GOF

Naveen Kumar K S
Adith.Naveen@gmail.com

Introduction

- Design patterns represent the best practices used by experienced object-oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development
- All Design Patterns are got based on trial and error by software engineers.

GOF?

- Discussed in 1994
- By Four people
  - Erich Gamma
  - Richard Helm
  - Ralph Johnson
  - John Vlissides
- To encourage people to go with reusable coding

Types of Design Patterns

- There are 23 design patterns which can be classified in three categories: **Creational, Structural and Behavioral patterns.**
- Another design pattern J2EE will be discussed in other course

Creational Patterns

Abstract Factory	Creates an instance of several families of classes
Builder	Separates object construction from its representation
Factory Method	Creates an instance of several derived classes
Prototype	A fully initialized instance to be copied or cloned
Singleton	A class of which only a single instance can exist

Structural Patterns

Adapter	Match interfaces of different classes
Bridge	Separates an object’s interface from its implementation
Composite	A tree structure of simple and composite objects
Decorator	Add responsibilities to objects dynamically
Facade	A single class that represents an entire subsystem
Flyweight	A fine-grained instance used for efficient sharing
Proxy	An object representing another object

Behavioral Patterns

Chain of Resp.	A way of passing a request between a chain of objects
Command	Encapsulate a command request as an object
Interpreter	A way to include language elements in a program
Iterator	Sequentially access the elements of a collection
Mediator	Defines simplified communication between classes
Memento	Capture and restore an object's internal state
Observer	A way of notifying change to a number of classes
State	Alter an object's behavior when its state changes
Strategy	Encapsulates an algorithm inside a class
Template Method	Defer the exact steps of an algorithm to a subclass
Visitor	Defines a new operation to a class without change

Singleton

- Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under **creational pattern** as this pattern provides one of the best ways to create an object.

Factory Pattern

- Factory pattern is one of most used design pattern in Java. This type of design pattern comes under **creational pattern** as this pattern provides one of the best ways to create an object.
- In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Abstract Factory Pattern

- Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under **creational pattern** as this pattern provides one of the best ways to create an object.

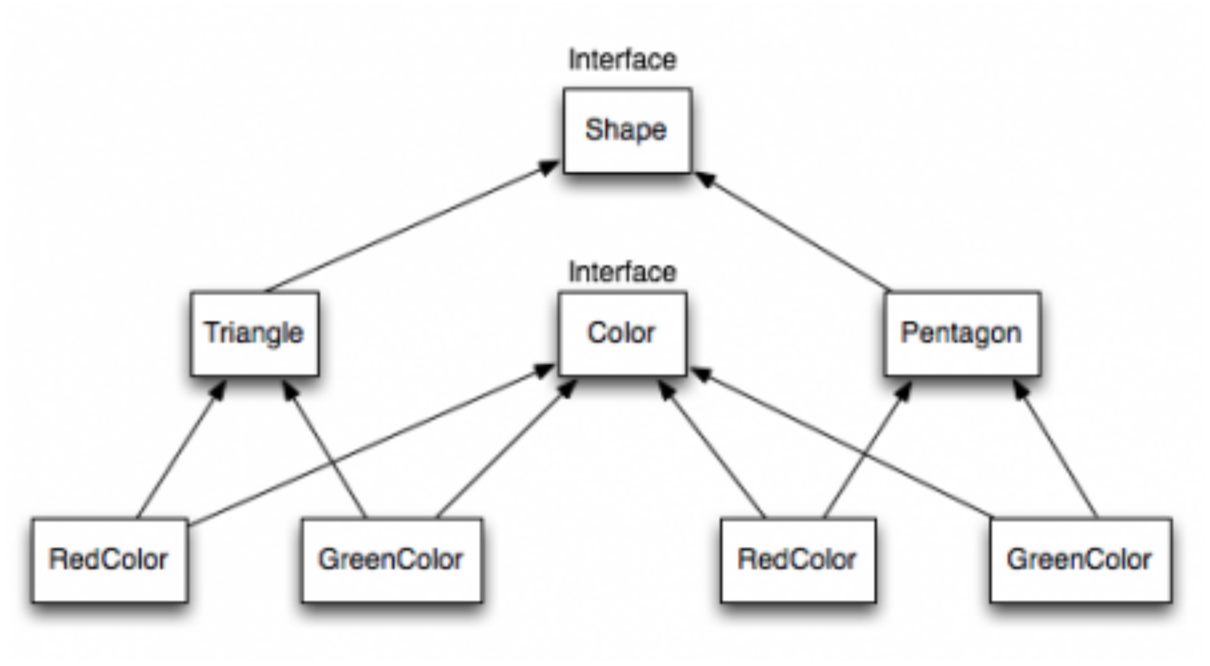
Prototype Pattern

- Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under **creational pattern** as this pattern provides one of the best ways to create an object.

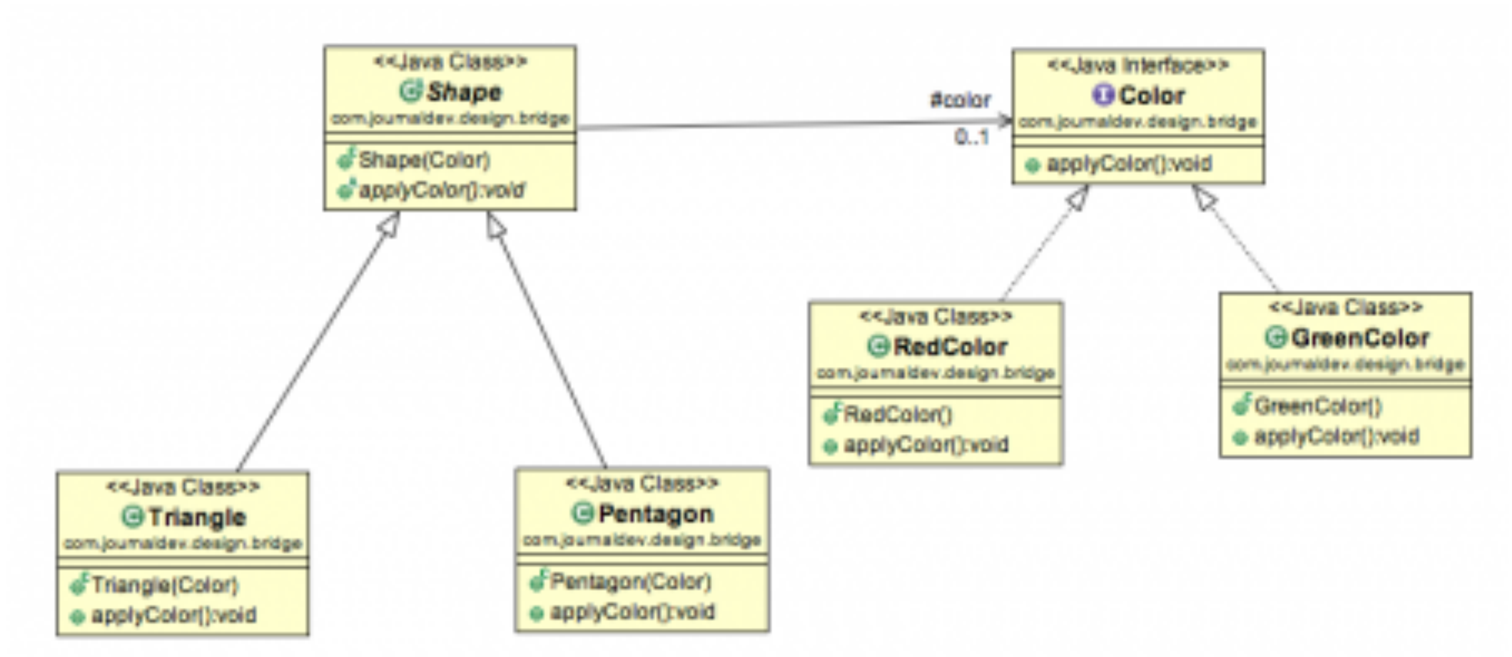
Bridge Pattern

- Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under **structural pattern** as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

Need for Bridge Pattern



Bridge Pattern



Filter Pattern

- Filter pattern or Criteria pattern is a design pattern that enables developers to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations. This type of design pattern comes under **structural pattern** as this pattern combines multiple criteria to obtain single criteria.

Decorator Pattern

- Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under **structural pattern** as this pattern acts as a wrapper to existing class.

Façade Pattern

- Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under **structural pattern** as this pattern adds an interface to existing system to hide its complexities.

Observer Pattern

- Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

Delegate Pattern

- Business Delegate Pattern is used to decouple presentation tier and business tier. It is basically use to reduce communication or remote lookup functionality to business tier code in presentation tier code. In business tier we have following entities.

Thank You

J2EE Design Pattern

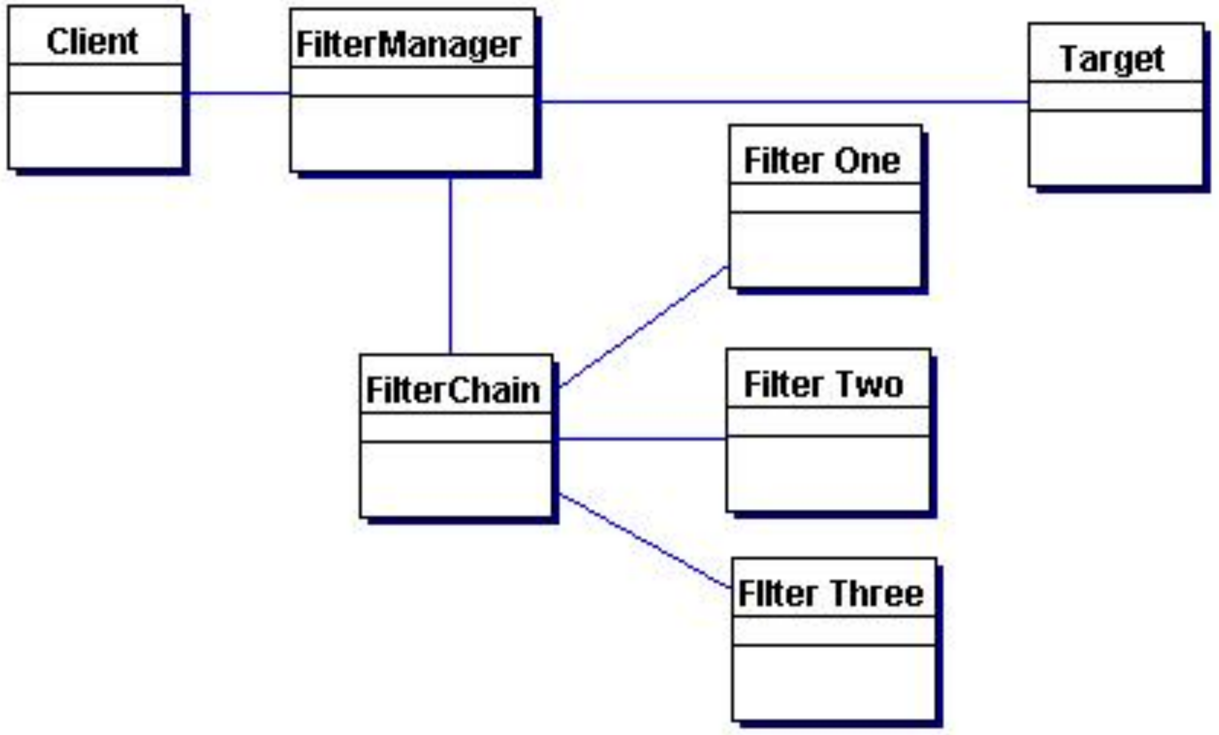
J2EE Patterns

- Presentation Tier Patterns
- Business Tier Patterns
- Integration Tier Patterns

Intercepting Filter

- intercepts incoming requests and outgoing responses and applies a filter.
- These filters may be added and removed in a declarative manner
- For an incoming request, this is often a Front Controller

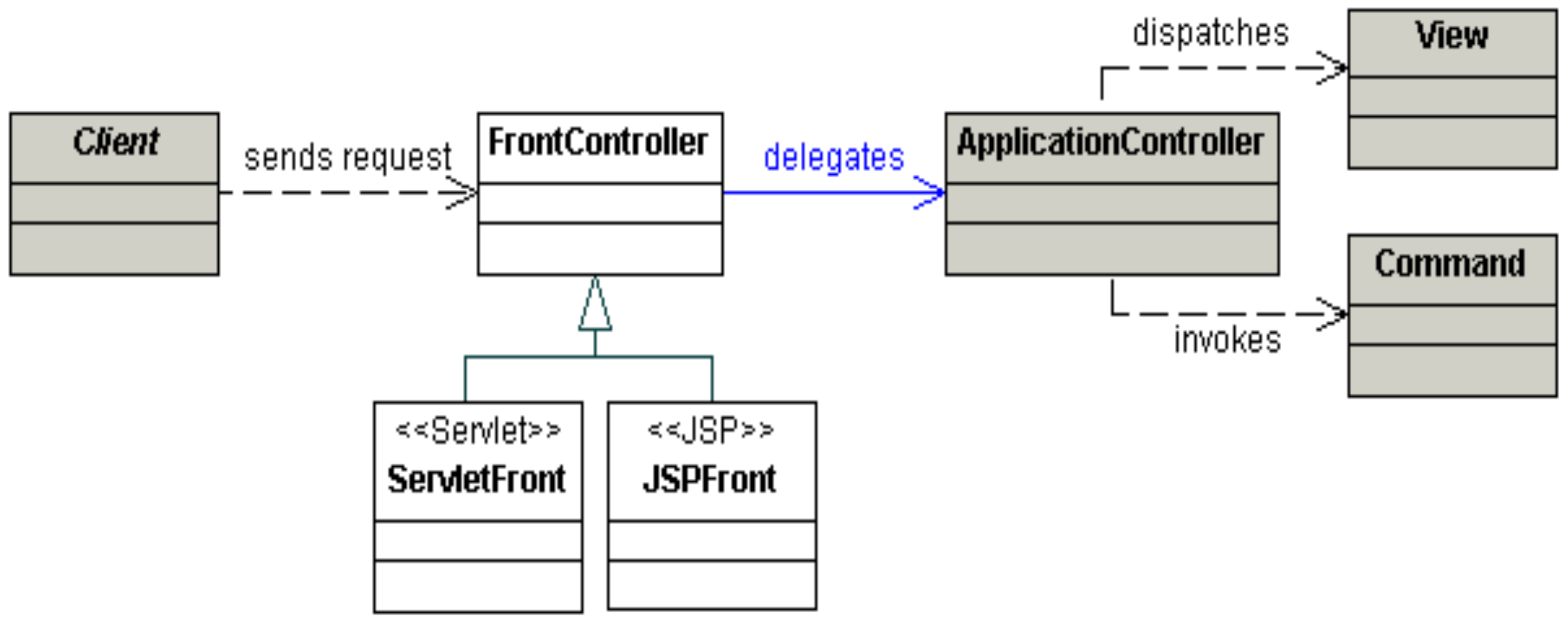
Intercepting Filter



Front Controller

- Is a container to hold the common processing logic that occurs within the presentation tier
- A controller handles requests and manages content retrieval, security, view management, and navigation, delegating to a Dispatcher component to dispatch to a View.
- The Front Controller pattern creates central control logic for presentation request handling.

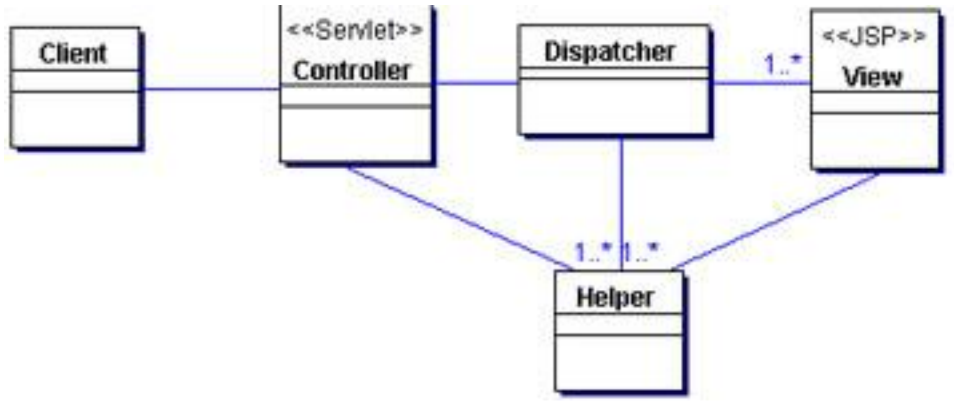
Front controller



Dispatcher View

- Combines a controller and dispatcher with views and helpers to handle client requests and prepare a dynamic presentation as the response.
- A dispatcher is responsible for view management and navigation and can be encapsulated either within a controller, a view, or a separate component.
- The Dispatcher View pattern handles the request and generates a response while managing limited business processing.

Dispatcher View

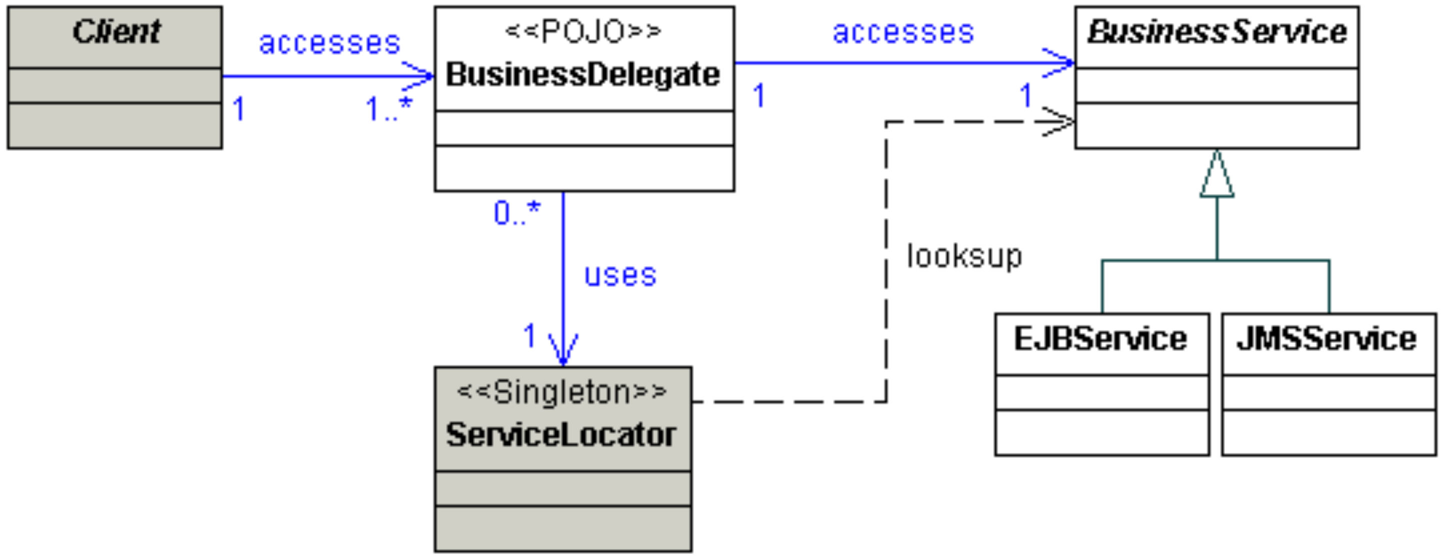


Business Tier Patterns

Business Delegate

- Reduces coupling between remote tiers and provides an entry point for accessing remote services in the business tier.
- A Business Delegate might also cache data as necessary to improve performance.
- A Business Delegate encapsulates a Session Façade and maintains a one-to-one relationship with that Session Façade.
- You want to hide clients from the complexity of remote communication with business service components.

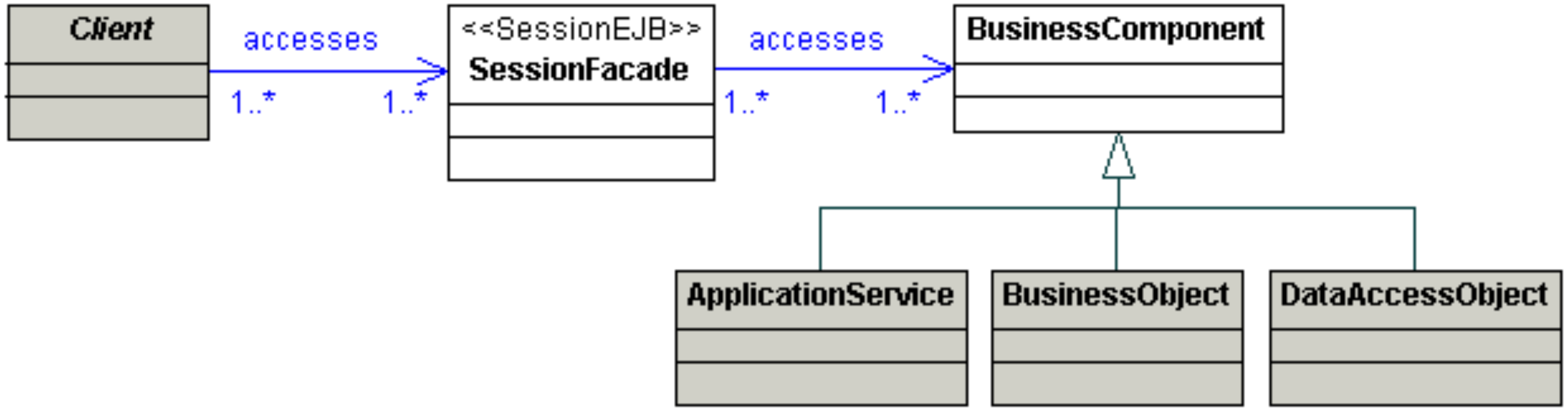
Business Delegate



Session Façade

- Provides coarse-grained services to the clients by hiding the complexities of the business service interactions.
- This is the same as a Façade pattern as described in GOF Design Patterns, but just provides an interface to a service instead of code.

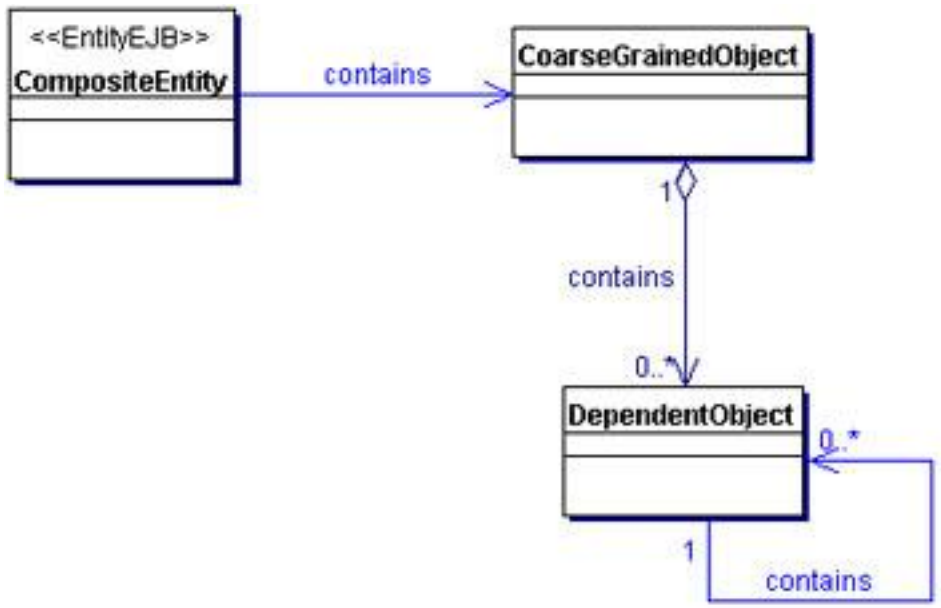
Session Façade



Composite Entity

- Implements a Business Object using local entity beans and POJOs.
- When implemented with bean-managed persistence, a Composite Entity uses Data Access Objects to facilitate persistence.
- to avoid the drawbacks of remote entity beans, such as network overhead and remote inter-entity bean relationships.

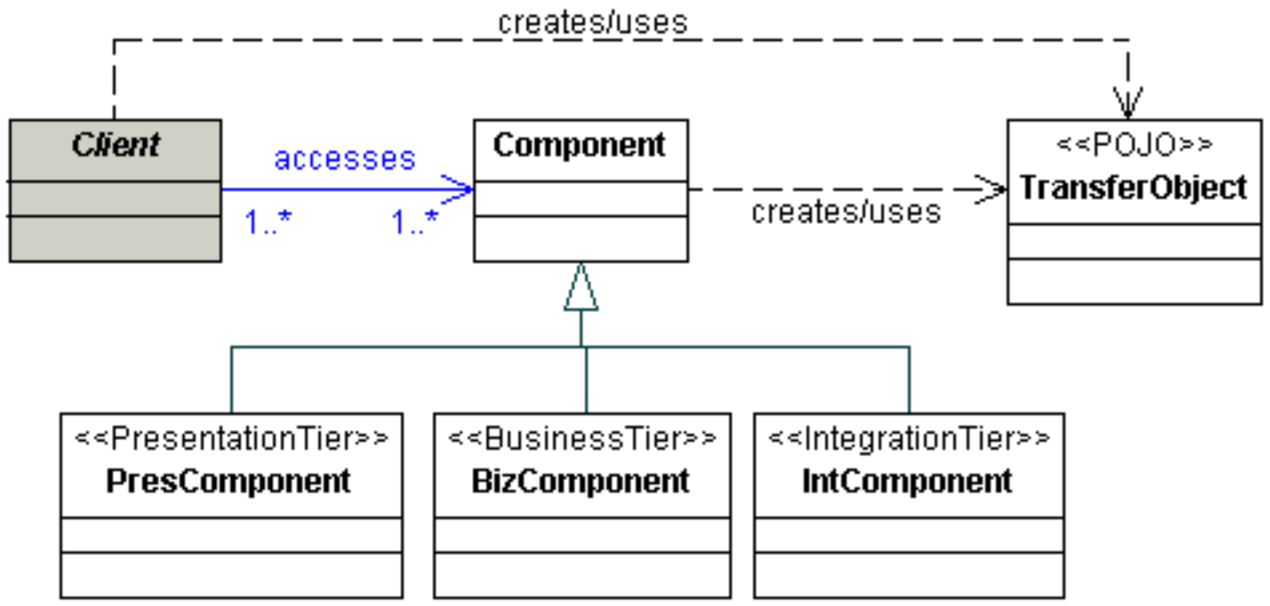
Composite Entity



Transfer Object

- You want clients to access components in other tiers to retrieve and update data.
- You want to reduce remote requests across the network.
- You want to avoid network performance degradation caused by chattier applications that have high network traffic.
- Reduces network traffic
- Simplifies remote object and remote interface

Transfer Object

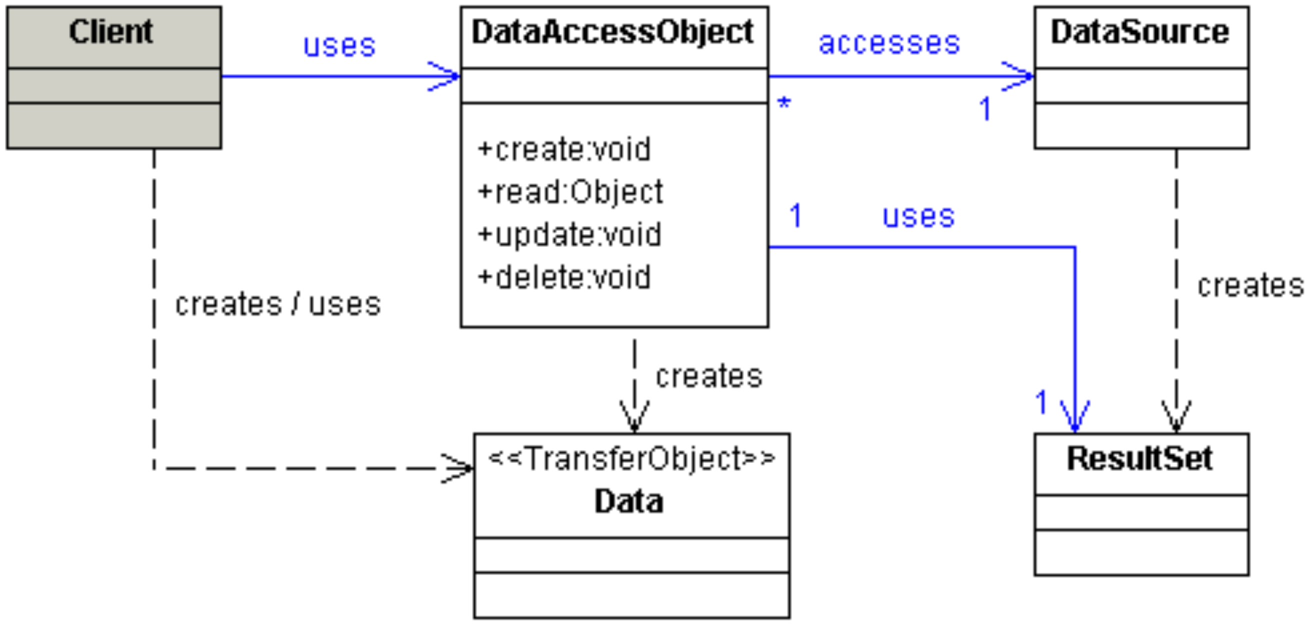


Integration Tier Patterns

Data Access Object

- Enables loose coupling between the business and resource tiers.
- Data Access Object encapsulates all the data access logic to create, retrieve, delete, and update data from a persistent store.
- Data Access Object uses Transfer Object to send and receive data.

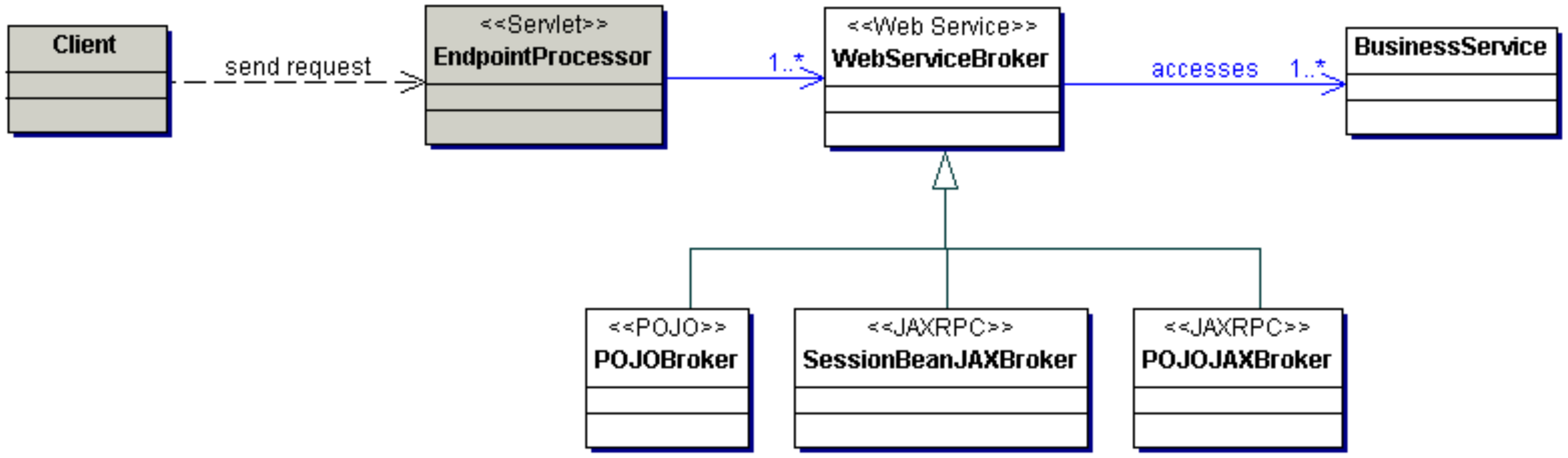
Data Access Object



Web Service Broker

- Exposes and brokers one or more services in your application to external clients as a web service using XML and standard web protocols.
- A Web Service Broker can interact with Application Service and Session Façade.
- A Web Service Broker uses one or more Service Activators to perform asynchronous processing of a request.
- The Web Service Broker pattern exposes and brokers services using XML and web protocols.

Web Service Broker



Question??