# LESSON 5 MAINTAINING STATE

Greater Success with Greater Breadth of Awareness

## Spring MVC Model

**ALL** [.NET, STRUTS,JSF] component based MVCs :

- Manage the model

  Gather, convert and validate request parameters

  Developer focuses on application/business function

  Model reflects state of user application

**SPRING MVC uses Model interface instead of HTTP Objects**

- Goal of Spring MVC framework:

  As view-agnostic as possible -- not bound to the HTTP

  Model is POJO

- **public interface Model** **Model Interface**

  Defines a holder for model attributes.

  Allows for accessing the overall model as a java.util.Map.

## JavaBeans .vs. POJO .vs. Spring Bean

- JavaBean
    - Adhere to Sun’s JavaBeans specification
    - Implements Serializable interface
    - Reusable Java classes for visual application composition

POJO

- ‘Fancy’ way to describe ordinary Java Objects
- Doesn’t require a framework
- Doesn’t require an application server environment
- Simpler, lightweight compared to ‘heavyweight’ EJBs

Spring Bean

- Spring managed - configured, instantiated and injected

A Java object can be a JavaBean, a POJO and a Spring bean all at the same time.

## Model Scoped Attributes

- Request scope
  - only be available for that request.
  - Thread Safe
- Session Scope
  - Session is defined by set of session scoped attributes.
  - Lifetime is a browser session.
  - **Sessions are a critical state management service provided by the web container**
- Context scope
  - Application level state
  - Lifetime is “usually” defined by deployment of application
  - Attributes available to every controller and request in the application

## Managing state information

- How to handle the different scopes of model information :
- Request scope: short term computed results to pass from one servlet to another (i.e., “forward”)
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - request.setAtttribute(key,value)
  - **model.addAttribute(key,value)**
- Session scope: conversational state info across a series of sequential requests from a particular user
  - HttpSession session = request.getSession(); session.setAttribute(key,value)
  - **@SessionAttributes - model.addAttribute(key,value)**
- Application/context scope: global info available to all controllers in this application
  - request.getServletContext(). getAttribute(String name)
  - **XML configuration OR @Autowired ServletContext servletContext;**

## Request Scope Attribute

```
 public String getForward (Model model) {
     model.addAttribute("requestAttribute","requestAttribute");
     // Should see RequestAttribute on session.jsp
    return "session";
 }
 public String redirect (Model model ) {
     // This is a request parameter shouldn't see it on redirect
     model.addAttribute("requestAttribute","requestAttribute");
    return "redirect:/get_redirect;
 }
 @RequestMapping(value="/get_redirect" )
 public String getRedirect  (...) {
     return "session";
 }
```

**session.jsp**

```
  <!--Should NOT see the request attribute if from redirect-->
 requestAttribute is:   ${requestAttribute}<br>
```

## @SessionAttributes

Class level annotation that indicates an object is to be **added/retrieved** from Session

**Add to Model:**

```
• @Controller
• @SessionAttributes(“Leonardo")
• public class ProductController {
• @RequestMapping(value={"/","/product_input"},method= RequestMethod.GET)
•   public String inputProduct(Model model){
        Product product = new Product();
        product.setName(“Leonardo Turtle");
        model.addAttribute( “Leonardo",product);
```

**Retrieve from Model:**

```
public String saveProduct(Product newProduct, Model model,
                          SessionStatus status) {
Product product = (Product)( ((ModelMap) model).get(" Leonardo") );
```

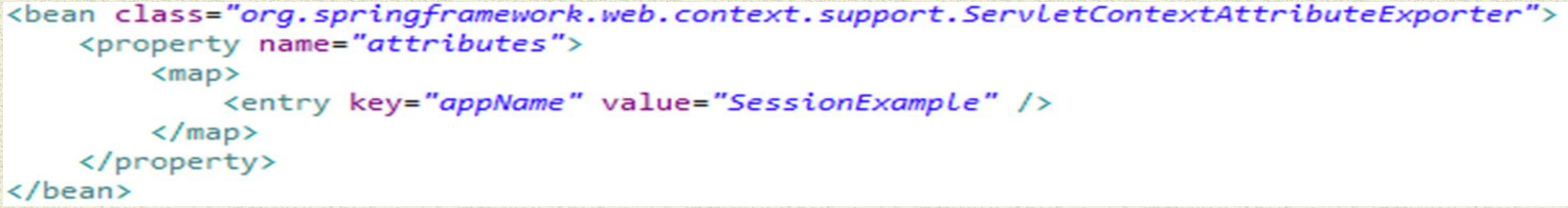
**Remove @SessionAttributes**

```
   status.setComplete();
NOTE: Will also use request.getSession.setAttribute() in Demo
```

## Application level Attributes

- ServletContext contains Application level state  information

- XML configuration:



```
<bean class="org.springframework.web.context.support.ServletContextAttributeExporter">
    <property name="attributes">
        <map>
            <entry key="appName" value="SessionExample" />
        </map>
    </property>
</bean>
```

- Programmatic access:
- @Autowired
- ServletContext servletContext;
-  servletContext.getAttribute("appName");

## Main Point

State information can be stored in successively broader application levels: request, session, and application.

*Deeper levels of consciousness are broader in scope.*

.

## Request GET versus POST

Difference between GET and POST:

- GET request has no message body, so parameters are limited to what can fit into Query String.

  GET /advisor/selectBreadTaste.do?color=dark&taste=salty
- GET requests are *idempotent*
- GET is to retrieve data

***Idempotent means that multiple calls with the same operation doesn't change the server***

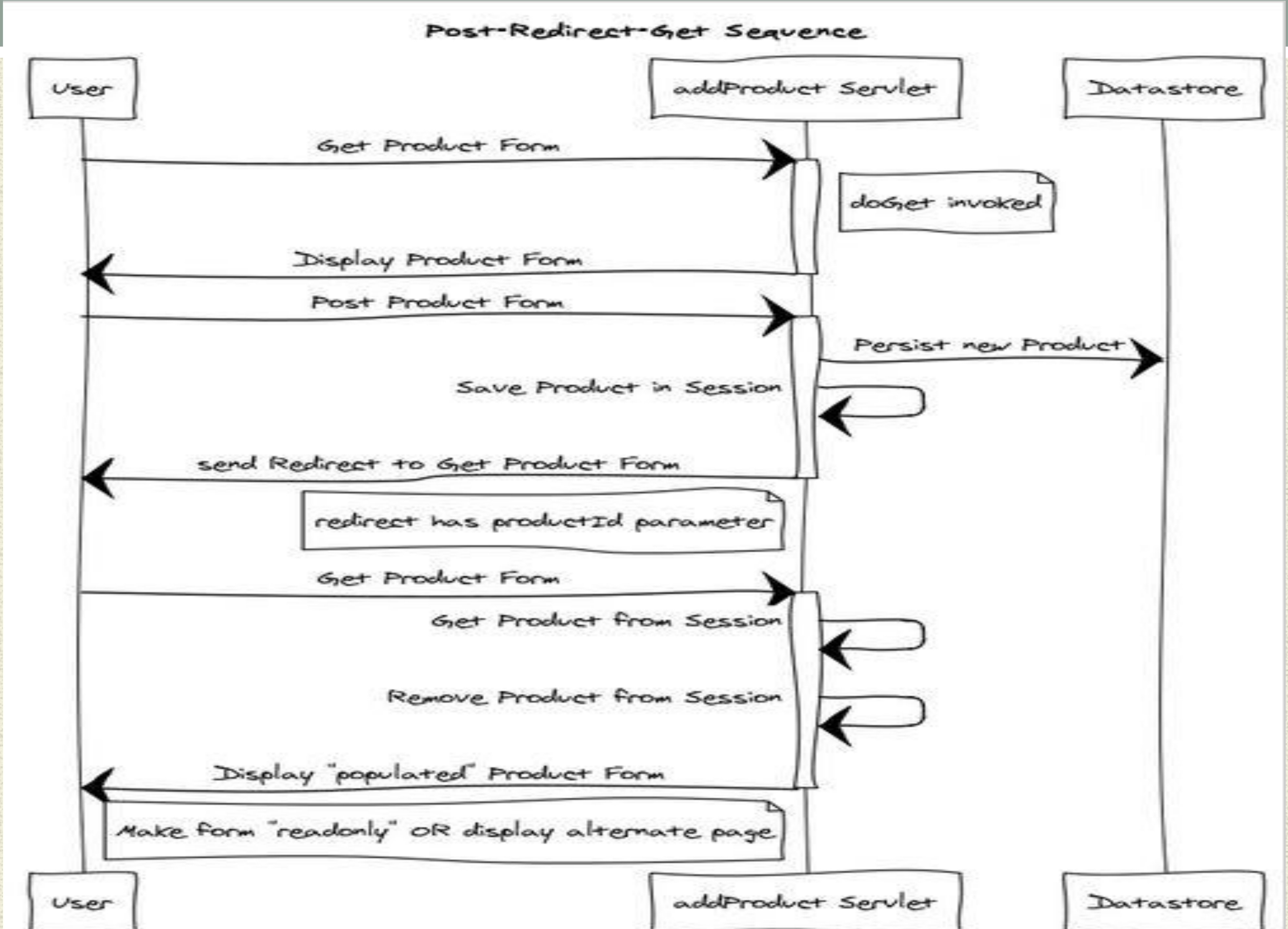
- POST is to send data to be processed and stored
- POST has a body
- POST "more secure" since parameters not visible in browser bar

## Post/Redirect/Get (PRG) Pattern

- POST-REDIRECT-GET, or the PRG pattern for short. The rules of the pattern are as follows:
- Never show pages in response to POST
- Always load pages using GET
- Navigate from POST to GET using REDIRECT

Forward – if operation can be safely repeated upon a browser reload of the resulting web page [Use with GET].

- Redirect -  If operation performs an edit on the datastore,  to avoid the possibility of inadvertently duplicating an edit to the database[Use with POST].


Post-Redirect-Get Sequence
User
addProduct Servlet
Datastore
Get Product Form
doGet invoked
Display Product Form
Post Product Form
Persist new Product
Save Product in Session
send Redirect to Get Product Form
redirect has productId parameter
Get Product Form
Get Product from Session
Remove Product from Session
Display "populated" Product Form
Make form "readonly" OR display alternate page
User
addProduct Servlet
Datastore

# Spring MVC Forward & Redirect

Work Just like JSP Forward & Redirect

SYNTAX:

```
return “forward:/demo";
return "redirect:/demo";
```

**REDIRECT NOTE:** Attributes that are primitive types are automatically appended as query parameters.

WHERE:

```
@RequestMapping(value="/demo" )
public String getDemo (Model model ) {
```

Disable with:
<mvc:annotation-driven ignore-default-model-on-redirect="true" />

EXTERNAL REDIRECT:

```
return "redirect:http://www.mum.edu";
```

## Flash Attributes

- Efficient solution for the *Post/Redirect/Get* pattern.
-
- `public String saveProduct(Product newProduct, Model model, RedirectAttributes redirectAttributes,`
- `redirectAttributes.addFlashAttribute( newProduct);`
- Attributes are saved [ in Session] temporarily before the redirect
- Attributes are added to the Model of the target controller and are deleted [from Session] immediately.
- `redirectAttributes.addAttribute( “name”,newProduct.name);`
- String & primitive types are added to URL [e.g., GET]

## Main Point

- Understanding the function and capability of the POST, Redirect and GET, leads to a combination[PRG] that overcomes a weakness [page refresh on POST] in web applications.

*The development of consciousness, increases awareness and eliminates restrictions that cause weakness*

## Web Conversational flow

- A web conversational flow involves a series of multiple screens that work as a unit. During the conversation, state is maintained across the entire conversation

- @SessionAttributes  facilitates conversational flows
- Flash Attributes are also a building  block of conversational flows

**HOWEVER**

- As the flow get more complex

State Management get more difficult

## Spring Web Flow
## User Conversational Flows

- A Spring Web Flow is a blueprint for user conversations that drive a business process.

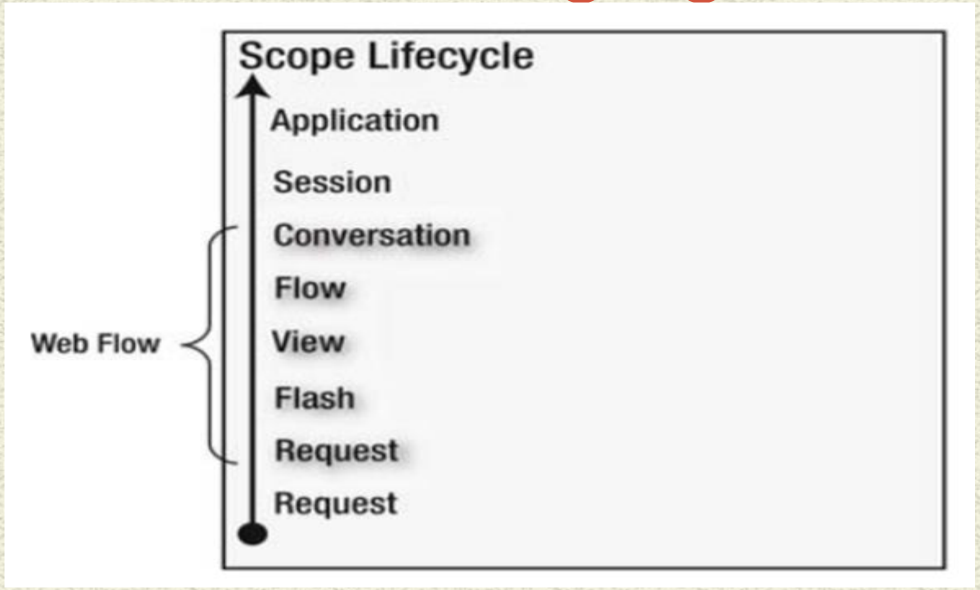
- Fundamental capability is Managing Data in the back & forth process involving multiple input screens.

**Automates State Management**

**And Introduces**

**Additional Scopes [beyond Request,Session & Application]**

# SWF Additional Scopes for Managing Data



Stored in Session
**Managed by SWF**
**[Creation & Deletion]**

- **Conversation** - The conversation scope starts when a flow starts and ends when the flow ends. **It is available in sub flows**
- **Flow -** Available within a flow. **Not available in sub flows**
- **Request -** Available during the life of a request in a flow
- **Flash -** Available during the lifetime of a flow. However, once a state is rendered, the variable is cleared.
- **View -** Available only during the lifetime of a view. Created when a view is created and destroyed once a view is destroyed

## CONTROLLER METHOD ARGUMENTS

| | |
|---|---|
| Map Model/ModelMap | |
| Command/form object [ optional @ModelAttribute] | |
| RedirectAttributes | |
| SessionStatus | |
| BindingResult | Validation |
| @RequestParam | |
| @RequestBody | RESTful Services |
| @ResponseBody | RESTful Services |
| @PathVariable | Template |
| HttpServletRequest HttpServletResponse HttpSession | |
| @RequestHeader | |

## Controller Method Return Types

1. **ModelAndView** object,
2. **Model** object, with the view name implicitly determined through a **RequestToViewNameTranslator**
3. **Map** object for exposing a model, the view name implicitly determined through a **RequestToViewNameTranslator**
4. **String** value interpreted as the logical view name, the model implicitly determined through command objects
5. **void** if the method handles the response itself (by writing the response content directly, declaring an argument of type ServletResponse / HttpServletResponse for that purpose) or if the view name is supposed to be implicitly determined through a **RequestToViewNameTranslator**

**RequestToViewNameTranslator** – basically uses the URL from the @RequestMapping

## More Model, ModelMap, ModelAndView

- Model is an interface while ModelMap is a class.
- Model has method asMap to get actual map.
- ModelMap is a class that is a custom[convenience] Map implementation that automatically generates a key for an object when an object is added to it.
- ModelAndView is just a container for both a ModelMap and a view object. It allows a controller to return both as a single value.

## Main point

Spring MVC is “Open for extension, closed for modification”.

As a result, Spring provides a wide range of opportunities to change the behavior of an application based on the framework.

*Likewise, Pure Consciousness offers a wide range of possibilities. They both represent good design.*