BEHAVORIAL DESIGN PATTERNS

WHAT IS IT ?

- Behavioral design patterns are concerned with **algorithms and interaction and the assignment of responsibilities between objects**.

- In these design patterns, the interaction between the objects should be in such a way that they can easily talk to each other and still should be loosely coupled.

- Behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.

THUMB RULES

- **Chain of responsibility** passes a sender request along a chain of potential receivers. **Command** normally specifies a sender-receiver connection with a subclass. **Mediator** has senders and receivers reference each other indirectly. **Observer** defines a very decoupled interface that allows for multiple receivers to be configured at run-time.

1. **Chain of responsibility** can use **Command** to represent requests as objects.

2. **Chain of responsibility** is often applied in conjunction with **Composite**. There, a component's parent can act as its successor.

3. **Command** and **Memento** act as magic tokens to be passed around and invoked at a later time. In **Command**, the token represents a request; in **Memento**, it represents the internal state of an object at a particular time. Polymorphism is important to **Command**, but not to **Memento** because its interface is so narrow that a memento can only be passed as a value.

4. Command can use **Memento** to maintain the state required for an undo operation.

5. Macro **Command**s can be implemented with **Composite**.

6. A **Command** that must be copied before being placed on a history list acts as a **Prototype**.

7. Interpreter can use **State** to define parsing contexts.

8. Terminal symbols within **Interpreter**'s abstract syntax tree can be shared with **Flyweight**.

9. Iterator can traverse a **Composite**. **Visitor** can apply an operation over a **Composite**.

10. Polymorphic **Iterator**s rely on **Factory Method**s to instantiate the appropriate **Iterator** subclass.

11. **Mediator** and **Observer** are competing patterns. The difference between them is that **Observer** distributes communication by introducing "observer" and "subject" objects, whereas a **Mediator** object encapsulates the communication between other objects.

12. **Mediator** can leverage **Observer** for dynamically registering colleagues and communicating with them.

13. **Memento** is often used in conjunction with **Iterator**. An **Iterator** can use a **Memento** to capture the state of an iteration. The **Iterator** stores the **Memento** internally.

14. **State** is like **Strategy** except in its intent.

15. **Flyweight** explains when and how **State** objects can be shared.

16. **State** objects are often **Singleton**s.

17. **Strategy** lets you change the guts of an object. **Decorator** lets you change the skin.

18. **Strategy** is to algorithm. as **Builder** is to creation.

19. Strategy objects often make good **Flyweight**s.

20. Strategy is like **Template method** except in its granularity.

21. Template method uses inheritance to vary part of an algorithm. **Strategy** uses delegation to vary the entire algorithm.

22. The **Visitor** pattern is like a more powerful **Command** pattern because the visitor may initiate whatever is appropriate for the kind of object it encounters.