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 contains POJO objects that reflect state of app
  - **SPRING MVC uses Model interface instead of HTTP Objects**
- Goal of Spring MVC framework
  - As view-agnostic as possible - not bound to the HTTP
- **public interface Model**
  - Defines a holder for model attributes.
  - Allows for accessing the overall model as a java.util.Map.

JavaBean vs POJO vs Spring Bean

- JavaBean
  - Adhere to Sun's JavaBeans specification
  - Implements Serializable interface
  - Must have default constructor, setters&getters
  - 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

- JSP page scope
  - The page scope restricts the scope and lifetime of attributes to the same page where it was created.
- 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")

  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("appName")
  ```

  - **OR**

  ```
  @Autowired
  ServletContext servletContext;

  servletContext.getAttribute("appName")
  ```

Request Scope Attribute

```
@RequestMapping(value = "/forward")
public String forward(Product product, Model model) {
   product.setDescription("Request Attribute Exists!!");
   model.addAttribute("requestAttribute", product);
   model.addAttribute("redirectParamTest", "Request Parameter EXISTS!");
   return "forward:/get_forward";
}

@RequestMapping(value = "/get_forward")
public String getForward(Model model) {
   return "ForwardRedirect";
}
```

- ForwardRedirect.jsp

```
<h4>${redirectParamTest}</h4>
<h4>${requestAttribute.description}</h4>
```

- Demo: ProductSessionExample – Forward

@SessionAttributes

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

```
@Controller
@SessionAttributes({ "Leonardo", "Splinter" })
public class SessionController {

    @RequestMapping(value = { "/getSession" }, method = RequestMethod.GET)
    public String inputProduct(Model model, HttpSession session) {
        Product product = new Product();
        product.setName("Leonardo Turtle");
        model.addAttribute("Leonardo", product);
        model.addAttribute("Splinter", "Splinter");
        // add Regular attribute
        session.setAttribute("Donatello", "Donatello Turtle");
        return "SessionForm";
    }
}
```

- Retrieve from Model

```
Product product = (Product) model.asMap().get("Leonardo");
```

- Used to mark a session attribute as not needed *after* the request has been processed by the controller

```
status.setComplete();
```


See demo: ProductSessionExample - Session

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="State Management Demo" />
        </map>
    </property>
</bean>
```

- Programmatic access:

```
@Autowired
ServletContext servletContext;

servletContext.getAttribute("appName");
```

Main Point

- State information can be stored in request, session, or context/application scope and also as hidden fields or cookies.
- *Deeper levels of consciousness are more powerful and have broader scope.*

Static Resources

- Want to handle static content, e.g., image file, js, css, etc.
- Need to identify them to the DispatcherServlet - since no Controller exists for serving static resources.
- Using Spring:
  - Declare resources folder[s]
  - Serve static content from there
  - Use mvc:resources – A Spring help element to map "url path" to a physical file path location.
- All references to `/resource/` will be mapped to the context root (webapp): `/css/` folder.

```
<mvc:resources mapping="/resource/**" location="/css/"/>
```

- Alternative: serves content from servlet containers
  - If we are using DefaultServletHttpRequestHandler, then we can replace :

```
<mvc:resources mapping="/js/**" location="/js/"/>
<mvc:resources mapping="/css/**" location="/css/"/>
<mvc:resources mapping="/images/**" location="/images/"/>
```

  - with :

```
<mvc:default-servlet-handler />
```

- path pattern - Apache ant
  - ('*') matches zero or more characters, up to the occurrence of a '/'.
  - ('**') matches zero or more characters. This could include the path separator '/'.

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

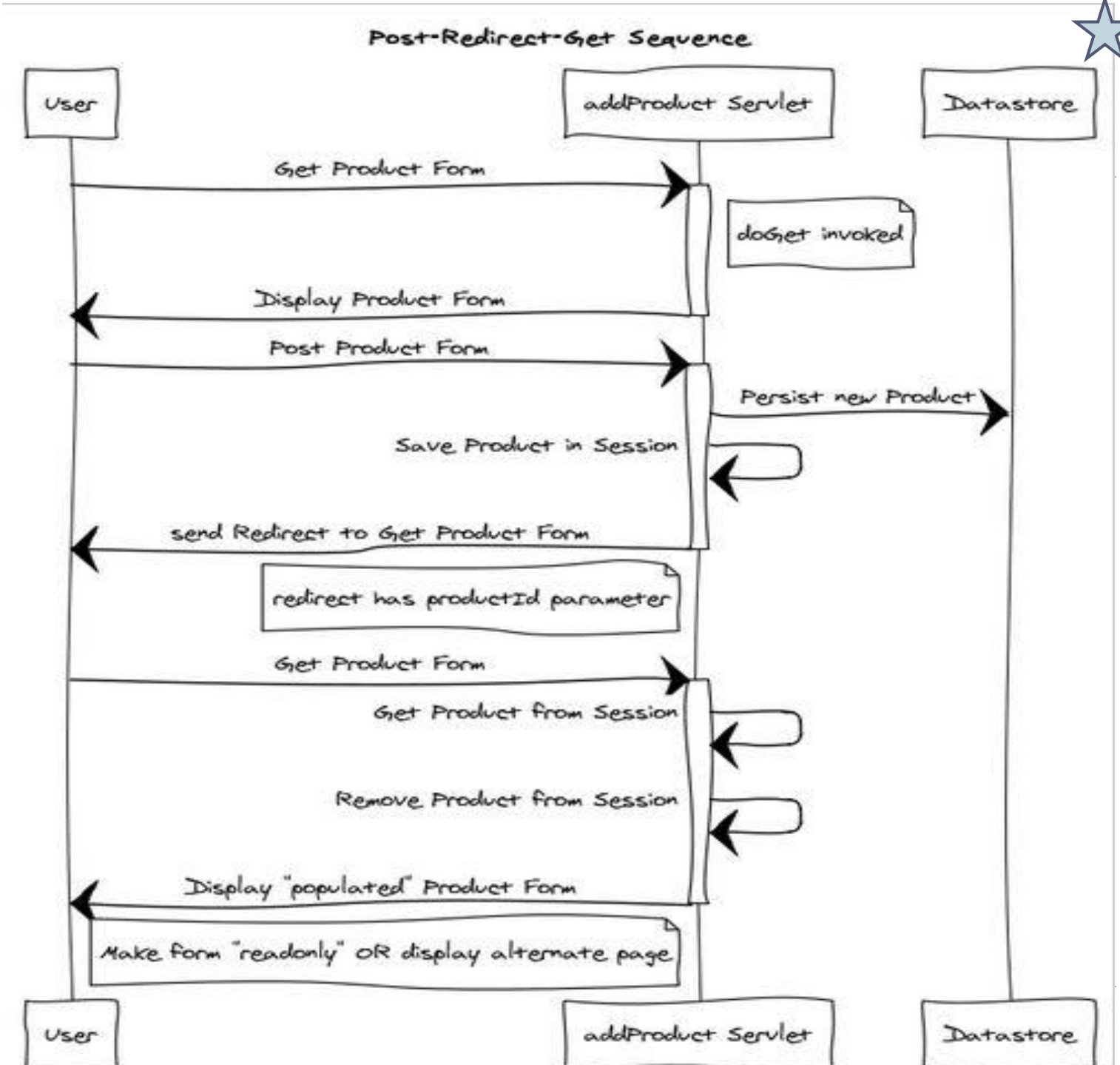
- POST is to send data to be processed and stored
- POST has a body
- POST "more secure" since parameters not visible in browser bar

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";
```

- WHERE:

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

- EXTERNAL REDIRECT:

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

See demo: ProductSessionExample - Redirect

Flash Attributes

- Efficient solution for the *Post/Redirect/Get* pattern.
- 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.

```
@RequestMapping(value = "/product", method = RequestMethod.POST)
public String saveProduct(Product newProduct, Model model,
   RedirectAttributes redirectAttributes,
    HttpServletRequest request) {

   redirectAttributes.addFlashAttribute(newProduct);
   // Returned as a parameter on GET URL
   redirectAttributes.addAttribute("name", "Kemosabe");
   return "redirect:/details";
}
```

- String & primitive types are added to URL [e.g., GET]

```
redirectAttributes.addAttribute(newProduct.name);
```

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

- Understanding the function and capability of the POST, Redirect and GET, leads to a combination that overcomes an inherent weakness in web applications.
- *The development of consciousness, increases awareness and eliminates the restrictions that cause inherent weakness.*