LESSON 5
MAINTAINING STATE

Greater Success with Greater Breadth of Awareness

Spring MVC Model

ALL 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.

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.getAtttribute(String name) [or setAttribute]
  - **model.getAttribute**
- Session scope: conversational state info across a series of sequential requests from a particular user
  - HttpSession session = request.getSession().getAttribute(String name)
  - **@SessionAttributes**
- 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 see request.getSession.setAttribute() example 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");

HTML Hidden Fields

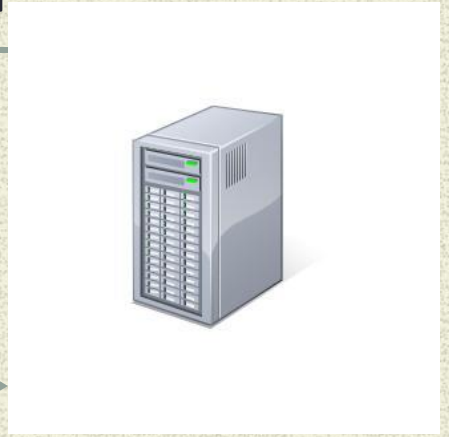
- A special type of field in a web form
- As the name indicates, this field is transparent to the user and sent with the request as any other parameter of the form
- Values can only be String and not arbitrary objects
- Allows sensitive session information to be encoded in the response
-

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.*

Keeping Track of Session: cookie exchange

http/1.1 200 OK
Date: 2/29/2012
Set-Cookie: JSESSIONID=0XA34G108
…



POST /select/selectTea HTTP/1.1
Host: www.cs.mum.edu
Cookie: JSESSIONID=0XA34G108
…

Session tracking via cookie exchange [Cont]

- A web conversation, holds information across multiple requests.
- Before container sends back to browser, it saves the session info, and then sends an HTTP “cookie” with session id back to the browser
  - Set-cookie header
- Browser stores the info and puts cookie/sessionid back into a header with the next request
  - Cookie header
- Container then needs to reconstitute session from storage before calling servlet with subsequent requests in the “conversation”

Session lifetime

Client side

- Browser discards all “temporary” cookies when it closes
- Every tab or window of browser will have access to all cookies

Server side

- How to get a session
  - session = request.getSession(); //creates new session if none exists
    - **session.isNew(); //checks whether is a new session**
    - **request.getSession(false); //returns null if none exists**
- How to get rid of the session
  - sessions can become a memory resource issue
  - container can’t tell when browser is finished with session
  - 3 ways for container to remove sessions
    - **session timeout in the DD**
    - **session.setMaxInactiveInterval(20*60); //seconds**
    - **session.invalidate(); //immediate**

```
<session-config>
        <session-timeout>
          30
        </session-timeout>
    </session-config>
</web-app>
```

(HTTP) Cookies

- Can be used for other things besides implementing sessions
  - Temporary cookie
    - browser removes when it closes
    - this is default
    - session cookies are like this
  - permanent cookie
    - a cookie that has a max age set

- Sending a Cookie

```
Cookie cookie = new Cookie("Name", "Jack");
cookie.setMaxAge(minutes);
response.addCookie(cookie);
```

- Reading a cookie
  - Cookies come with the request
  - Can only get all cookies[for site], then search for the one you want.

```
for (Cookie cookie : request.getCookies()) {
if (cookie.getName().equals("Name")) {
String userName= cookie.getValue();  } }
```

Static Resources

- Want to handle static content, e.g., image file, js, css, etc.
- Circumvent Controller mechanism since no dynamic content
- Best practice:
  - 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.

- `<mvc:resources  location="/resources/"  mapping="/resource/**"/>`
- All references to /resource/ will be mapped to the Contextroot:/resources/ folder

Main point

Session Tracking is an important part of session management. It provides basic continuity to a web application. *At the level of the unified field there is a continuous frictionless flow of information.*

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

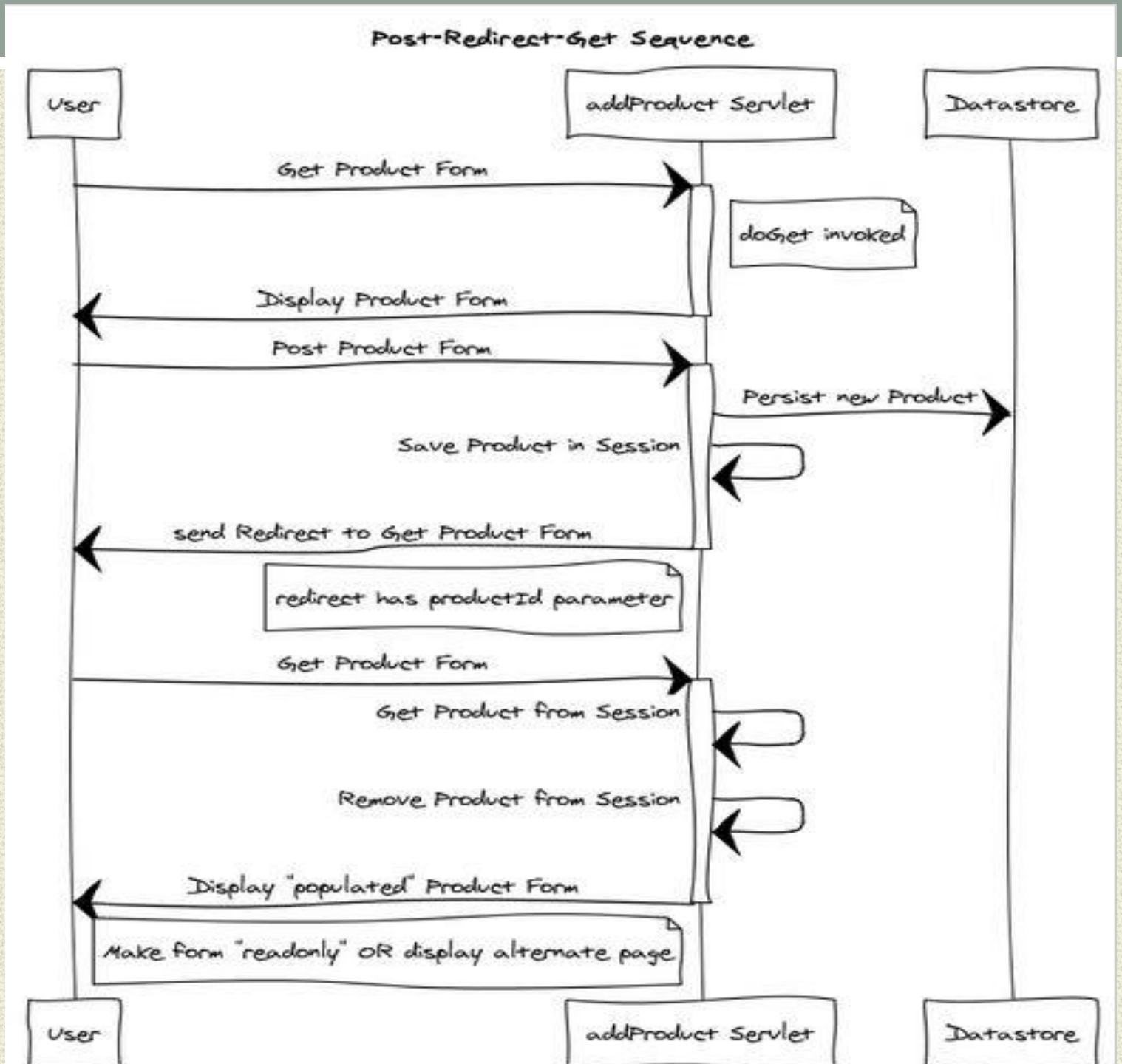
- POST is to send data to be processed and stored
- POST has a body
- POST more secure since parameters not visible in browser bar

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

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( 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 that overcomes an inherent weakness in web applications. *The development of consciousness, increases awareness and eliminates the restrictions that cause inherent weakness*

Domain Model Objects

- Serialization
  - Implement Serializable interface when
    - Use in Web Service
    - Need to Persist object
  - Serializable objects have an identifier: **serialVersionUID**.
    If class is updated - a new identifier is auto-generated
    To control versioning, manually add **serialVersionUID**
    IDE's can generate serialVersionUID
  - Serializable interface has no methods or fields and serves only to identify the semantics of being serializable.
  - transient [keyword] marks a field as: not be saved.