1604207337036

MJCET Assignment / Tutorial Sheet Page No. 1

① Discuss the intent, motivation, structure of facade pattern?

Facade pattern is a structural design pattern commonly used in software engineering. It's primary intent is to provide a simplified interface to a larger body of code, thus hiding the complexities of the subsystems and providing a simplified interface to a larger body of code, thus hiding the complexities of the subsystems and providing a unified interface for clients to interact with.

Intent:

Simplify Complex Systems:

⊙ The facade pattern aims to simplify the interface of a complex system by providing a higher-level interface that shields clients from its complexities.

Encapsulates subsystems:

⊙ It encapsulates the interactions and dependencies between multiple subsystems, providing a unified interface to access them.

Motivation:

Client Simplification:

- For clients who only need to interact with certain aspects of a system, the facade pattern provides a more straightforward and intuitive interface, abstracting away unnecessary details.

Subsystem Decoupling:

- By using a facade, subsystems can be decoupled from the clients, allowing for easier modification or replacement of the subsystems without affecting client code.

Structure:

Facade:

- This is the core of the pattern, providing a unified interface to the subsystems.
- It knows which subsystems are responsible for a particular request and delegates client requests to appropriate objects within the subsystems.

Client:

- This is the code that interacts with the facade to access the system.
- Clients interact only with the facade, unaware of the individual subsystems and their complexities.

Example:

Consider a multimedia player application with subsystems for Audioplayer, Videoplayer, playlist manager. Facade pattern can be applied to provide a simplified interface to the multimedia functionalities:

- Facade: Multimedia Player Facade
- Subsystems:
  - → Audio Player
  - → Video Player
  - → Playlist Manager.
- Client: Application code that interacts with Multimedia Player Facade to play audio, video and manage playlist.

Q2 Explain the Consequences and implementation of Interpreter Pattern?

The interpreter pattern is a behavioural design pattern that is used to define the grammar of a language and to interpret sentences in that language. It involves defining a language grammar as a set of domain-specific language constructs, and providing an interpreter that understands and executes these constructs.

Consequences:

① Flexibility:

⊙ The interpreter pattern makes it easy to change and extend the grammar of a language by adding new expressions or modifying existing ones.

⊙ Flexibility allows for the adaption of the language to changing requirements without modifying the interpreter.

② Performance:

⊙ Depending on the Complexity of the language grammar and the efficiency of the interpreter implementation, there maybe performance overhead associated with interpreting sentences.

③ Ease of Maintaince:

⊙ Despite the potential Complexity, the pattern can improve the maintainability of Code by encapsulating language-related behaviour within expression classes.

⊙ This makes it easier to understand, modify and extend the interpreter.

160420737036

MJCET Assignment / Tutorial Sheet Page No. 5

Implementation:

① Define the Grammar:

⊙ Begin by defining the grammar of the language that the interpreter will understand

⊙ This involves identifying the language constructs and their syntactic rules.

② Build the Abstract Syntax Tree (AST):

⊙ As the interpreter processes input sentences, it constructs an abstract syntax tree representing the structure of the sentences according to the grammar.

③ Implement the interpreter:

⊙ Create an interpreter class that traverses the AST and interprets the input sentences by evaluating the expressions in classes.

④ Client Code:

⊙ We write the client code that creates instances of the interpreter and uses it to interpret sentences in the language.

By following these steps, we can implement the interpreter pattern to interpret sentences in a custom language, such as mathematical expressions, query language or configuration languages.

Q3 Explain the motivation and Consequences of Command pattern?

Motivation:

① Support for Queues and logging:

⊙ Commands can be stored in queues, allowing for the implementation of features like task scheduling or deferred execution.

⊙ Additionally, Commands can be logged or persisted, providing a history of executed actions for auditing or debugging purposes.

② Dynamic behaviour:

⊙ Since Commands are objects, they can be created and configured dynamically at runtime.

⊙ This allows for the implementation of dynamic behaviour, such as building Composite Commands from smaller ones or modifying Command parameters based on user input.

Consequences:

① Flexibility:

⊙ The Command pattern promotes flexibility by allowing Commands to be parameterized, queued and composed dynamically.

⊙ This flexibility enables the implementation of complex behaviour without tightly coupling the sender and receiver object.

② Complexity:

⊙ The overhead is often outweighed by benefits of improved flexibility and maintainability.

160420737056

MJCET Assignment / Tutorial Sheet Page No. 7

(4) Short notes on
a) Black board 5) Layers Pattern

a) Black Board Pattern:

Description:

⊙ Blackboard pattern is a design pattern used in systems where complex problems can be broken into smaller, more manageable tasks that can be worked on independently.

Components:

⊙ Blackboard:
→ Central repository where data, knowledge and hypotheses are stored.

⊙ Controller:
→ oversees the collaboration process, co-ordinating the interactions between agents and managing the flow of information on the blackboard.

Applications:

⊙ AI & Expert Systems:
→ used in AI and expert systems for complex problem-solving tasks, such as medical diagnosis or strategic planning

⊙ Data Analysis:
→ Applied in data-intensive domains like predictive analytics and pattern recognition, where multiple algorithms analyze data collaboratively.

b) Layers Pattern:

⊙ Description:

→ promotes seperation of concerns and modulization by enforcing a clear seperation between different types of responsibilities or functionalities.

⊙ Benefits:

→ Facilitates Scalability by allowing individual layers to be Scaled independently based on performance requirements

⊙ Considerations

→ choosing the right layering strategy depends on the specific requirements and architecture of the system.

⊙ Examples:

⊙ MVC (Model - View - Controller):

→ Seperates user interface, business logic, and data access into three distinct layers.

⊙ TCP/IP protocol Stack:

→ Organizes network Communication into multiple layers, Such as application, transport, network, physical layers, each responsible for specific tasks.

Q5 Explain in detail the essential steps of the Component system engineering process.

(1) Requirement Analysis:

⊙ Identify Requirements

→ Gather and analyze the requirements of the system to be developed, including functional and non-functional requirements

(2) Component - Selection:

⊙ Make or Buy Decision:

→ Decide whether to develop custom components in house or acquire off the shelf components from external sources

(3) Component design & implementation:

⊙ Consider Stands, Develop Components and Testing:

→ understand the stands of the market develop a new product or shape up to the client's requirements by developing and before deploying test the product or project before deployment or delivery.

(4) Deployment and maintaince

⊙ After Successful testing comes the step for deployment and post delivery the project or product must be taken care by updating and ensuring the reliability and security.

Documentation and knowledge management

⊙ Documentation:

→ By Documenting the projects SDLC future projects would be done on less cost and delivery of project can be deployed beyond deadline.

⊙ Knowledge transfer:

→ Giving the knowledge sessions to the team members by documenting best practices, lessons learned and troubleshooting tips.