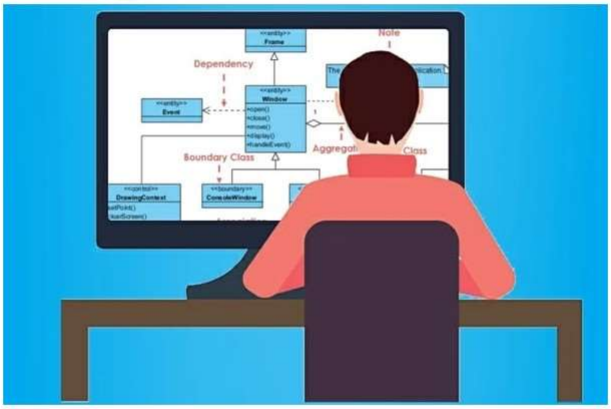
Software Design & Architecture
Spring 2022 - Week-08



مدرس: مہندس ماجد کلیم

جامعہ بحریہ، واقعگاہ کراچی

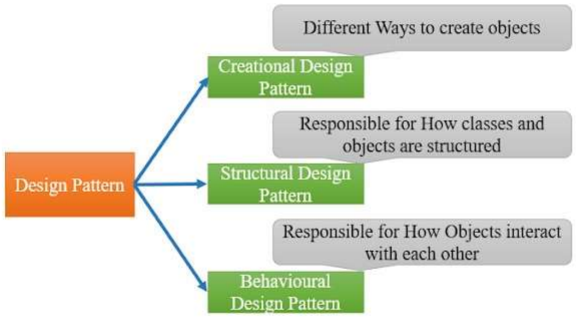
Engr. Majid Kaleem

WEEKLY AGENDA

TENTATIVE WEEKLY DATES		TENTATIVE TOPICS
1	Mar 7th – Mar 11th	INTRODUCTION TO THE COURSE; DEFINING SOFTWARE ARCHITECTURE & DESIGN CONCEPTS
2	Mar 14th – Mar 18th	DESIGN PRINCIPLES; OBJECT-ORIENTED DESIGN WITH UML
3	Mar 21st – Mar 25th	SYSTEM DESIGN & SOFTWARE ARCHITECTURE; OBJECT DESIGN, MAPPING DESIGN TO CODE
4	Mar 28th – Apr 1st	FUNCTIONAL DESIGN; UI DESIGN; WEB APPLICATIONS DESIGN **ASSIGNMENT & QUIZ #1**
5	Apr 4th – Apr 8th	MOBILE APPLICATION DESIGN; PERSISTENCE LAYER DESIGN
6	Apr 11th – Apr 15th	CREATIONAL DESIGN PATTERNS
7	Apr 18th – Apr 22nd	STRUCTURAL DESIGN PATTERNS **ASSIGNMENT & QUIZ #2**
8	Apr 25th – Apr 29th	BEHAVIORAL DESIGN PATTERNS
← MID TERM EXAMINATIONS →		
9	May 9th – May 13th	INTERACTIVE SYSTEMS WITH MVC ARCHITECTURE; SOFTWARE REUSE
10	May 16th – May 20th	ARCHITECTURAL DESIGN ISSUES; ARCHITECTURE DESCRIPTION LANGUAGES (ADLS)
11	May 23rd – May 27th	ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES
12	May 30th – Jun 3rd	ARCHITECTURAL STYLES/PATTERNS & DESIGN QUALITIES **ASSIGNMENT & QUIZ #3**
13	Jun 6th – Jun 10th	QUALITY TACTICS; ARCHITECTURE DOCUMENTATION
14	Jun 13th – Jun 17th	ARCHITECTURAL EVALUATION TECHNIQUES
15	Jun 20th – Jun 24th	MODEL DRIVEN DEVELOPMENT **ASSIGNMENT (PRESENTATIONS) & QUIZ #4**
16	Jun 27th – Jul 1st	REVISION WEEK
← FINAL TERM EXAMINATIONS →		

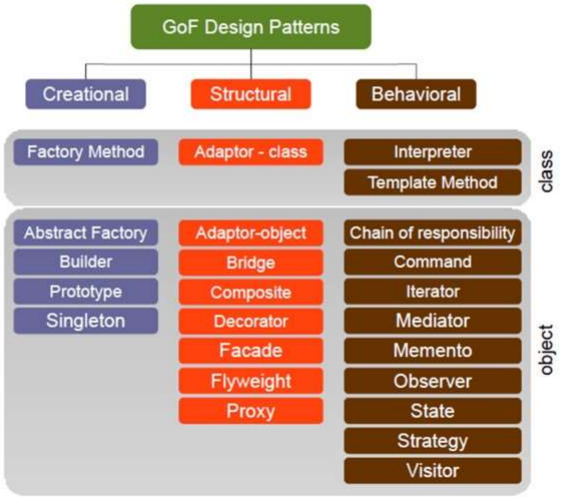
CLASSIFICATION OF DESIGN PATTERNS

- They are categorized in three groups: Creational, Structural, and Behavioral as listed below:

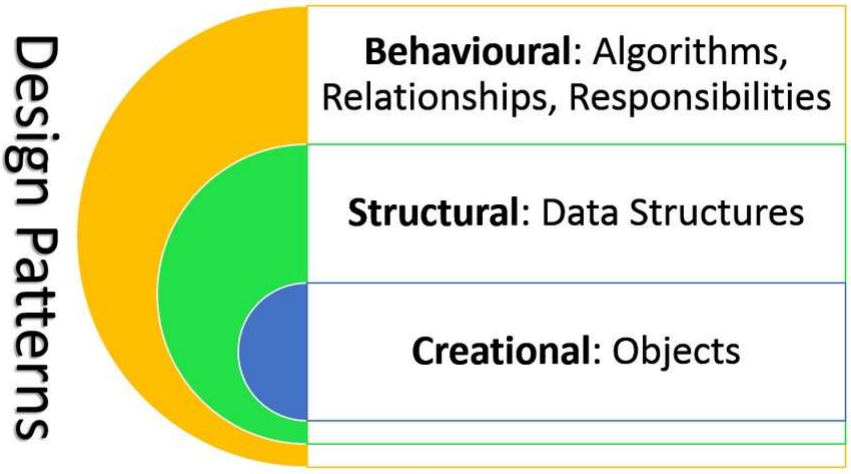


CLASSIFICATION OF DESIGN PATTERNS

- 23 GoF Design Patterns:



Classification of Design Patterns



Useful Web Resources

1. https://sourcemaking.com/design_patterns
2. https://www.dofactory.com/net/design-patterns
3. https://refactoring.guru/design-patterns/catalog

BEHAVIORAL PATTERNS

- Behavioral patterns are concerned with ***algorithms*** and ***communication*** between them.
- The operations that make up a single algorithm might be ***split*** up between different classes, making a complex arrangement that is difficult to manage and maintain.
- The behavioral patterns capture ways of expressing the ***division*** of ***operations*** between classes and optimize how the communication should be handled.

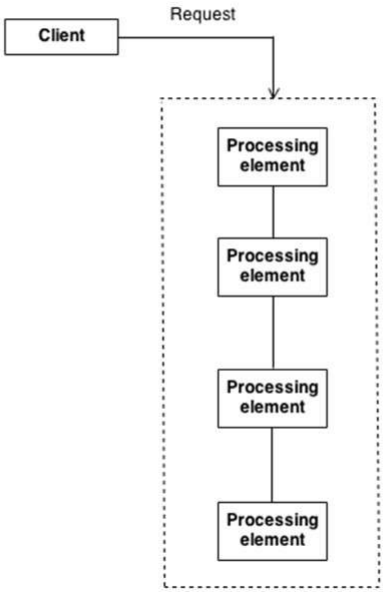
CHAIN OF RESPONSIBILITY

- The Chain of Responsibility pattern works with a list of *Handler* objects that have limitations on the nature of the requests they can deal with.
- If an object cannot handle a request, it passes it on to the next object in the chain.
- At the end of the chain, there can be either default or exceptional behavior.

OR

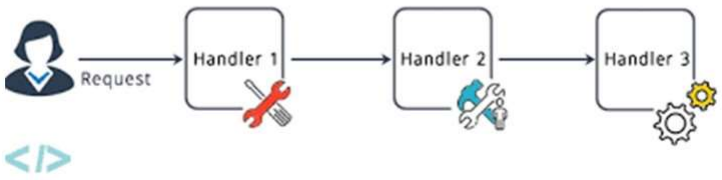
- *A way of passing a request between a chain of objects*
- This pattern defines a chain of processing objects in a chain in such a way that the incoming request is processed by each processing objects in sequence.

CHAIN OF RESPONSIBILITY



CHAIN OF RESPONSIBILITY EXAMPLE

- Imagine members of the public (such as you and me) approaching a bank, a government office, a hospital, or a similar institution with a request.
- In the case of a bank, we know that a first level of clerks will handle straightforward requests.
- Those requests requiring more inspection will be sent up to a supervisor, and ultimately the manager might have to authorize or rule on a request.
- The clerk, supervisor, an manager together are an example of a chain of responsibility.



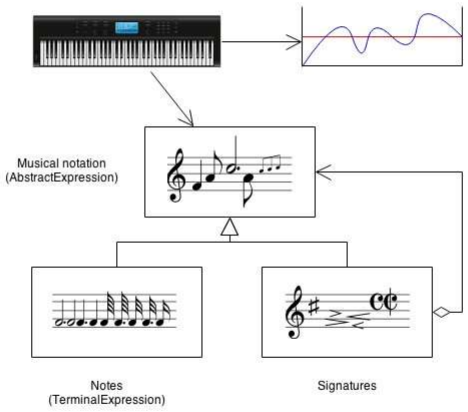
INTERPRETER PATTERN

- The *Interpreter* pattern defines a *grammatical representation* for a language and an interpreter to interpret the grammar.
- Interpreter pattern allows us to interpret grammar in to code solutions.
- The Interpreter pattern supports the interpretation of instructions written in a language or notation defined for a specific purpose.
- The notation is precise and can be defined in terms of a grammar.

INTERPRETER PATTERN EXAMPLE

- Musicians are examples of *Interpreters.*
- The pitch of a sound and its duration can be represented in musical notation on a staff.
- This notation provides the language of music.
- Musicians playing the music from the score are able to reproduce the original pitch and duration of each sound represented.
- An example of this is the musical keyboard, which interprets clicks on keys for specific tones and notes.
- Implement an interpreter for a language, first defining a formal *grammar* for that language and then implementing that grammar with a hierarchy of classes (one subclass per *production* or *nonterminal*).

INTERPRETER PATTERN EXAMPLE



STATE PATTERN

- State pattern allows an object to change its behavior depending on the current values of the object.
- When the state inside an object changes, it can change its behavior by switching to a set of different operations.
- This is achieved by an object variable changing its subclass, within a hierarchy.

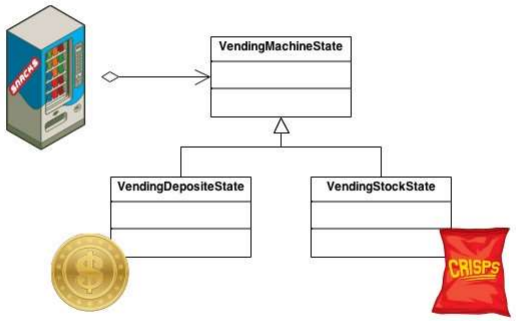
STATE PATTERN EXAMPLE

- The buttons and switches in your smartphone behave differently depending on the current state of the device:
- When the phone is unlocked, pressing buttons leads to executing various functions.
- When the phone is locked, pressing any button leads to the unlock screen.
- When the phone's charge is low, pressing any button shows the charging screen.

STATE PATTERN EXAMPLE

- The State pattern allows an object to change its behavior when its internal state changes.
- This pattern can be observed in a vending machine.
- Vending machines have states based on the inventory, amount of currency deposited, the ability to make change, the item selected, etc.
- When currency is deposited and a selection is made, a vending machine will either deliver a product and no change, deliver a product and change, deliver no product due to insufficient currency on deposit, or deliver no product due to inventory depletion.

STATE PATTERN EXAMPLE

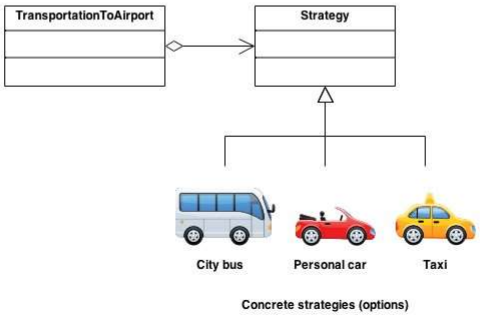


STRATEGY PATTERN

- A *Strategy* defines a *set of algorithms* that can be used *interchangeably*.
- The Strategy pattern involves removing an algorithm from its host class and putting it in a separate class.
- There may be different algorithms (strategies) that are applicable for a given problem.
- The Strategy pattern enables a client to choose which algorithm to use from a family of algorithms and gives it a simple way to access it.
- The algorithms can also be expressed independently of the data they are using.

STRATEGY PATTERN EXAMPLE

- Modes of transportation to an airport is an example of a *Strategy*
- Imagine that you have to get to the university. You can catch a bus, order a cab, or get on your motorbike.
- These are your transportation strategies.
- You can pick one of the strategies depending on factors such as budget or time constraints.



TEMPLATE PATTERN

- The Template Method pattern enables algorithms to defer certain steps to subclasses.
- The structure of the algorithm does not change, but small well-defined parts of its operation are handled elsewhere.
- In template pattern we have an abstract class which acts as a skeleton for its inherited classes.
- The inherited classes get the shared functionality.
- The inherited classes take the shared functionality and add enhancements to the existing functionality.

TEMPLATE PATTERN EXAMPLE

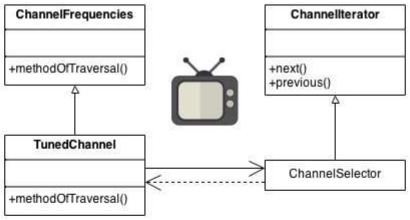
- In word or power point how we take templates and then prepare our own custom presentation using the base class.
- Template classes works on the same fundamental.

ITERATOR PATTERN

- *Sequentially access the elements of a collection.*
- The Iterator provides ways to access elements of an ***aggregate object sequentially*** without exposing the underlying structure of the object.

ITERATOR PATTERN EXAMPLE

- On early television sets, a dial was used to change channels. When channel surfing, the viewer was required to move the dial through each channel position, regardless of whether or not that channel had reception. On modern television sets, a next and previous button are used. When the viewer selects the "next" button, the next tuned channel will be displayed.



24

COMMAND PATTERN

- *Encapsulate a command request as an object*
- In this pattern an object is used to represent and encapsulate all the information needed to call a method at a later time..

COMMAND PATTERN EXAMPLE

- Suppose you get to a nice restaurant and sit at the table by the window.
- A friendly waiter approaches you and quickly takes your order, writing it down on a piece of paper.
- The waiter goes to the kitchen and sticks the order on the wall.
- After a while, the order gets to the chef, who reads it and cooks the meal accordingly.
- The cook places the meal on a tray along with the order.
- The waiter discovers the tray, checks the order to make sure everything is as you wanted it, and brings everything to your table.
- ***The paper order serves as a command.***
- It remains in a queue until the chef is ready to serve it.
- The order contains all the relevant information required to cook the meal.
- It allows the chef to start cooking right away instead of running around clarifying the order details from you directly.

MEDIATOR PATTERN EXAMPLE

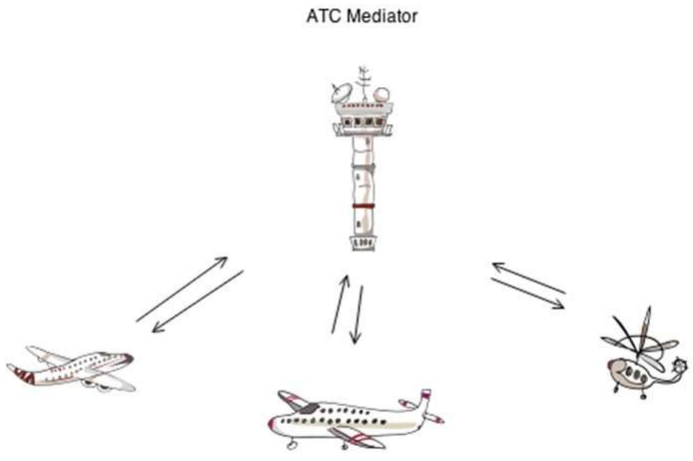
- *Defines simplified communication between classes*
- In this pattern communication between objects is encapsulated with a mediator object.
- Objects no longer communicate directly with each other, but instead communicate through the mediator.

MEDIATOR PATTERN EXAMPLE

- Pilots of aircraft that approach or depart the airport control area don't communicate directly with each other.
- Instead, they speak to an air traffic controller, who sits in a tall tower somewhere near the airstrip.
- Without the air traffic controller, pilots would need to be aware of every plane in the vicinity of the airport, discussing landing priorities with a committee of dozens of other pilots.
- That would probably skyrocket the airplane crash statistics.
- The tower doesn't need to control the whole flight.
- It exists only to enforce constraints in the terminal area because the number of involved actors there might be overwhelming to a pilot.

28

MEDIATOR PATTERN EXAMPLE



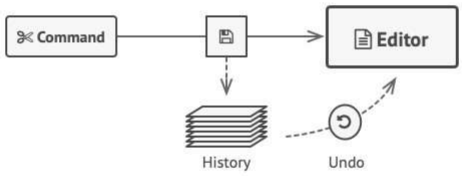
29

MEMENTO DESIGN PATTERN

- *Capture and restore an object's internal state*
- Mementos capture and externalize an object's internal state allowing the object to be restored to this state later.

MEMENTO DESIGN PATTERN-EXAMPLE

- The *Memento* captures and externalizes an object's internal state, so the object can be restored to that state later.



OBSERVER DESIGN PATTERN

- *A way of notifying change to a number of classes*
- An observable object called 'Subject' maintains a list of objects called 'Observers'. Subject notifies the observers of any state changes.

OBSERVER PATTERN EXAMPLE

- **Observer** design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- If you subscribe to a newspaper or magazine, you no longer need to go to the store to check if the next issue is available. Instead, the publisher sends new issues directly to your mailbox right after publication or even in advance.
- The publisher maintains a list of subscribers and knows which magazines they're interested in. Subscribers can leave the list at any time when they wish to stop the publisher sending new magazine issues to them.

STATE DESIGN PATTERN

- *Alter an object's behavior when its state changes.*

OR

- State pattern allows an object to alter its behavior when its internal state changes.

STATE PATTERN EXAMPLE

- The buttons and switches in your smartphone behave differently depending on the current state of the device:
- When the phone is unlocked, pressing buttons leads to executing various functions.
- When the phone is locked, pressing any button leads to the unlock screen.
- When the phone's charge is low, pressing any button shows the charging screen.

STRATEGY PATTERN EXAMPLE

- *Encapsulates an algorithm inside a class.*
- This pattern defines a family of algorithms, encapsulate each one, and make them interchangeable.

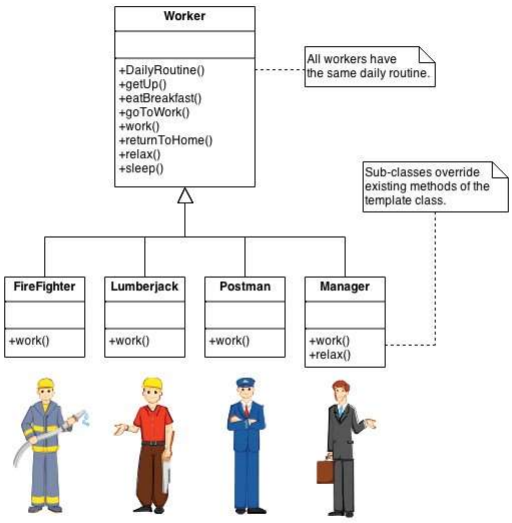
STRATEGY PATTERN

- A *Strategy* defines a set of algorithms that can be used interchangeably.
- Modes of transportation to an airport is an example of a *Strategy*
- Imagine that you have to get to the university. You can catch a bus, order a cab, or get on your motorbike. These are your transportation strategies. You can pick one of the strategies depending on factors such as budget or time constraints.

TEMPLATE METHOD

- *Defer the exact steps of an algorithm to a subclass.*
- Pattern defines steps of an algorithm as methods in an interface while allowing subclasses to override some steps in the overall algorithm.
- Template Method design pattern defines the ***skeleton*** of an ***algorithm*** in the ***superclass*** but lets ***subclasses override*** specific steps of the algorithm without changing its structure.
- The *Template Method* defines a skeleton of an algorithm in an operation, and ***defers*** some steps to ***subclasses***.

TEMPLATE METHOD



TEMPLATE METHOD

- Home builders use the *Template Method* when developing a new subdivision.
- A typical subdivision consists of a limited number of floor plans, with different variations available for each floor plan.
- Within a floor plan, the foundation, framing, plumbing, and wiring will be identical for each house.

VISITOR PATTERN

- It is used when we have to perform ***an operation*** on a ***group of similar kind of Objects***. With the help of visitor pattern, we can move the operational logic from the objects to another class.
- It is a way of ***separating an algorithm*** from an object structure on which it operates.
- A practical result of this separation is the ability to ***add new operations*** to existing object structures without modifying the structures.

VISITOR PATTERN

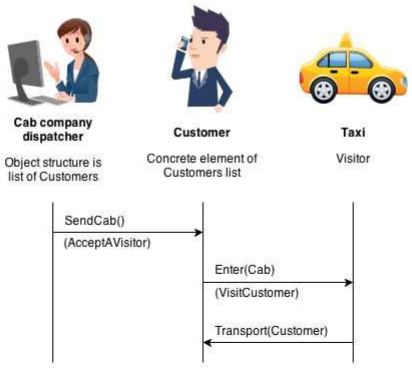
- Imagine an insurance agent who's eager to get new customers.
- He can visit every building in a neighborhood, trying to sell insurance to everyone he meets.
- Depending on the type of organization that occupies the building, he can offer ***specialized*** insurance policies:
- If it's a residential building, he sells medical insurance.
- If it's a bank, he sells theft insurance.
- If it's a coffee shop, he sells fire and theft insurance.

VISITOR DESIGN PATTERN-EXAMPLE

- Shopping in the supermarket is another common example, where the shopping cart is your set of ***elements***.
- When you get to the checkout, the cashier acts as a ***visitor***, taking the disparate set of elements (your shopping), some with prices and others that need to be weighed, in order to provide you with a total.

VISITOR PATTERN EXAMPLE

- This pattern can be observed in the operation of a taxi company.
- When a person calls a taxi company (accepting a visitor), the company dispatches a cab to the customer.
- Upon entering the taxi the customer, or ***Visitor***, is no longer in control of his or her own transportation, the taxi (driver) is..




```
If(anyQuestions)
{
  askNow();
}
else
{
  thankYou();
  submitAttendance();
  endClass();
}
```

REFERENCES

1. *Software Architecture, Perspectives on an Emerging Discipline By Mary Shaw & David Garlan*
2. *The Art of Software Architecture, Design Methods & Techniques By Stephen T. Albin*
3. *Essential Software Architecture By Ian Gorton*
4. *Microsoft Application Architecture Guide By Microsoft*
5. *Design Patterns, Elements of Reusable Object-Oriented Software By by Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides*
6. *Refactoring, Improving the Design of Existing Code By Martin Fowler & Kent Beck*