**1. MVC: (pg 12)**

The high-level design of Struts 2 follows the well-established Model-View-Controller design pattern. The MVC pattern provides a separation of concerns that applies well to web applications. Separation of concerns allows us to manage the complexity of large software systems by dividing them into high-level components. The MVC design pattern identifies three distinct concerns: model, view, and controller.

CONTROLLER-FILTERDISPATCHER

The MVC variant used in Struts is often referred to as a front controller MVC. This means that the controller is out front and is the first component to act in the processing. The controller’s job is to map requests to actions. In a web application, the incoming HTTP requests can be thought of as commands that the user issues to the application. One of the fundamental tasks of a web application is routing these requests to the appropriate set of actions that should be taken within the application itself. The role of the controller is played by the Struts 2 FilterDispatcher. This important object is a servlet filter that inspects each incoming request to determine which Struts 2 action should handle the request. The framework handles all of the controller work for you. You just need to inform the framework which request URLs map to which of your actions.

MODEL—ACTION

The controller, after receiving the request, must consult its mappings and determine which of these actions should handle the request. Once it finds the appropriate action, the controller hands over control of the request processing to the action by invoking it. This invocation process, conducted by the framework, will both prepare the necessary data and execute the action’s business logic. When the action completes its work, it’ll be time to render a view back to the user who submitted the request. Toward this end, an action, upon completing its work, will forward the result to the Struts 2 view component.

VIEW—RESULT  
The view is the presentation component of the MVC pattern. The result returns the page to the web browser. This page is the user interface that presents a representation of the application’s state to the user. These are commonly JSP pages, Velocity templates, or some other presentation-layer technology. While there are many choices for the view, the role of the view is clear-cut: it translates the state of the application into a visual presentation with which the user can interact. Good MVC lays the groundwork for easily managing the most complex front end.

**2. How Struts 2 works (pg 15)**

Struts 2 provides a cleaner implementation of MVC. These clean lines are only possible with the help of a few other key architectural components that participate in processing every request. Chief among these are the interceptors, OGNL, and the ValueStack. In the figure, the FilterDispatcher has already done its controller work by selecting the appropriate action to handle the request. The figure demonstrates what really happens when the action is invoked by the controller.

INTERCEPTORS

The interceptors are invoked both before and after the action, though we should note that they actually fire after the result has executed. Interceptors don’t necessarily have to do something both times they fire, but they do have the opportunity. Some interceptors only do work before the action has been executed, and others only do work afterward. The important thing is that the interceptor allows common, cross-cutting tasks to be defined in clean, reusable components that you can keep separate from your action code.

THE VALUESTACK AND OGNL  
The ValueStack is a storage area that holds all of the data associated with the processing of a request. You could think of it as a piece of scratch paper where the framework does its work while solving the problems of request processing. Rather than passing the data around, Struts 2 keeps it in a convenient, central location—the ValueStack.

OGNL stands for the Object-Graph Navigation Language. OGNL is the tool that allows us to access the data we put in that central repository. More specifically, it is an expression language that allows you to reference and manipulate the data on the ValueStack.

The tricky, and powerful, thing about the ValueStack and OGNL is that they don’t belong to any of the individual framework components. Note that both interceptors and results can use OGNL to target values on the ValueStack. The data in the ValueStack follows the request processing through all phases, it slices through the whole length of the framework. It can do this because it is stored in a ThreadLocal context called the ActionContext.

The ActionContext contains all of the data that makes up the context in which an action occurs. This includes the ValueStack but also includes stuff the framework itself will use internally, such as the request, session, and application maps from the Servlet API. The use of ThreadLocal makes the ActionContext, and thus the ValueStack, accessible from anywhere in the same thread of execution.

**3) OGNL (pg 103)**

OGNL is a powerful technology that’s been integrated into the Struts 2 framework to help with data transfer and type conversion. OGNL is the glue between the framework’s string-based HTTP input and output and the Java-based internal processing. OGNL consists of two things: an expression language and type converters.  
EXPRESSION LANGUAGE

We use OGNL’s expression language in our form input field names and JSP tags. In both places, we’ve been using OGNL expressions to bind Java-side data properties to strings in the text-based view layers, commonly found in the name attributes of form input fields, or in various attributes of the Struts 2 tags.

Struts 2 tag using the OGNL expression language :  
<h5>Congratulations! You have created </h5>  
<h3>The <s:property value="portfolioName" /> Portfolio</h3>  
The OGNL expression language is the bit inside the double quotes of the value attribute. This Struts 2 property tag takes a value from a property on one of our Java objects and writes it into the HTML in place of the tag. This is the point of expression languages. They allow us to use a simplified syntax to reference objects that reside in the Java environment. It even supports advanced features such as invoking method calls on the Java objects that it can access, but the whole idea of an expression language is to simplify access to data.

OGNL creates the pathways for data to flow through the framework. It helps move data from the request parameters onto our action’s JavaBeans properties, and it helps move data from those properties out into rendering HTML pages.