Sun Certified Enterprise Architect for Java EE Study Guide

SECOND EDITION

Mark Cade ▪ Humphrey Sheil

Library of Congress Cataloging-in-Publication Data:

Cade, Mark.

Sun Certified Enterprise Architect for Java EE study guide / Mark Cade, Humphrey Sheil. — 2nd ed.

p. cm.

Previous ed.: Sun Certified Enterprise Architect for J2EE technology study guide, 2002.

ISBN 978-0-13-148203-6 (pbk. : alk. paper) 1. Electronic data processing personnel—Certification. 2. Java (Computer program language)—Examinations—Study guides. I. Sheil, Humphrey. II. Cade, Mark. Sun Certified Enterprise Architect for J2EE technology study guide. III. Title.

QA76.3.C23 2010

005.13'3—dc22

2009052010

ISBN-13: 978-0-13-148203-6
ISBN-10: 0-13-148203-3

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana
First printing February 2010

CHAPTER 3

Web Tier Technologies

- State the benefits and drawbacks of adopting a web framework in designing a Java EE application.
- Explain standard uses for JSPs and Servlets in a typical Java EE application.
- Explain standard uses for JSF components in a typical Java EE application.
- Given a system requirements definition, explain and justify your rationale for choosing a web-centric or EJB-centric implementation to solve the requirements. Web-centric means that you are providing a solution that does not use EJBs. An EJB-centric solution will require an application server that supports EJBs.

Introduction

This chapter covers the area of presentation in the JEE platform, and focuses on presentation technologies designed to render in a standards-compliant HTML browser. In addition to focusing on the presentation specifications and technologies that are included in the JEE platform, we go one step further and analyze the benefits and drawbacks of using a web framework to lend additional structure to a web application (whether it also uses EJB or not) at the expense of additional complexity or runtime overhead.

Prerequisite Review

The following list details resources and specifications that you should be familiar with before reading this chapter. The main resources are as follows:

- **The JavaServer Pages 2.1 specification**—JSR 245
- **The Servlet 2.5 specification**—JSR 154
- **The JSF 1.2 specification**—JSR 252
- **The JSTL 1.2 specification**—JSR 52
- **The Java EE 5 specification**—JSR 244

We now cover the specific topics that should be addressed at a high level before more esoteric and advanced discussions on the relative advantages and disadvantages of the various JEE web tier technologies.

Model View Controller (MVC)

Regardless of application domain or industry vertical, technology platform, and client-side technology, everyone agrees that three fundamental concepts should be decoupled and kept separate—namely, the data, the business logic that operates on that data, and the presentation of that data to the end user (see Figure 3-1). In the JEE platform, the seminal design pattern that enforces this separation of concerns is called the MVC model. In the earliest releases of the specification, references were made to Model 1 and Model 2 architectures. However, all mainstream frameworks now embrace Model 2 exclusively—where views (implemented as JSP pages with or without JSF components) forward to a central controller (implemented as a Servlet), which invokes a named handler for the page or action before forwarding the user to a well-defined page to render the outcome of the request.

Web Container

The web container is analogous to the EJB container described in Chapter 4, "Business Tier Technologies." Simply put, one of the biggest advantages of the JEE platform is how much it gives the developer out of the box from an infrastructure perspective, leaving the developer free to focus on how to use the JEE platform to implement the required business logic for their application. A web container provides services to presentation and control components provided by the developer, implemented as JSPs, JSF components, Servlets, filters, web event listeners, and plain old Java classes (POJOs). These services include concurrency control, access to user-managed transactions (more on this later), configuration, and security management.

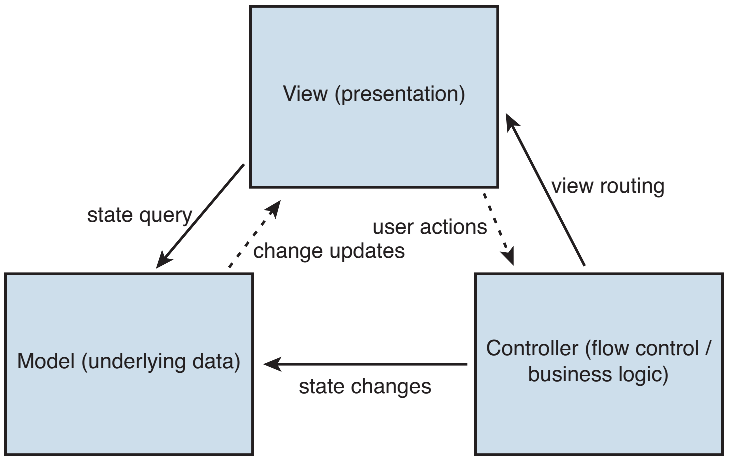

Figure 3-1 A high-level schematic depicting the basic flow between the three major components of the Model-View-Controller design pattern. All MVC web frameworks follow this basic separation of concerns to a greater or lesser extent.

Servlets

A Servlet is a server-side component designed to handle inbound service requests from remote clients. Although the vast majority of all Servlets implemented are designed to respond to HTTP/HTTPS GET and POST requests, the Servlet model is designed to accommodate any protocol that is predicated around a request/response model. Servlet developers must implement the `javax.servlet.Servlet` interface, and specifically for HTTP Servlet developers, the `javax.servlet.HttpServlet` interface. The core service method contains the routing logic that forwards the inbound request to the appropriate handler. A Servlet is hosted by the container, and multiple threads use it in order to provide a scalable system unless the developer explicitly chooses not to do this by implementing the `SingleThreadedModel` tagging interface. (This interface has been deprecated, as it results in systems that do not scale.) Figure 3-2 illustrates the Servlet lifecycle, as managed by the web container.

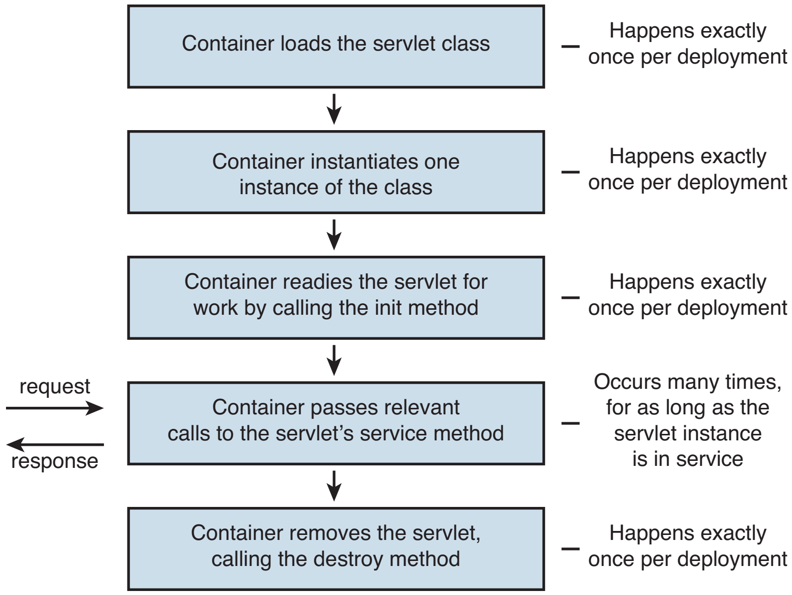
Container loads the servlet class
Happens exactly once per deployment
Container instantiates one instance of the class
Happens exactly once per deployment
Container readies the servlet for work by calling the init method
Happens exactly once per deployment
request
response
Container passes relevant calls to the servlet's service method
Occurs many times, for as long as the servlet instance is in service
Container removes the servlet, calling the destroy method
Happens exactly once per deployment

Figure 3-2 The Servlet lifecycle is quite simple, as opposed to that of other server-side components in the JEE stack. Most developers simply override three methods—`init()`, `doGet()`/`doPost()`, and `destroy()` to add required behavior.

Filters

Filters are server-side components hosted by the web container that receive an inbound request *before* it is received by any other component. Filters then are used to pre-process requests—for example, log the event, perform security checks, and so on. Filters are frequently used by web frameworks to make their operation as transparent to the developer as possible, removing or at least ameliorating a significant barrier to their adoption—the complexity (perceived or otherwise) of their development overhead. In addition, filters can be used to perform dedicated processing after a request has been received and processed.

Listeners

Listeners are server-side components hosted by the web container that are notified about specific events that occur during a Servlet's lifecycle. Listeners are used to take actions based on these events. The event model is well-defined, consisting solely of notifications on the web context (Servlet initialization and destruction, attribute adds/edits/deletes) and session activity (creation, invalidation and timeout, and attribute adds/edits/deletes).

JavaServer Pages (JSP)

JavaServer Pages are HTML pages with embedded mark-up that is evaluated at runtime by the web container to create complete HTML pages, which are sent to the client for rendering to the end user. JSP technology has matured significantly in the JEE platform—key elements added since its inception have been the JSTL (Java Standard Tag Library) and the Unified Expression Language (EL), which are covered in separate sections later in the chapter. However, from an architect's perspective, their purpose is simple—they represent the ongoing effort from Sun to enforce a workable MVC model in the JEE, separating presentation logic from business logic. Like all container-managed objects in the JEE, JSPs have a well-defined lifecycle, depicted in Figure 3-3.

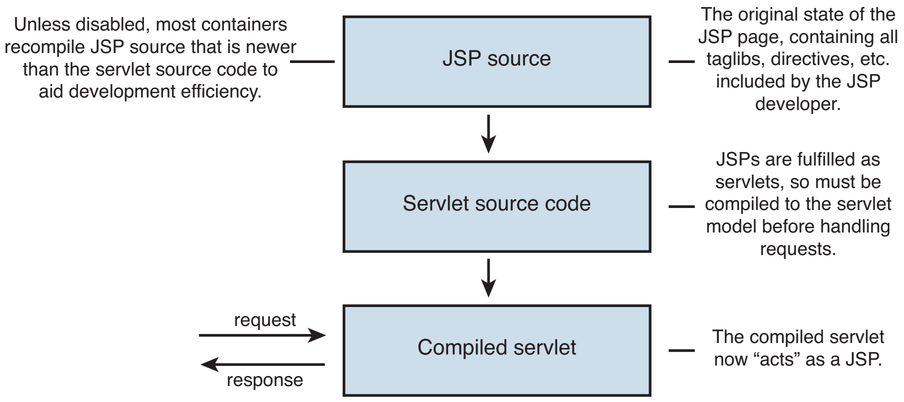
Unless disabled, most containers recompile JSP source that is newer than the servlet source code to aid development efficiency.
JSP source
The original state of the JSP page, containing all taglibs, directives, etc. included by the JSP developer.
Servlet source code
JSPs are fulfilled as servlets, so must be compiled to the servlet model before handling requests.
request
response
Compiled servlet
The compiled servlet now "acts" as a JSP.

Figure 3-3 Although JSPs appear more complex than Servlets, and represent a huge improvement on developer productivity and code maintainability, they are actually implemented as Servlets under the hood by the web container.

Java Standard Tag Library (JSTL)

The JSTL is a set of tag libraries that forms part of the JSP specification. Before the advent of the JSTL, open source communities such as Apache, commercial companies, and indeed individual software teams built their own tag libraries. The JSTL brought much needed standardization to the tag library space, allowing developers and architects to effectively delegate control and enhanced presentation logic tags to the specification writers and focus instead on their application logic. The JSTL is an example of open standards adding tangible value to developers as the JSP specification grows out to bring structure to an area badly needing it.

Unified Expression Language (EL)

The EL was introduced in the JSP 2.0 specification, whereas the JSF 1.1 specification introduced its own EL. The word *Unified* indicates that in JEE 5, these two EL definitions come together in a logical attempt to simplify the overall platform. Simply put, the addition of an EL provides developers with the ability to banish Java scriptlets from JSP pages completely. There are two constructs to represent EL expressions: ${expr} and #{expr}. $ indicates that the expr is evaluated immediately, whereas # indicates to the container that evaluation should be deferred. The container also makes a number of useful implicit objects available to an executing EL snippet—for example, requestScope, sessionScope, and so on. Access to this information further improves the ability of EL to replace custom Java snippets in JSP. Custom Java snippets on their own are not necessarily a bad thing (although the code is often more readable, elegant, and easier to maintain). The single biggest danger when developers need to code Java in JSPs is that they implement not only presentation logic, but business logic as well, violating the core tenet of the MVC design pattern.

Managing Sessions

The Servlet specification provides an elegant way to allow a client-server conversation to manage session state over the HTTP protocol, which is essentially stateless. The web container provides access to a simple map, called the `HttpSession`, where developers can read from and write to any data that needs to be stored in order to process a client's request. Judicious use of this object is needed, however—storing large objects, such as collections of search results, is a known performance and scalability anti-pattern.

JavaServer Faces (JSF)

JavaServer Faces is a UI framework for web applications based on the JEE platform. Initially controversial (and still so in some quarters), with many developers and architects resenting the imposition of yet another web framework in an already over-crowded space, JSF has grown to represent the foremost method of constructing web UIs as recommended by Sun Microsystems. Nevertheless, many application architects still eschew JSF and use a simpler MVC model, typically leveraging a web framework like vanilla Struts.

Disregarding any good or bad will toward JSF, let's examine its goals. JSF is designed to be easy to use by developers; it is also designed to allow developers to stop thinking in terms of HTTP requests and responses and instead to think about UI development in terms of user- and system-generated events. JSF components are re-usable, improving developer productivity, software quality, and system maintainability; the clear intent of the JSF specification is that the technology be toolable, or provided with deep and mature support from IDEs like Eclipse, Netbeans, and IntelliJ. In this respect, JSF maps closely onto other technologies like ASP.NET from Microsoft and, in turn, is a clear break with directions from frameworks like Ruby on Rails, where the developer is never far away or insulated from the underlying HTTP request/response model.

Templating Frameworks

Especially in the early days of JSP and even today, a segment of the developer and architect population railed against what they saw as the poor ease of development and runtime performance provided by the JSP-centric model. These malcontents fought back against the tide by using the Servlet container to build out a simpler, more efficient way of including dynamic content in HTML fragments, resulting in the creation of template-centric frameworks such as Velocity and FreeMarker. The presence of these frameworks has kept the JSP and JSF communities honest, in showing how simple web development can and should be. However, no matter how relevant or pressing the claims of these frameworks may be, the fact remains that the mandated way to build presentation logic in the JEE platform is either using JSP or JSF.

Web Frameworks

Web frameworks fill the gap between the JSP/Servlet/JSF specification and what an architect needs in order to build a consistent, high-quality web application in the UI platform. The authors are often struck, after reading a three- or four-hundred-page specification, how many open questions there are. In the case of web UIs, a good web framework fills that void, providing the architect and developer with a clear roadmap on exactly how to implement core features such as action handlers, client- and server-side validation, how to handle transactions in a sensible manner, integrate security, manage session state, and build a maintainable and understandable web UI. In fact, mainstream web frameworks have been so successful, a significant percentage of architects have decided not to use EJB in their applications at all—so confident are they that a good web framework is all that is needed to design and construct a good JEE system. And in many, if not most, cases, they are correct. EJBs in the JEE platform provide specific features that are necessary only when business requirements dictate it (these features are detailed in Chapter 4, along with the decision matrix governing when the use of EJBs is appropriate). If you choose not to specify or use a web framework in Part II of the exam, be prepared to clearly justify your decision. We believe that very few, if any, non-trivial Java projects are not using a web framework to impose standard practices on the development team, to produce maintainable code, and to avoid re-inventing the wheel on every new development.

Discussion

In this section, we examine the best uses for each of the various components of the JEE web technology stack. Almost all the components can be used to tackle any presentation/flow control/business logic problem, but the specifics of JSPs, Servlets, JSF, and so on mean that they each are better-suited to specific scenarios, as detailed here.

JSPs and Servlets—Standard Uses

JSPs handle the presentation of data to the end user. They should contain no business logic. A good rule of thumb is to minimize or eliminate entirely all Java code from JSPs and replace it instead with either

EL, the JSTL, or a custom/third-party tag. This guideline tends to reinforce the role of JSPs as the V in MVC—that is, the View.

JSF—Standard Uses

The standard uses for JSF are the same as for JSP. As an architect, you are faced with a choice: either continue to use JSP with JSTL and a good MVC framework, or use JSF. They do the same thing. Also, they are not mutually exclusive. It is perfectly possible to add tags to a JSP page that represent a specific JSF UI component, resulting in a hybrid solution. JSF garnered a significant amount of bad press when it first launched (as have many 1.0 implementations of specifications in the JEE platform), but it has matured since then. Many architects, however, simply see no need for it and prefer JSP with JSTL and EL.

Web-Centric Implementations

As intimated earlier, a significant proportion (exact figures are not available and indeed vary by industry vertical) of all JEE applications in existence today are deployed using only a web container—that is, they do not use EJBs. This class of JEE application is termed web-centric.

The current version of the exam tests this concept in detail. As a JEE architect, you are perfectly entitled to stipulate that EJBs not be used in your design, but you must clearly understand why that decision is mandated and the impact of that decision on your developers as they implement the business logic. The exam tests this concept by presenting you with a set of scenarios. Scenarios that have a strong messaging, transaction, or security management component are all candidates where an EJB-centric implementation is warranted and indeed necessary. (Let's be blunt—choosing EJB is the right answer.) Scenarios where ease of development is key, where an existing application is already web-centric, or where transactions are not key to the business (read-only or read-mostly) mean that you should choose a web-centric answer from those provided in the exam.

There are some stand-out reasons where using EJB is simply not warranted. The most straight-forward example is a standard Create, Read, Update, and Delete (CRUD) application built using Struts to organize and control the presentation and business logic tiers, and Hibernate plus a DAO access layer to implement the persistence tier. Assuming that there are no asynchronous messaging requirements or

JMS queues or topics to access, and that the functionality contained in the web container for concurrency control, security, and session management is sufficient, then the right decision is to adopt a web-centric approach.

Now, let's consider an alternative scenario. You work for XYZ Bank, a large multinational bank with investment and retail operations, which has invested significant amounts of capital into a transactional system based on mainframe technology over the last thirty years. Ensuring system reliability and security are paramount; there is absolutely no room for data corruption from edge conditions, such as the lost update or optimistic locking going wrong. If the system enters into an unknown state because of a technology failure, not only will the system need to be brought back within 10 minutes in order to avoid a service-level agreement (SLA) breach, the relevant regulatory authorities must also be notified and a full system audit will be enforced. As the solution architect, do you believe that using only the web container segment of the JEE platform is sufficient to meet the non-functional requirements detailed here?

We would answer this rhetorical question as follows: It is possible to fulfill the preceding scenario using only a web framework, but we would not be comfortable in doing so. Many aspects of the EJB framework lend themselves very well to this type of deployment; choosing to use only a web framework will essentially force you, as the architect, into replicating in your code the reliability and availability characteristics that already exist in the core JEE platform. This is not a good use of your time and will result in a buggier implementation that needs to be maintained moving forward.

EJB-Centric Implementations

Let's reconsider the bank scenario laid out in the previous section. Looking at the business requirements, we can see that they translate into nonfunctional requirements (NFRs) focusing on system correctness, reliability, and security. In this scenario, and answering the question posed in the last section, assuming that the internal bank systems can be accessed by a non-EJB solution, it is possible to achieve a solution that will meet the NFRs using only a web-centric solution. But, and this is the key point, you will need to commit your team to writing entire modules of custom code to replace features that you get from an EJB container for free. In addition, it is likely that you will also need to take advantage of vendor-specific libraries/mechanisms to implement these modules. That is the key point. In the scenarios examined here, there is no right or wrong answer—just more correct and less correct. And that is the key role of an architect: to examine the possible solutions and select the most correct solution, taking into account the vagaries of the known set of business requirements.

Rationale for Choosing Between EJB-Centric and Web-Centric Implementations

As you may have gathered from the two preceding sections, neither we, nor indeed the exam, believe that a web-centric or an EJB-centric architecture is always right or always wrong. The decision to select one over the other is based purely on an impassionate review of the facts relating to a specific project. In order of decreasing importance, the pertinent facets to consider are as follows:

- Transaction requirements—The more onerous, the bigger the reason to select EJB.
- Security requirements—Again, the more onerous, the bigger the reason to select EJB.
- Messaging requirements—Need to integrate with an asynchronous messaging system—Again, if present, a clear reason to select message-driven beans (MDBs); that is, the EJB-centric approach.
- Performance.
- Ease of development.
- Scalability.
- Existing team skills or existing project implementation.

The last four facets listed are not reasons in themselves that will conclusively force you to choose one approach over the other; indeed, the waters have been muddied in recent JEE releases for each. The primary focus for EJB 3.0 (and continued in 3.1) is improving the ease of development. As you will see in Chapter 4, the general consensus is that EJBs are now, at last, easy enough to develop that their use is warranted in situations where previously system designers did not specify their use. Assuming an efficient container implementation, stateless session beans should be as efficient as Servlets/Action handlers in executing business logic on the server side as a proxy for the client. The obvious exception here is stateful session beans. The need to maintain one session bean per connected client for the duration of the conversation will always make stateful session beans a poor scaling design choice, suitable only for a small subset of applications with very specific requirements.

The Future of Client-Server Communication

It is worth noting that the current release of the exam was written in 2007 and contains material on Asynchronous JavaScript and XML, or AJAX. Architects must understand the benefits of AJAX as they relate to providing an enhanced end-user experience and how the JEE 5 platform allows server-side components to service AJAX requests from browsers. Looking forward, the exam will be refreshed in sympathy with the release of future JEE versions. If JEE 6 or 7 is released into a world where AJAX is declining in favor of cometd (HTTP continuations), or another way of enhancing the end-user experience for browser-based applications, then expect that technology to be reflected in the questions posed. After all, the exam is written by a team of subject matter experts who construct the questions and answers for Part I based on the current state of play in the Enterprise Java space.

Essential Points

- Presentation tier technologies remain a major element of the JEE 5 platform and are a significant source of exam content for Parts I and III.
- Part II is less concerned with the actual presentation technology selected (within reason, of course) and more concerned with the candidate displaying two things—in-depth understanding of the business requirements and selecting a presentation technology that meets those requirements.
- JSF has grown from the presentation tier that disgruntled architects tried to ignore to a significant element of the JEE platform—and for the exam. If you are a JSP-centric architect, beef up on JSF because you need to know it.

- The exam tests your understanding of the best UI technologies to use in the JEE platform by presenting a series of scenarios. The description of the scenario provides all the information you need to select the correct technology/combination of technologies to use from the multiple choice answers provided.
- In the real world, there are no official "Sun recommended" blueprint patterns—only guidelines and recommendations. As a JEE architect, one of your key skills is the ability to analyze application requirements and choose the best combination of JEE technologies—especially at the web tier—to meet those requirements, while not over-engineering the solution.

Review Your Progress

These questions test your understanding of JEE web components and their most appropriate use to solve a given business problem:

1. You are the architect at a large investment bank. Your main area of responsibility is a new web application designed to replace the aging user interface for the existing clearing house back office system. One of the systems is read from/written to via a JMS Queue in asynchronous fashion and transactions and security management are paramount. Select the most appropriate implementation from the following list:

A. JSP and JSTL accessing a business logic tier built using EJBs and MDBs.
B. JSP and JSTL accessing a business logic tier built using MDBs only.
C. JSF accessing the systems directly.
D. JSP accessing the systems directly.

Answer: A. B is not flexible enough, omitting EJBs and allowing only MDBs. C and D couple the presentation tier directly to the backend resource, creating potential security, performance, and maintenance problems. A provides what is needed.

2. You are the architect at ACME Corporation—the hottest Internet start-up of the moment. The start-up provides a front-end accessible by multiple devices, from smart phones to desktops, and provides innovative social networking features to its members. The key considerations for the system are performance and scalability, and individual messages between members are not considered important (that is, they can be resent). Select the most appropriate implementation for this system from the following list:

A. JSP + JSTL accessing the messaging layer directly.
B. JSF accessing EJBs, with access to the messaging layer mediated by a JMS client and MDB.
C. JSF accessing stateful session beans—one for each connected client.
D. JSP + JSTL accessing a JPA-based persistence tier.

Answer: A. All of the other options contain a reasonable chance that there will be an unnecessary overhead associated with the components used—EJBs, JPA, and so on. A is the simplest answer for the business problem described, especially when the priority of performance and scalability is stated in the stem of the question.

3. You are a subject matter expert on JEE consulting for ACME Corporation. ACME has an existing application built using an earlier version of the JEE platform. Performance and scalability are not an issue, although system is not as maintainable as ACME would like. The application uses JSP pages as part of a Model 2 MVC architecture with Java code in the JSPs and some presentation coded as Servlets. What do you recommend?

A. A complete rewrite of the existing presentation architecture to leverage JSF and JPA.
B. A deeper analysis of the current system to ensure that JEE best practices (especially the MVC model) are respected throughout the code, replacing Java code in JSPs with JSTL and EL as necessary and making Servlets act purely as controllers.
C. ACME move the system to use Ruby on Rails.

D. A complete rewrite of the current architecture to leverage JSF, session beans, JMS, and JPA.

Answer: B. All of the other answers are nonsensical when you realize where ACME is. They have a system that works today, which requires some refactoring to move to MVC, and they simply need a roadmap after this work is completed to guide them onto JEE 6, 7, and beyond. No rewrites are necessary.

4. You are a JEE architect at ABC Bank and have been tasked with designing their next-generation UI framework for online banking. The online banking application must be accessible by both standard browser clients and mobile devices. What do you recommend as the simplest and most optimal solution?

A. A JSF-based architecture, leveraging the capability of device or channel-specific JSF renderers to support both mobile and standard browser clients.
B. A JSP-only architecture, with custom logic to probe and handle individual devices at runtime.
C. A Servlet-based architecture.
D. A template-based architecture.

Answer: A. JSF is designed to support exactly this type of use case—the other available options, while workable, are not the most optimal or most simple.

5. XYZ Corp has retained you as the architect for their latest web application: XYZOnline. This application allows customers to search, browse, and order catalog content online. XYZOnline accesses the inventory and payment systems as web services. What architecture do you recommend?

A. JSP/JSF pages accessing the web services layer using stateless session beans.
B. Servlets accessing the web services directly using JAX-WS as necessary.
C. JSP/JSF pages accessing the web services layer using JAX-WS as necessary.
D. JSP/JSF pages accessing the web services using JMS.

Answer: C. A uses stateless session beans when nothing in the description warrants their usage. B uses Servlets to generate the presentation, while D uses JMS in the wrong context. C is the best solution for the stated business requirements.

6. You have been asked to evaluate multiple web presentation technologies for ABC Corp. Their priorities are future-proofing, tooling support from IDEs and the ability to render multiple versions of the same component for different devices. What do you recommend to ABC?

A. Use JSF components as part of a Servlet.
B. Use JSTL and the EL as part of JSP pages.
C. Use JSF components as part of JSP pages.
D. Use JSTL and the EL as part of Servlets.

Answer: C. The key to choosing C is to realize that the question guides you there by mentioning tooling support and future proofness. B is close but does not match the requirements exactly. D and A are not valid answers.

Index