JAVA INSTITUTE for Advanced Technology

INNOVATIONS FOR THE KNOWLEDGE SOCIETY

ISO 9001:2008 Certified Institute

Java Institute for Advanced Technology

Department of Examinations

SQA Approved Centre
Scottish Qualifications Authority

scqf
scottish credit and qualifications framework

PCJT
The Software Engineer

BIRMINGHAM CITY University

COURSE(S) – (LEADING TO)	BIRMINGHAM CITY BSC (HONS) SE - TOP-UP
SUBMISSION DATE	
UNIT NAME	OBJECT ORIENTED DESIGN PATTERN II
UNIT ID	JIAT/DP II
EXAMINATION ID	JIAT/DP II/AS/12
BRANCH	JAVA INSTITUTE, COLOMBO

NAME	: M.R.P.N.THARUKSHA RAJAPAKSHA (BLOCK CAPITALS)
BCU STUDENT ID	: 22178965
NIC NO	: 200019401866
SCN NO	: 207977608

Java Institute for Advanced Technology
Sri Lanka

Acknowledgments

First and foremost, I would like to express my deep and sincere gratitude to my lecturer and advisor for the object-oriented design patterns II subject, Ms. Narmadha Harischandra for the continuous support, motivation, and knowledge he has given to me to complete this project.

I am incredibly grateful to my parent for their love, care, and sacrifices they made to build a better future for me. And the encouragement they have given me to complete this project.

I would like to thank all of my friends for the support they have given and for the sleepless nights we were working together before deadlines.

I thank the management of Java Institute for their support in doing this project.

Finally, my thanks go to all the people who have helped me to complete this project directly or indirectly.

Table of Contents

1. What is Compound Pattern?

There are three types of design patterns such as structural, creational, and behavioral design patterns.

Every software design or solution does not use a single design pattern. Patterns, however, do not work in isolation in software implementation. Actually, patterns are frequently used in tandem and in combination to achieve a specific design solution. A compound pattern, according to GoF, "A compound pattern combines two or more patterns into a solution that solves a recurring or general problem." A compound pattern is a general solution to a problem, not a collection of patterns that work together.

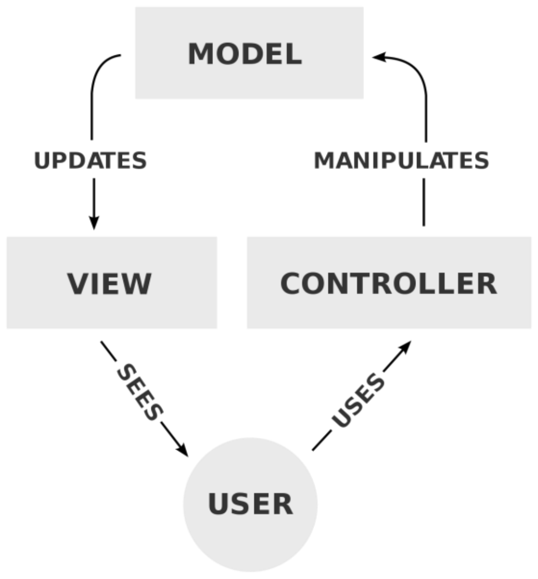
The best example of a compound pattern is the Model-View-Controller pattern, which has been used in many design solutions over the years.

2. Structure of MVC architecture.

The Model-View-Controller (MVC) pattern is a software pattern used to create user interfaces and an architecture that is simple to maintain and modify. The MVC pattern essentially divides the application into three essential parts such as model, view, and controller. These three parts are interconnected and aid in the separation of how information is represented from how information is presented.

The MVC pattern works as follows. The model represents the data and business logic (how information is stored and queried), the view is simply the representation (how it is presented), and the controller is the glue between the two, directing the model and view to behave in a specific way based on what the user requires. The MVC pattern is defined by the fact that models can be worked with independently. Surprisingly, the model is dependent on the view but not the other way around. This is primarily due to the user's concern about the data.

3. Patterns in MVC architecture.



- Model (Observer Pattern): This defines a class for storing and manipulating data.
- View (Composite Pattern): This declares a class that will be used to create user interfaces and data displays.
- Controller (Strategy Pattern): This declares a class that connects the model and the view.

Model (Application Knowledge)

- Because it is independent of the view and controller the model is the foundation of an application. The model, in turn, influences the view and controller.
- Because the model is completely independent, developers working on this piece can concentrate on maintenance without worrying about the most recent view changes.
- The model also provides the data that the client has requested. In most applications, the model is represented by database tables that store and return data. The model has a state and methods for changing it, but it is unaware of how the data will be viewed by the client.

- It is critical that the model remains consistent across multiple operations. Otherwise, the client may become corrupted or display stale data both of which are highly undesirable.

View (The Appearance)

- The view is a representation of data on the client's interface.
- The view can be created independently, but it should not include any complex logic. The controller or model should still contain the logic.
- Views should avoid interacting directly with databases and instead rely on models to obtain the necessary data.
- In today's world, views must be adaptable to multiple platforms, including desktop, mobile, tablets, and various screen sizes.

Controller (The Glue)

- The controller, as the name implies controls the user's interaction with the interface.
- Controllers also send data to views, which display the information on the interface for the user.
- When the user clicks on certain elements of the interface the controller makes a call to the model which then creates, updates, or deletes the data based on the interaction (button click or touch).
- The Controller should not make database calls or participate in data presentation.
- The controller should be as thin as possible and serve as the glue between the model and the view.

The benefits of MVC architecture

- The application becomes more understandable when MVC is used.
- The developers can work on all three layers (Model, View, and Controller) at the same time.
- MVC has a scalability feature, which aids in the growth of the application.
- A model can be reused by multiple views, allowing for code reusability.

- Because MVC maintains each layer separately, we do not have to deal with large amounts of code.
- It is now easier to extend and test applications.
- Because there is less reliance, the components are simple to maintain.

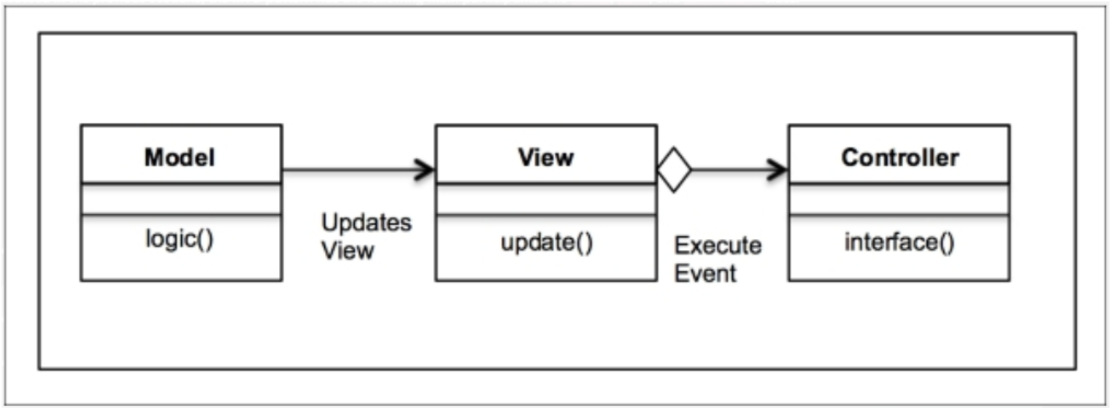
Usage of MVC design pattern

- To change the representation on the user interface, multiple controllers can be used in conjunction with multiple views.
- Again, because they can work independently of each other, the model can be changed without affecting the view.
- When it is necessary to change the presentation without affecting the business logic can use it.

The MVC pattern main goals:

- The class and implementation are simple to maintain.
- The ability to change how data is stored and displayed. Both are self-contained and as a result adaptable.
- Keeping the data and its presentation separate.

UML Diagram of MVC Architecture



4. Difference between compound pattern and composite pattern.

A composite pattern is something that is made up of interconnected parts. A service composition, for example, could legitimately be referred to as a composite of services because the individual parts must be designed into an aggregate in order to function as a whole.

A compound pattern, on the other hand, is simply the result of combining a particular set of things. A chemical compound is made up of several ingredients that when combined create something new.

5. Patterns that can be used together.

1. **Adapter Pattern**
    - Prototype Pattern: This pattern wraps another object but does not change its interface.
    - Decorator Pattern: This pattern adds functionality to an object without changing its interface and also allows for recursive composition, which adapters do not allow.

2. **Bridge Pattern**
    - Adapter Pattern: The adapter pattern appears similar but serves a different purpose. Unlike bridge pattern, which decouples abstractions and implementations, it aims to make unrelated classes work together.
    - Factory Pattern: This pattern can be used to build and configure bridges.

3. **Chain of Responsibility Pattern**
    - Composite Pattern: Chain of responsibility pattern can be used in conjunction with Composite pattern, where the parent of a component can act as the successor.

4. **Command Pattern**
    - Memento Pattern: This pattern can be used to keep the state required to undo the effect of a command.

- Composite Pattern: commands can be implemented using this pattern.
- Prototype Pattern: A command can sometimes be copied before it is added to the command history. The original command can act as a prototype that is copied in this case.

5. Composite Pattern

- Flyweight Pattern: With this pattern, you can share components.
- Iterator Pattern: This pattern is useful for traversing composites.
- Chain of Responsibility Pattern: A component-parent link is frequently used in this pattern.
- Decorator Pattern: This pattern is frequently used in conjunction with Composite.
- Visitor Pattern: This pattern can be used to extract behavior from the composite and leaf classes.

6. Decorator Pattern

- Composite Pattern: A decorator pattern is frequently used in conjunction with the Composite pattern. The decorator takes on additional responsibilities and, unlike composite pattern, is not intended for object aggregation.

7. Facade Pattern

- Singleton Pattern: This pattern is frequently used because most of the time only one facade object is required.
- Factory Pattern: This pattern, in conjunction with the façade pattern, can be used to create subsystem objects in a subsystem-independent manner.

8. Flyweight Pattern

- Composite Pattern: The flyweight pattern is frequently combined with composite pattern, where the leaf nodes are shared.
- State Pattern and Strategy Pattern: Objects from these two patterns are frequently best implemented as flyweights.

9. Interpreter Pattern

- Iterator Pattern: Use this pattern to traverse the interpreter structure.
- Flyweight Pattern: Use this pattern to share the terminal symbols.
- Composite Pattern: A composite pattern is used in the abstract syntax tree.
- Visitor Pattern: Use this pattern to collect the behavior of all abstract syntax tree nodes in a single class.

10.Iterator Pattern

- Composite Pattern: The iterator is useful for traversing recursive structures such as composites.
- Memento Pattern: Iterators frequently use this pattern to capture the state of a traversal. Rather than using a separate caretaker, the iterator frequently stores the memento internally.
- Factory Pattern: Polymorphic iterators require a factory method to ensure that the correct iterator is created.

11.Mediator Pattern

- Façade Pattern: A facade is a more user-friendly interface to a subsystem. It communicates with the subsystem classes, but never with the Facade. In the case of the Mediator, the colleague objects communicate with the mediator and vice versa.
- Observer Pattern: Using the observer design pattern, colleagues can communicate with mediator objects.

12.Memento Pattern

- Iterator Pattern: Mementos have the ability to iterate.
- Command Pattern: Memento pattern can be used to save the state of an undoable operation.

13. Observer Pattern

- Singleton Pattern: When a change manager is used to facilitate communication between the subject and its observers, only one instance of the change manager is usually required, allowing the Singleton pattern to be used.
- Mediator Pattern: An observer pattern can help mediator colleagues and the mediator object communicate.

14. Prototype Pattern

- Composite Pattern and Decorator Pattern: These patterns are frequently used in designs that benefit from using Prototypes to instantiate objects.
- Factory Pattern: Factory pattern and Prototype pattern can be combined, with factories using prototypes to create product objects.

15. Singleton Pattern

- The Singleton pattern can be used to implement a variety of patterns. Such as factory pattern, builder pattern, and prototype pattern.

16. State Pattern

- Singleton Pattern: In many cases, only one instance of each concrete sate is required. They could then be implemented as singletons.
- Flyweight Pattern: This pattern can be used to share State objects.

17. Strategy Pattern

- Flyweight Pattern: The flyweight pattern is frequently used to share strategy objects.
- Decorator Pattern: A method of altering an object by wrapping it.

18. Template Method Pattern

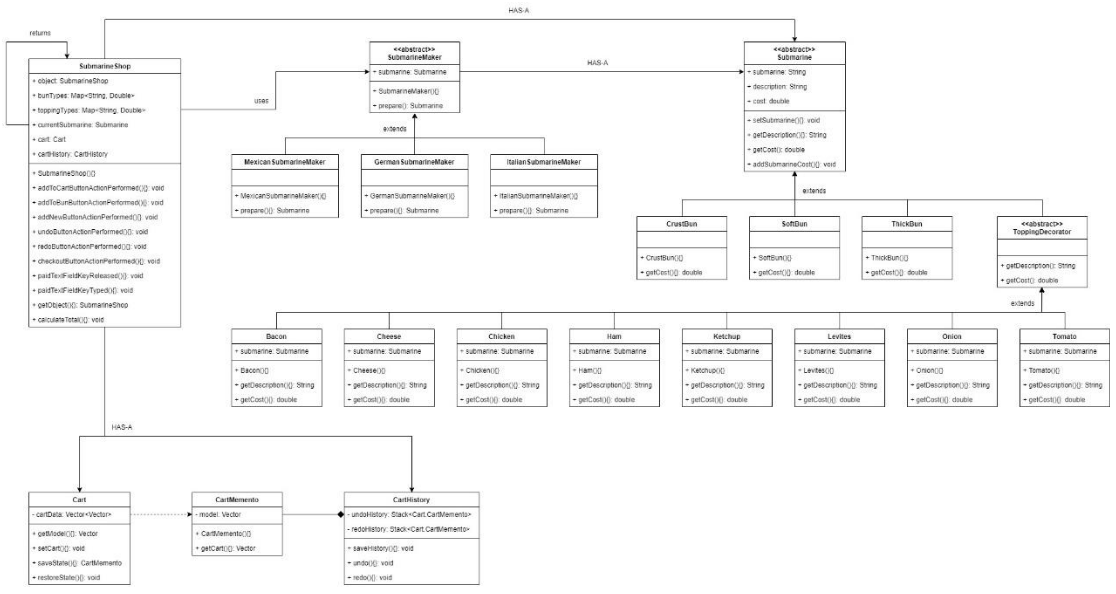
- Strategy Pattern: The template method varies the algorithm through inheritance, whereas the strategy pattern does so through composition and delegation.

- Factory Pattern: To create objects, template methods frequently invoke factory methods.

19. Visitor Pattern

- Interpreter Pattern: Visitors can help with interpretation.
- Composite Pattern: Visitors can be useful for performing operations on a composite object structure.

6. Compound UML diagram.



7. References

Wiktionary. (2018). *compound pattern*. [online] Available at: https://en.wiktionary.org/wiki/compound_pattern [Accessed 10 Oct. 2022].

patterns.dev. (n.d.). *Compound Pattern*. [online] Available at: https://www.patterns.dev/posts/compound-pattern/ [Accessed 10 Oct. 2022].

Wikipedia Contributors (2022). *Model–view–controller*. [online] Wikipedia. Available at:

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#:~:text=Model%E2%80%93view%E2%80%93controller%20(MVC [Accessed 10 Oct. 2022].

GeeksforGeeks. (2018). *MVC Design Pattern - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/mvc-design-pattern/.

Tutorialspoint.com. (2019). *Design Patterns - MVC Pattern - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm.

DEV Community. (n.d.). *Design Pattern: The Compound Pattern*. [online] Available at: https://dev.to/blowideas/design-pattern-the-compound-pattern-35j3 [Accessed 14 Oct. 2022].

Anon, (n.d.). *SOA Patterns | Compound Patterns | Overview | Arcitura Patterns*. [online] Available at: https://patterns.arcitura.com/soa-patterns/compound_patterns/overview.