and received. This makes it easy to intercept and manipulate the data as an object.
- The mediator encapsulate the logic of mediation between the colleagues. From this reason it' more easier to understand this logic since it is kept in only one class.
- Decoupled Colleagues- The colleague classes are totally decoupled. Adding a new colleague class is very easy due to this decoupling level.
- Simplified object protocols- The colleague objects need to communicate only with the mediator objects. Practically the mediator pattern reduce the required communication channels(protocols) from many to many to one to many and many to one.
- Limits Subclassing- Because the entire communication logic is encapsulated by the mediator class, when this logic need to be extended only the mediator class need to be extended.
- It is particularly best for message based systems and works just fine.

Advantages of the Memento pattern

- This pattern allowed us to easily preserve encapsulation boundaries of our classes. Particularly the Provider (Originator) and TempStorage (CareTaker) by providing a go between mediator called the Userbean that allows easy transfer of the object without changing the content of the classes directly.
- The Server class particularly was made a lot easier to write because it only had to instantiate an object of type MessageBean and pass data, call the TemporalStorage class object and save the MessageBean object and retrieve it for later use. It was that simple.
- Memento protects encapsulation and avoids exposing originator's internal state and

implementation. It also simplifies originator code such that the originator does not need to keep track of its previous state since this is the responsibility of the CareTaker.

Disadvantages of patterns used.

Disadvantages of Memento pattern

- This pattern is limited to simple applications and cannot handle complex application because it's scope is limited.

Consequences:

- Using this pattern can be expensive depending on the amount of state information that has to be stored inside the memento object. In addition the caretaker must contain additional logic to be able to manage mementos.

Disadvantages of the Factory Pattern

- At times the client class may not know all the implementations of the interfaces and can miss out on some of the functionalities implemented.
- It brings unnecessary complexity in application.

Disadvantages of the Mediator

- Complexity- in practice the mediators tends to become more complex and complex. A good practice is to take care to make the mediator classes responsible only for the communication part. For example when implementing different screens the the screen class should not contain code which is not a part of the screen operations. It should be put in some other classes.
- This pattern allows creation of many objects in memory and can greatly affect performance of the application. This can affect the JVM particularly with Operating systems that limit memory usage.

Conclusion

All in all, despite these patterns having serious disadvantages, they seem to have been the perfect ones for this kind of application and are well suited.

References

[1] https://www.tutorialspoint.com