DESIGN PATTERNS

OPTIMIZE YOUR CODE

Agenda

- Model View Controller (MVC)
- Service Class
- DAO Pattern

M V C

MODEL VIEW CONTROLLER

What is MVC Architecture?

- Model–view–controller (usually known as MVC) is a software design pattern
- Commonly used for developing user interfaces that divides the related program logic into three interconnected elements.
- This is done to separate internal representations of information from the ways information is presented to and accepted from the user
- This kind of pattern is used for designing the layout of the page.

What is MVC Architecture?

- Traditionally used for desktop graphical user interfaces (GUIs), this pattern has become popular for designing web applications.

- Popular programming languages like JavaScript, Python, Ruby, PHP, **Java**, C#, and Swift have MVC frameworks that are used for web or mobile application development straight out of the box.

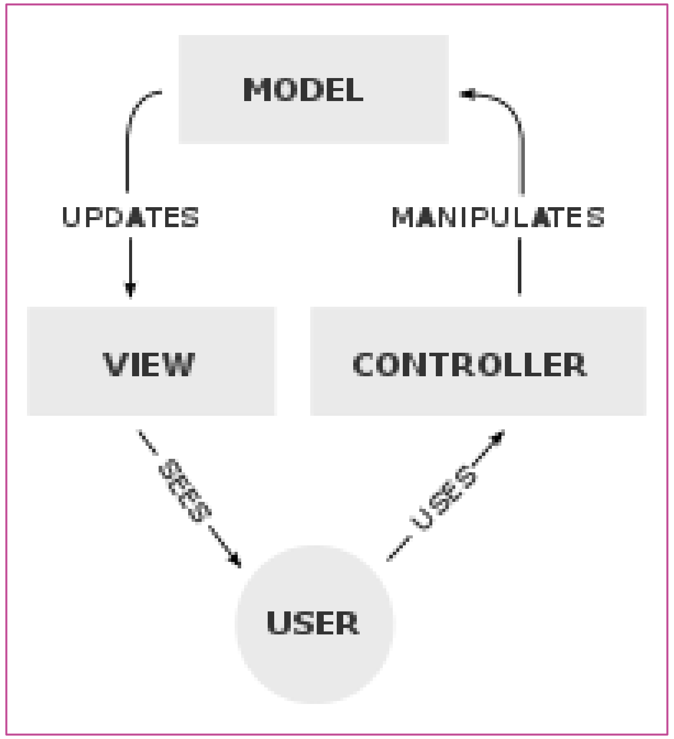
Components of MVC

- **Model**
  - The central component of the pattern. It is the application's dynamic data structure, independent of the user interface.[5] It directly manages the data, logic and rules of the application.
- **View**
  - Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- **Controller**
  - Accepts input and converts it to commands for the model or view.

- In addition to dividing the application into these components, the model–view–controller design defines the interactions between them

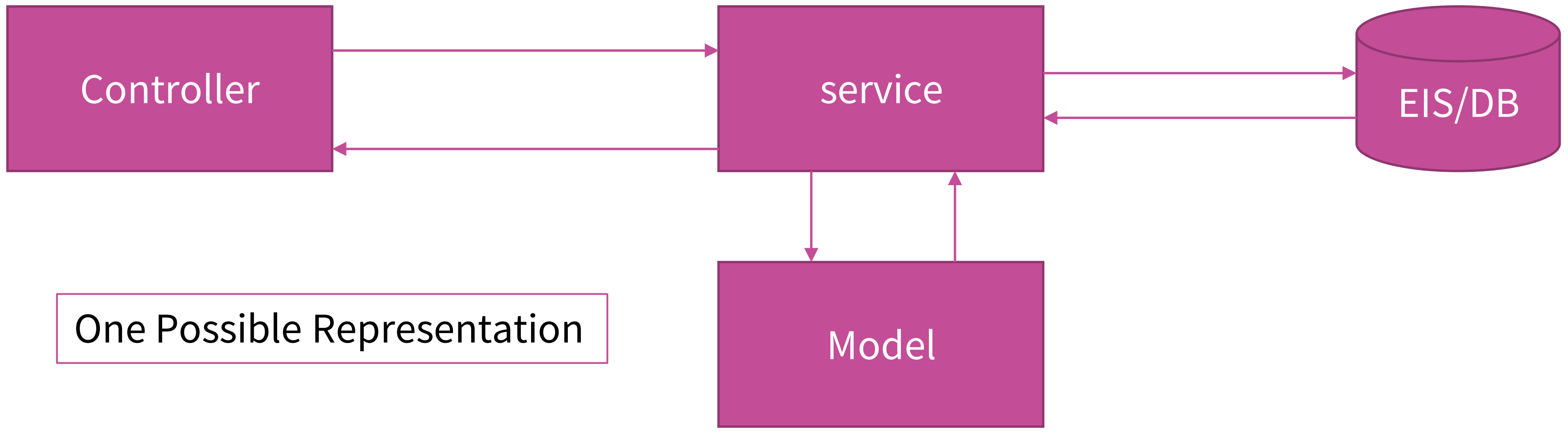
Interaction Between Components

- The model is responsible for managing the data of the application.
  - It receives user input from the controller.
- The view means presentation of the model in a particular format.
- The controller responds to the user input and performs interactions on the data model objects.
- The controller receives the input, optionally validates it and then passes the input to the model.



Service

- Between the controller and the model sometimes goes a layer which is called a service
- It fetches data from the model and lets the controller use the fetched data.
- This layer allows to separate data storage (model), data fetching (service) and data manipulation (controller).
- Since this layer is not part of the original MVC concept, it is optional in most cases but can be useful for code management and reusability purposes in some cases.



One Possible Representation

Advantages

- Simultaneous development
  - Multiple developers can work simultaneously on the model, controller and views.
- High cohesion
  - MVC enables logical grouping of related actions on a controller together.
  - The views for a specific model are also grouped together.
- Loose coupling
  - The very nature of the MVC framework is such that there is low coupling among models, views or controllers
- Ease of modification
  - Because of the separation of responsibilities, future development or modification is easier
- Multiple views for a model
  - Models can have multiple views
- Testability
  - with the clearer separation of concerns, each part can be better tested independently (e.g. exercising the model without having to stub the view)

Disadvantages

- Code navigability
  - The framework navigation can be complex because it introduces new layers of indirection and requires users to adapt to the decomposition criteria of MVC.
- Multi-artifact consistency
  - Decomposing a feature into three artifacts causes scattering. Thus, requiring developers to maintain the consistency of multiple representations at once.
- Undermined by inevitable clustering
  - Applications tend to have heavy interaction between what the user sees and what the user uses.
  - Therefore each feature's computation and state tends to get clustered into one of the 3 program parts, erasing the purported advantages of MVC.

Disadvantages

- Excessive boilerplate
  - Due to the application computation and state being typically clustered into one of the 3 parts, the other parts degenerate into either boilerplate shims or code-behind that exists only to satisfy the MVC pattern.
- Pronounced learning curve
  - Knowledge on multiple technologies becomes the norm.
  - Developers using MVC need to be skilled in multiple technologies.
- Lack of incremental benefit
  - UI applications are already factored into components, and achieve code reuse and independence via the component architecture, leaving no incremental benefit to MVC.

Demo Application

Let’s see a Demo

PATTERNS IN WEB APPLICATION DEVELOPMENT

LAYERED APPLICATION DESIGN

Layered Application Design

- Structure application in two layers
  - Interaction Layer and Processing Layer
- Interaction layer
  - Interface to clients
  - Receive requests and perform required translations and transformations
  - Delegate request to processing layer for processing
  - Respond to clients

Layered Application Design

- Processing layer
  - Process request by performing business logic
  - Access database
  - Integrate with EIS

Why Layered Application Design?

- Clearly divide responsibilities
  - De-couple business logic from presentation
  - Change in business logic layer does not affect the presentation layer and vice-versa
- Provide a common "place" for pre-processing and post-processing of requests and responses
  - logging, translations, transformations, etc.

MVC PATTERNS

Web Applications

Three Logical Layers in a Web Application: Model

- Model (Business process layer)
  - Models the data and behavior behind the business process
  - Responsible for actually doing Performing DB queries Calculating the business process Processing orders
  - Encapsulate of data and behavior which are independent of presentation

Three Logical Layers in a Web Application: View

- View (Presentation layer)
  - Display information according to client types
  - Display result of business logic (Model)
  - Not concerned with how the information was obtained, or from where (since that is the responsibility of Model)

Three Logical Layers in a Web Application: Controller

- Controller (Control layer)
  - Serves as the logical connection between the user's interaction and the business services on the back
  - Responsible for making decisions among multiple presentations
    - User's language, locale or access level dictates a different presentation.
  - A request enters the application through the control layer, it will decide how the request should be handled and what information should be returned

EVOLUTION OF WEB APPLICATION DESIGN ARCHITECTURE

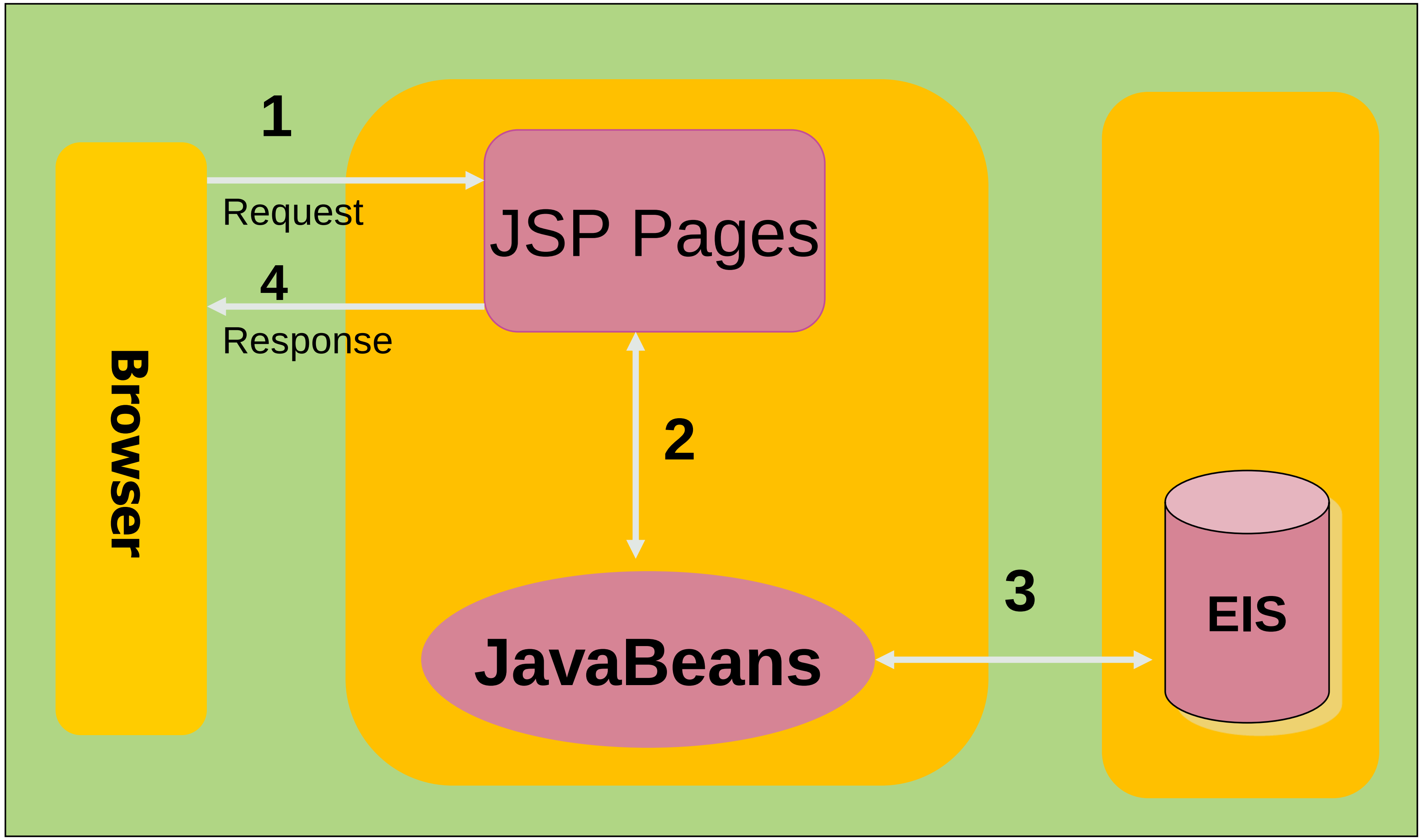
Evolution of MVC Architecture

- No MVC
- MVC Model 1 (Page-centric)
- MVC Model 2 (Servlet-centric)
- Web application frameworks

MODEL 1

(PAGE-CENTRIC ARCHITECTURE)

Mdel 1 Architecture



Page-centric Architecture

- Composed of a series of interrelated JSP pages
  - JSP pages handle all aspects of the application - presentation, control, and business process Business process logic and control decisions are hard coded inside JSP pages
  - in the form of JavaBeans, scriptlets, expression
- Next page selection is determined by
  - A user clicking on a hyper link, e.g. " <A HERF=find.jsp>
  - Through the action of submitting a form, e.g.

<FORM "" ACTION=search.jsp >

Page-centric: Simple Application

- One page might display a menu of options, another might provide a form for selecting items from the catalog,and another would be to complete shopping process
  - This doesn't mean we lose separation of presentation and content
  - Still use the dynamic nature of JSP and its support for JavaBeans component to factor out business logic from presentation
  - The pages are tightly coupled:
- Need to sync up request parameters
- Be aware of each other's URLs

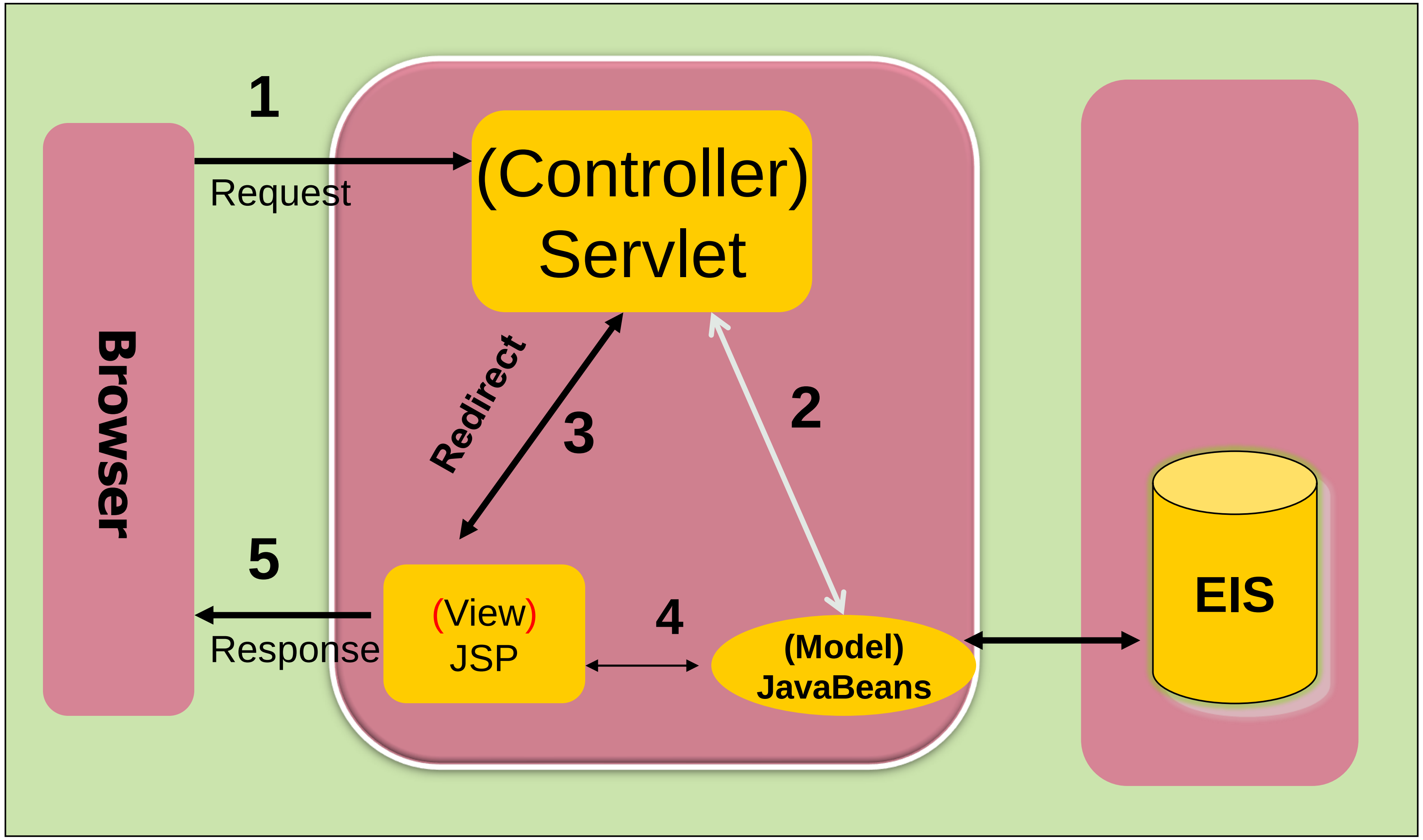
Page-centric: Component Page

- Create headers, footers and navigation bars in JSP pages
- Provides better flexibility and reusability.
- Easy to maintain.
- <%@ include file = header.jsp %>
- Use it when the file (included) changes rarely.
- Faster than jsp:include.
- <jsp:include page=header.jsp flush=true >
- Use it for content that changes often
- if which page to include can not be decided until the main page is requested.

MODEL 2

(SERVLET-CENTRIC ARCHITECTURE)

Model 2 Architecture



Why Model 2 Architecture?

- What if you want to present different JSP pages depending on the data you receive?
  - – JSP technology alone even with JavaBeans and custom tags (Model 1) cannot handle it well
- Solution
  - – Use Servlet and JSP together (Model 2)
  - – Servlet handles initial request, partially process the data, set up beans, then forward the results to one of a number of different JSP pages

Servlet-centric Architecture

- JSP pages are used only for presentation
  - – Control and application logic handled by a servlet (or set of servlets)
- ? Servlet serves as a gatekeeper
  - – Provides common services, such as authentication,authorization, login, error handling, and etc
- ? Servlet serves as a central controller
  - – Act as a state machine or an event dispatcher to decide upon the appropriate logic to handle the request
  - – Performs redirecting

How many Servlets in Servlet-centric Approach?

- It depends on the granularity of your application
  - One master Servlet
  - One servlet per use case or business function
  - Combination of the two
- master servlet handles common function (i.e. common login) for all business functions
- master servlet then delegates to child servlets for further gatekeeping tasks

When to Use
Model 1 or Model 2?

Model 1 (Page-centric)

- May encourage spaghetti JSP pages
  - Business logic may get lost in the display pages
- Use JavaBeans or custom tags that captures business logic (instead of scriptlets)
  - Page selection is done by each page
- JSPs are harder to debug than straight Java code:
  - Result in a failed compilation and a long list of useless compiler errors referring to the auto- generated code

Model 2 (Servlet-centric)

- Loosens the coupling between the pages and improves the abstraction between presentation and application logic
- Use JSPs for pure data display and input collection activities
- Most of the business logic can be debugged through the servlet before passed to JavaBeans and JSP

Best Practice Guideline

- Factor out the business logic into business objects and complex display logic into view objects
- Improves reusability, maintainability, unit testing and regression testing.

How Do I Decide?

- Use page-centric
  - If the application is simple enough that links from page to page.
- Use servlet-centric
  - Each link or button click requires a great deal of processing and decision-making about what should be displayed next.
- “How mapping between requests and responses are done” can help you to decide
  - Each request maps to one and only one response
- No need for controller.
  - Each request spawns a great deal of logic and a variety of different views can result
- A servlet is ideal

WEB APPLICATION FRAMEWORKS

Web Application Frameworks

- Based on MVC Model 2 architecture
- Web-tier applications share common set of functionality
  - Dispatching HTTP requests
  - Invoking model methods
  - Selecting and assembling views
- Provide classes and interfaces that can be used/extended by developers

Why Web Application Framework?

- De-coupling of presentation tier and business logic into separate components
- Provides a central point of control
- Provides rich set of features
- Facilitates unit-testing and maintenance
- Availability of compatible tools
- Provides stability
- Enjoys community-supports
- Simplifies internationalization
- Simplifies input validation

Why Web Application Framework?

- Frameworks have evolved with Java Server technology
- JSP/Servlets are still hard to use
- Frameworks define re-usable components to make this job easier.
- A good framework defines how components

  work to create a usable application.

Web Application Frameworks

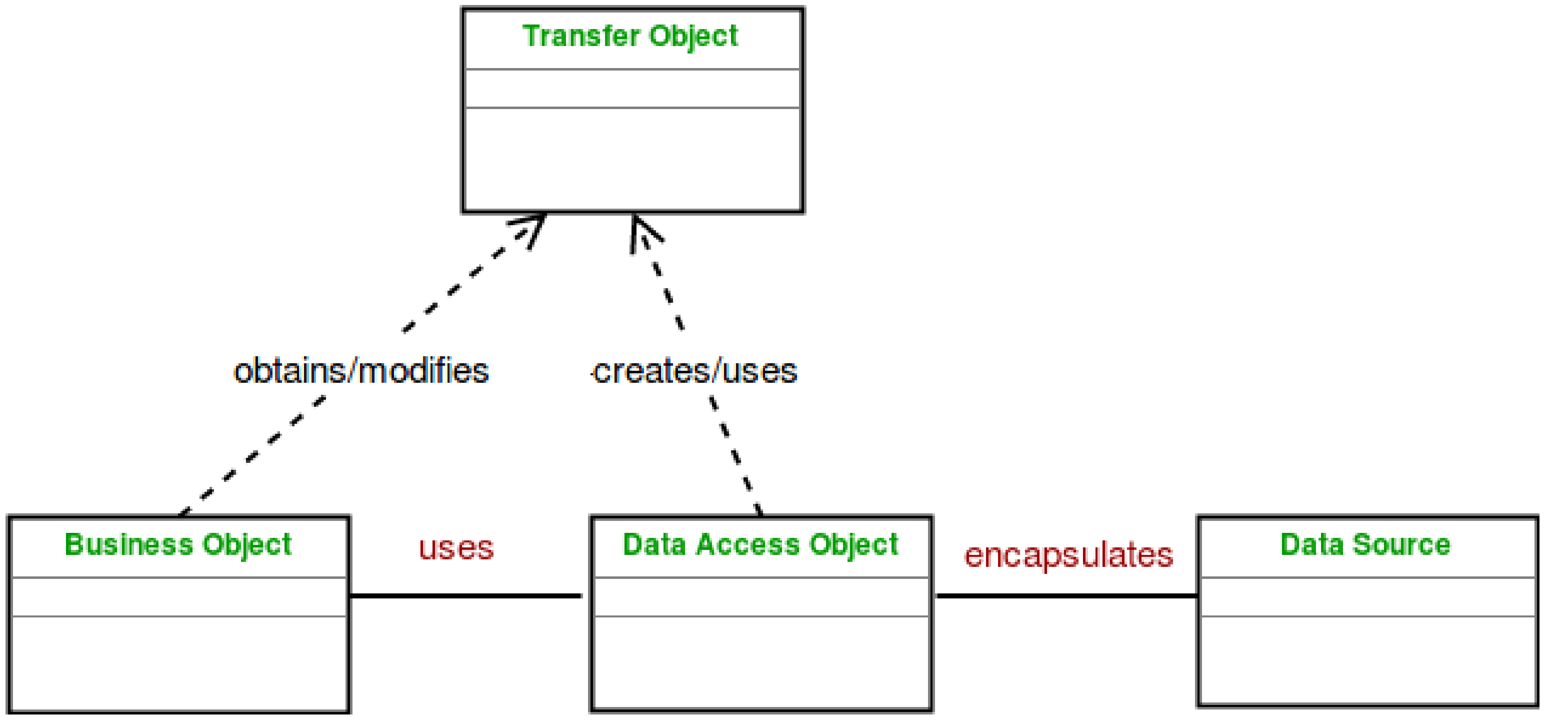
- Spring MVC
- Apache Struts
- JavaServer Faces
  - A server side user interface component framework for Java TM technology-based web applications
- Echo
- Tapestry

DAO

Data Access Object Pattern

Data Access Object Pattern

- Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services.
- Following are the participants in Data Access Object Pattern.



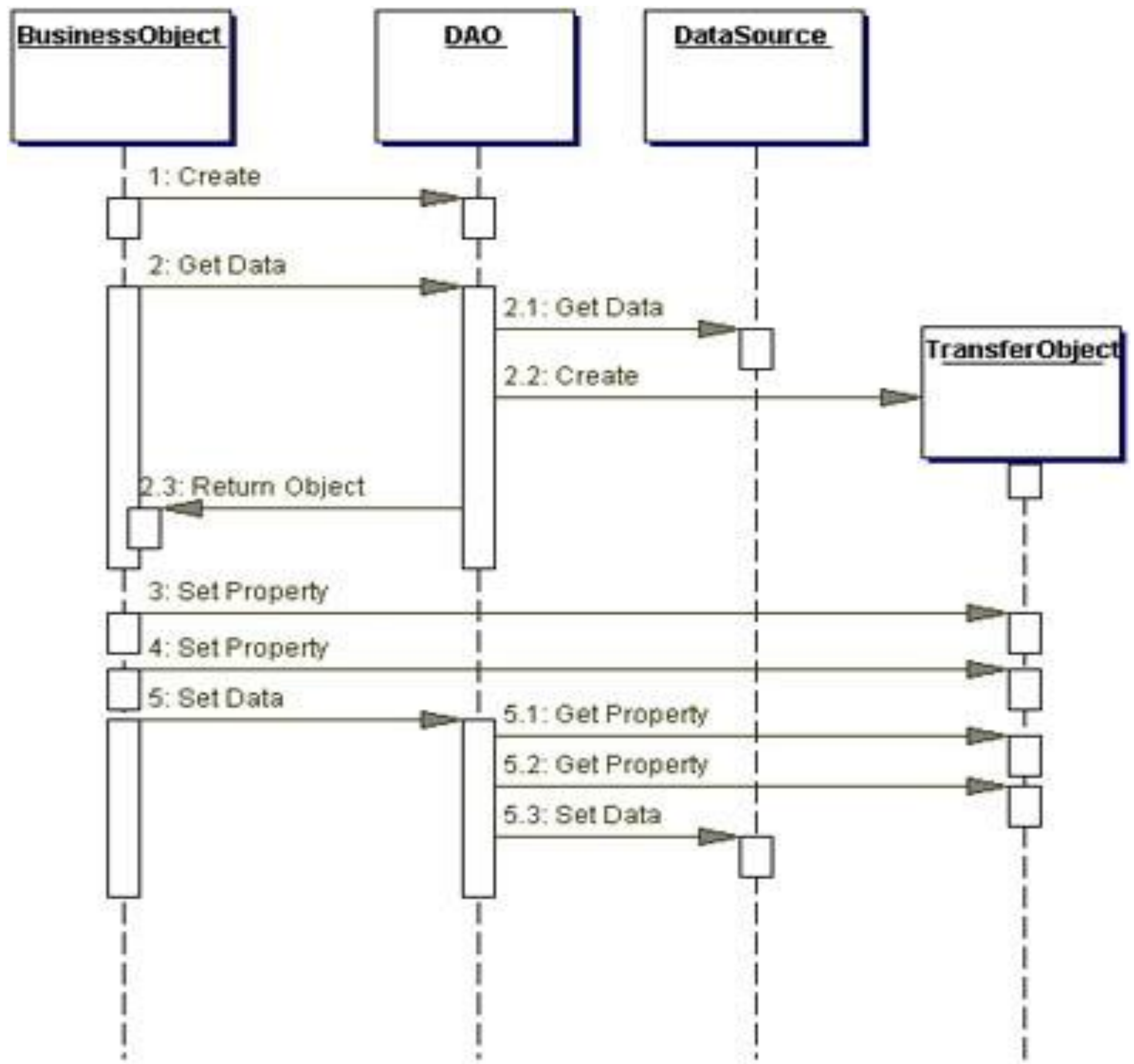
Components of DAO

- BusinessObject
  - The BusinessObject represents the data client.
  - It is the object that requires access to the data source to obtain and store data.
  - A BusinessObject may be implemented as a session bean, entity bean or some other Java object in addition to a servlet or helper bean that accesses the data source.
- DataAccessObject
  - The DataAccessObject is the primary object of this pattern.
  - The DataAccessObject abstracts the underlying data access implementation for the BusinessObject to enable transparent access to the data source.

Components of DAO

- DataSource
  - This represents a data source implementation.
  - A data source could be a database such as an RDBMS, OODBMS, XML repository, flat file system, and so forth.
  - A data source can also be another system service or some kind of repository.
- TransferObject
  - This represents a Transfer Object used as a data carrier.
  - The DataAccessObject may use a Transfer Object to return data to the client.
  - The DataAccessObject may also receive the data from the client in a Transfer Object to update the data in the data source.

DAO Interaction



Advantages

- The advantage of using data access objects is the relatively simple and rigorous separation between two important parts of an application that can but should not know anything of each other, and which can be expected to evolve frequently and independently.
- If we need to change the underlying persistence mechanism we only have to change the DAO layer, and not all the places in the domain logic where the DAO layer is used from.

Disadvantages

- Potential disadvantages of using DAO is leaky abstraction, code duplication, and abstraction inversion

Demo Application

Questions?